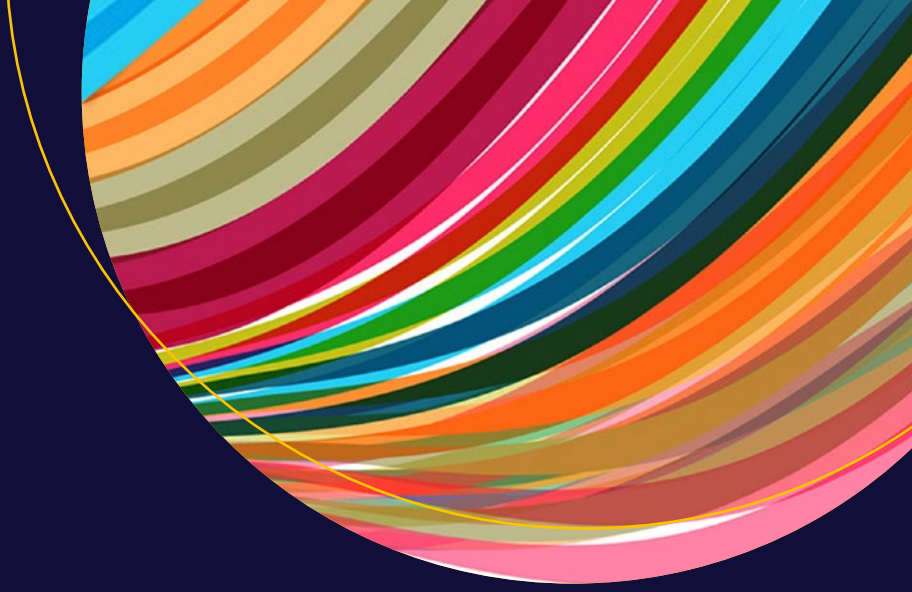

VFX Fundamentals

Visual Special Effects Using Fusion 8.0

Wallace Jackson

Apress®

VFX Fundamentals

Visual Special Effects Using Fusion 8.0

Wallace Jackson

Apress®

VFX Fundamentals: Visual Special Effects Using Fusion 8.0

Wallace Jackson
Lompoc, California
USA

ISBN-13 (pbk): 978-1-4842-2130-3 ISBN-13 (electronic): 978-1-4842-2131-0
DOI 10.1007/978-1-4842-2131-0

Library of Congress Control Number: 2016948843

Managing Director: Welmoed Spahr
Lead Editor: Steve Anglin
Technical Reviewer: Chád Darby
Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Mark Powers
Copy Editor: April Rondeau
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com/9781484221303. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

This book is dedicated to affordable software developers like Blackmagic Design as well as to all members of the open source new media software community who are working so diligently to make professional new media application development software, audio video, 2D image compositing, digital painting, and 3D content development tools freely available for new media application developers so that they can utilize these tools to achieve their creative dreams and their financial goals. Last, but not least, I dedicate this book to my loving father, Parker Jackson; my family; my life-long friends; my content production ranch neighbors; and my business partners for their continual help, assistance, and those relaxing, beautiful sunset BBQs underneath pink clouds on Point Conception.

Contents at a Glance

Contents

About the Author

Wallace Jackson has written for several leading multimedia publications about production for the media content development industry since contributing an article about advanced computer processing architectures for the centerfold (a removable "mini issue" insert) of the original issue of *AV Video Multimedia Producer* magazine, which was distributed at the SIGGRAPH trade show.

Jackson has written for a number of popular publications regarding his work in interactive 3D and new media advertising campaign design, including *3D Artist*, *Desktop Publisher Journal*, *CrossMedia*, *Kiosk*, *AV Video Multimedia Producer*, and *Digital Signage*, as well as for many other publications.

Wallace Jackson has authored more than twenty Apress book titles, including several titles in the ever popular Apress *Pro Android* series, Java and JavaFX game engine development titles, digital image compositing titles, digital audio editing titles, digital video editing titles, digital illustration titles, VFX special effects titles, digital painting titles, Android 7 new media content production titles, and JSON and HTML5 titles.

In this current book on digital video editing and effects, Wallace focuses on Blackmagic Design Fusion 8 digital visual effects software, using it to demonstrate digital video special effects as well as digital video editing and compositing fundamentals to beginners who want to become more digital effects editing–savvy.

Wallace is currently the CEO of MindTaffy Design, the new media advertising agency that specializes in new media content production and digital campaign design and development. It is located by La Purisima State Park in Northern Santa Barbara County on the Point Conception Peninsula, halfway between their clientele in Silicon Valley to the north and Hollywood, the OC, West Los Angeles, and San Diego to the south.

Mind Taffy Design has created open source, technology-based (HTML5, JavaScript, Java 9, JavaFX 9, and Android 7.0) digital new media i3D content deliverables for more than a quarter century, since January 1991.

The company's clients consist of a significant number of international brand manufacturers, including IBM, Sony, Tyco, Samsung, Dell, Epson, Nokia, TEAC, Sun Microsystems (Oracle), Micron, SGI, KDS USA, EIZO, CTX International, KFC, Nanao USA, Techmedia, EZC, and Mitsubishi Electronics.

Jackson received his undergraduate bachelor's degree in Business Economics from the University of California at Los Angeles, or UCLA, and his graduate degrees in MIS/IT, Business Information Systems Design, and Implementation, from University of Southern California, located in South Central Los Angeles (USC).

He also received post-graduate degrees from USC in Entrepreneurship and Marketing Strategy and completed the USC Graduate Entrepreneurship Program. Jackson earned his two USC degrees while at USC's evening Marshall School of Business MBA Program, which allowed him to work full time as a COBOL and RPG-II programmer while completing his business and IT degrees.

You can visit Wallace's blog at `www.wallacejackson.com` to view his multimedia production content. You can also follow him on Twitter at `@wallacejackson` or connect with him on LinkedIn at `LinkedIn.com/in/wallacejackson`.

About the Technical Reviewer

Chád ("Shod") Darby is an author, instructor, and speaker in the Java development world. As a recognized authority on Java applications and architectures, he has presented technical sessions at software development conferences worldwide (in the United States, United Kingdom, India, Russia, and Australia). In his fifteen years as a professional software architect, he's had the opportunity to work for Blue Cross/Blue Shield, Merck, Boeing, Red Hat, and a handful of startup companies.

Chád is a contributing author to several Java books, including *Professional Java E-Commerce* (Wrox Press), *Beginning Java Networking* (Wrox Press), and *XML and Web Services Unleashed* (Sams Publishing). Chád has Java certifications from Sun Microsystems and IBM. He holds a bachelor's degree in Computer Science from Carnegie Mellon University.

You can visit Chád's blog at `www.luv2code.com` to view his free video tutorials on Java. You can also follow him on Twitter at `@darbyluvs2code`.

Acknowledgments

I would like to acknowledge all my fantastic editors and their support staff at Apress who worked long hours and toiled diligently on this book to make it the leading visual effects fundamentals title currently available in the marketplace.

I would like to thank the following people:

Steve Anglin, for his work as the acquisitions editor for this book, and for recruiting me to write development titles at Apress that cover widely popular open source content development platforms (Android, Java, JavaFX, HTML5, CSS3, JS, JSON, etc.).

Matthew Moodie, for his work as the development editor on this book, and for his experience and guidance during the process of making this book one of those fantastic digital visual effects editing, image compositing and 3D special effects titles.

Mark Powers, for his work as the coordinating editor for this book, and for his constant diligence in making sure that I either hit my chapter delivery deadlines or far surpassed them.

Chád Darby, for his work as the technical reviewer on this book, and for making sure that I didn't make technical mistakes.

Introduction

VFX Fundamentals is intended for the digital artist, digital videographer, multimedia producer, illustrator, application developer, website developer, user interface design architect, user experience designer, social media user, effects compositor, matte painter, or just about anyone who is interested in generating superior quality digital visual effects and 3D VFX, and delivering them in popular MPEG4 and MOV video data formats.

This book covers digital visual effects concepts, editing, special effects, titling, bluescreen mattes, masking, particle systems, physics, motion tracking, compositing, blending modes, 3D, and more. This equates to digital video and special effects fundamentals combined into one unified title that includes technical terms, topics, concepts, and definitions.

Each chapter will build upon the knowledge learned in the previous chapter. Thus, later chapters in the book will have readers creating advanced digital visual effects projects using mattes, motion tracking, image compositing, special effects algorithms, 3D particle systems and similar VFX software features, dialogs, and tools, all using Fusion's powerful visual programming language.

There is even coverage at the end of this book regarding content publishing across consumer electronics devices via the use of open source platforms such as Java 9, JavaFX, HTML5, CSS3, iOS, Android 7.0, Kindle, Fire OS, EPUB, PDF, and JavaScript.

In Chapter 1, you will install open source **Fusion 8** as well as other related 3D and new media software packages that would be useful for your digital visual effects production workstations, including the impressive Blender 3D 2.8 for modeling, rendering, and animation and Editshare Lightworks 12.6 for collaborative digital video editing. You'll also download and install GIMP 2.

In Chapter 2, you will take a look at **raster** imaging and video concepts, like pixels, resolution, aspect ratios, compositing, color channels, alpha channels, masking, blending modes, and so forth. This information will provide your VFX technical foundation.

In Chapter 3, you will build on the raster concepts by adding **motion** to the pixels and by covering digital video or 2D/3D animation concepts such as codecs, frame rates, keyframes, bit rates, and similar concepts used for visual effects animation workflows.

In Chapter 4, you will learn about **digital audio** content production, since audio is a primary component of digital video and visual FX. You'll cover sound waves and sampling, amplitude and frequency, MIDI and Digital Audio data formats, and sound design concepts and fundamentals, all in one, unified chapter.

In Chapter 5, you'll explore the mathematical world of **2D vector** graphics, including points, lines, curves and paths, and the SVG, or Scalable Vector Graphics, format, and how these are used and even animated in Fusion visual effects processing pipelines.

In Chapter 6, you will be introduced to the next level of **i3D vector** graphics, including 3D modeling, 3D rendering, animation, and programming. This is important for visual effects, as Fusion 8 contains both 2D and 3D (both unified) compositing pipelines.

In Chapter 7, you will take a look at the core user interface, features, and approach of Fusion 8. You will get a bird's eye tour of the entire software package, as well as an overview of how each part of this software works together with the other components.

In Chapter 8, you will start to delve into the inner workings of Fusion 8, starting with the all-important visual algorithmic editing language utilized in the Fusion **Flow Node Editor**. Nodes and VFX tools and tiles are covered, as you learn how to create VFX processing pipelines using Hollywood studio–level software.

In Chapter 9, you will cover another core Fusion feature, the **Timeline Editor**, as well as the **Time Ruler**, where you will control the fourth dimension of time for all your VFX projects. We'll look at keyframing and the playback head and learn how to animate any VFX tool, setting, and attribute in your VFX project pipelines, increasing your VFX power by an order of magnitude.

In Chapter 10, you'll learn about Fusion's **Spline Editor**, which allows you to create vector assets such as mattes, masks, and LUT (Look Up Table) control curves, thus allowing your advanced VFX visual control representations to be edited mathematically.

In Chapter 11, you'll continue learning about how to best utilize Spline Editor in Fusion by taking a look at **animated polyline masking**. We'll look at how to create morphing and warping masks using animated polylines, as well as animated blend areas using double polyline masking techniques.

In Chapter 12, you will learn how to leverage Fusion chroma or luma keying algorithms to have Fusion 8 create your mattes and masking for you by utilizing popular bluescreen or greenscreen technology and workflows. We will look at several of Fusion's keying algorithms, including **Primatte V**, **UltraKeyer**, and **ChromaKeyer**.

In Chapter 13, you'll learn Fusion's workflow for another popular VFX software feature, called **motion tracking**. We will take a look at single-point motion tracking, as well as at more complex multi-point motion tracking, using public digital video assets. Motion tracking is a popular VFX topic allowing VFX artisans to bridge 3D software packages with VFX software such as Fusion 8.

In Chapter 14, you'll learn all about the **i3D compositing** engine in Fusion, which we will cover over the next six chapters of the book. We will cover how to import 3D models into Fusion using an FBX file format. We will also look at how to set up cameras, lighting, and texture maps with these 3D assets by using geometry, shaders, materials, and textures that are rendered in real-time. This is done using advanced GPU hardware rendering support that allows VFX artisans to work in real-time, seamlessly fusing 2D VFX pipelines and 3D VFX scenes together.

In Chapter 15, you will get more advanced in Fusion 3D VFX by learning about how **Shaders** are constructed using your materials algorithm in conjunction with texture mapping. You'll dissect advanced Shader pipelines in Fusion so you learn how to create custom advanced 3D Shader pipelines for your own 3D VFX.

In Chapter 16, you will learn about how to model 3D text title assets in Fusion, which can be used in advanced **3D text titling** animated sequences, which are popular in VFX project workflows. We cover extrusion, beveling, chamfering, advanced profiles and selective face texture mapping, and 3D text object lighting.

In Chapter 17, we will continue with 3D text titling and create an **animated 3D text titling** project by using the 3D text model created in Chapter 16. We will look at Fusion modifiers, which allow VFX artisans to create animation using algorithmic tools and custom settings with the Flow Node Editor tools and their control panel settings.

In Chapter 18, you will learn about Fusion's **particle systems**. Particles allow advanced visual effects, such as rain and snow, explosions, fluid dynamics, crowd simulations, flocking, sparks and fire, and just about

anything else you can imagine. I will cover particle emitters and particle effects, particle rendering, and the difference between Fusion 8's 2D and 3D particle systems.

In Chapter 19, you will get into advanced particle systems by covering the **particle physics** algorithms found in Fusion, as well as by using Fusion modifiers in particle systems. We will continue with the advanced version of your VFX particle simulation from Chapter 18 and apply digital imaging techniques and matte and masking effects to particle systems, assembling your complex 20-node Fusion VFX processing pipeline, spanning the two chapters.

In Chapter 20, I will cover **publishing** 2D or 3D visual effects content deliverables using leading content-delivery platforms and popular hardware devices, from smartwatches to UHD 4K iTV sets to legacy HD 2K iTV sets and everything in between, including eBook eReaders, tablets, gaming consoles, automobile dashboards, home appliances, IoT devices, and UHD smartphones.

If you are interested in digital visual effects, 2D image compositing, and 3D special effects, and you want to learn about each of these areas along with the basic fundamentals behind them, as well as about how everything works together in the visual effects domain, this is the book that you need to begin your journey with.

This book is overflowing with tips, tricks, tools, topics, terminology, techniques, concepts, and work processes. Indeed, this book should give you the boost to transition from being a visual effects compositing beginner to being that knowledgeable digital visual effects professional you seek to become, at least where a digital video, 3D VFX, or 2D VFX visual effects compositing pipeline is concerned. This book is a great follow-up to the earlier titles in the new media fundamentals series from Apress.

Chapter 1

Visual Effects: Set Up Your VFX Content Development Workstation

In this first chapter, let's put together your foundation for a highly professional visual effects (VFX) content development workstation. This is an important fusion of hardware and professional software, allowing you to reach your goal of doing interactive multimedia production VFX development. Let's spend a chapter considering your new media hardware needs and the software infrastructure that you will want to put together to create a professional, well-rounded visual effects software development workstation. This will put a bunch of arrows in your multimedia software development quiver right off the bat. You will then have everything you need to develop new media content for use in your visual effects pipeline throughout this book, no matter what type of VFX applications you decide you want to develop for your VFX clients!

We will get all of these tedious setup tasks out of the way, putting together a professional new media content production workstation using professional open source software.

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4842-2131-0_1) contains supplementary material, which is available to authorized users.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_1

If you already have your workstation configured, you can proceed to Chapter 2 for an overview of raster concepts. If you are already familiar with these raster concepts, proceed to the rest of the book. Since we want the readers of this book to use similar visual effects development environments, we'll outline all of the steps involved in putting together an enviable new media production workstation. You'll learn where to go to download and how to install several of the most professional open source software packages on the face of this planet. You are about to max out your new VFX workstation!

The VFX Content Production Workstation

The first thing that we'll do after taking a look at hardware requirements is to download and install visual effects new media asset development tools. These are used in conjunction with VFX software package Fusion 8.0 for things such as image editing (GIMP); non-linear digital video editing (Lightworks); digital audio sweetening or editing (Audacity); i3D modeling, rendering, and animation (Blender); digital illustration (Inkscape); and business productivity (OpenOffice). This chapter will take your development to an all-new level, showing you how to create the new media development and VFX workstation that will run your visual effects content development business.

All of these software development tools, which you'll be downloading and installing, will come close to matching all the primary feature sets of expensive paid software packages, such as those from Apple (Final Cut Pro), Autodesk (3D Studio Max), Adobe (Photoshop, Illustrator, After Effects), and Avid (ProTools), and all at ZERO cost to your production company!

Open source software is free to download, install, and even upgrade, and is continually adding features. It's becoming more and more like professional software every day. You will be completely amazed at how professional your open source software packages have become over the last decade or so.

New Media Content Production: Hardware Is Key!

Since during this chapter you will put together what will be the foundation for the VFX content production workstation you will use for the duration of this book, I want to take a moment to review new media content development workstation hardware requirements first. This will influence your VFX development performance (speed). This is clearly as important as the software itself, since hardware is what is actually running the VFX software package's algorithms.

Let's discuss what you need to make a multimedia content production workstation usable. You will need an **HDTV** (1920 x 1080 resolution using a 120FPS refresh rate) or **UHD** (4096 x 2160 at a 120FPS refresh rate) **widescreen display**. These are affordable and give you four to sixteen times the display "real estate" that the XGA displays typically provided. HDTVs are now $140 to $300; UHDTVs are now under $1,000. Two displays are always better than one!

I recommend using, at a bare minimum, the **Intel i7** quad-core processor or the **AMD 64-bit** octa-core processor. Install at least 16GB of DDR3-1600 memory; 32GB or 64GB would even be better. I'm using a 64-bit AMD octa-core with 16GB of DDR3-1600.

Intel also has a new hexa-core i7 processor. This will be the equivalent of having **twelve cores**, as each i7 core can host two threads. Similarly, the i7 octa-core will look like sixteen cores to a 64-bit operating system's thread-scheduling algorithm.

There are also high-speed DDR3-1866 as well as DDR3-2133 clock-speed memory module components available. A high number signifies fast memory access speeds. To calculate the actual megahertz speed the memory is cycling at, divide the number by 4 (1333 = 333 MHz, 1600 = 400 MHz, 1866 = 466 MHz, 2133 = 533 MHz).

Memory access speed is a massive workstation performance factor, because your processor is usually "bottlenecked" by the speed at which processor cores can access the data (in memory) that it needs to process your VFX algorithms.

With high-speed processing and memory access going on inside the workstation while it is operating, it's extremely important to keep everything cool so that you do not experience **thermal problems**. I recommend using a wide, full-tower enclosure with **120mm** or **200mm** cooling fans (one or two at least), as well as a captive liquid induction cooling fan for your CPU.

It is important to note that the cooler the system runs, the faster it can run, and the longer it will last, so load the workstation up with lots of silent, high-speed fans!

If you really want to maximize performance, install an SSD (solid state disk drive) as your primary disk drive that your applications and operating system software load from. Use legacy HDD hardware for your D:\ hard drive for the slower data storage (long term). Put your current project files on the SSD.

Finally, make sure you have a modern GPU graphics card in your system's PCI slot, as Fusion 8 can offload complex processing of calculations to the dozens, if not hundreds, of parallel DSPs that this card is hosting—calculations that are central to VFX software such as Fusion 8. Fusion forums recommend at least a 2GB DDR5 nVidia GTX 660, which is less than $200 on PriceWatch.com, or a 4GB DDR5 GTX 670, which is less than $350. If you've got a cool G ($1,000) to spend, get a 12GB DDR5 GTX Titan X from nVidia and call it a day.

As far as the OS goes, I'm using a 64-bit Windows 10 operating system, which is fairly memory efficient. The Linux 64-bit OS is extremely memory efficient. I recommend using any 64-bit OS so you can address more than 3GB of system memory. The more memory you have to work with the better, so try and get 64GB or even 128GB into your workstation. Fusion 8 is going to support Mac, Windows 10, and Linux (I use Ubuntu Mate and SUSE), so you can use the OS that you prefer as long as it is 64 bit and up to date! The Linux and Fusion software are both free, so spend your money on hardware if you have a limited budget, as VFX is very processor intensive and requires lots of fast memory and processor cores!

VXF Multimedia Asset Development: Open Source

To create a professional and well-rounded VFX content development workstation, you will be installing all of the primary genres of software. I'll show you how to download and install Fusion, GIMP, Blender, Inkscape, Audacity, Lightworks, and Open Office, all of which are highly professional open source software packages. You will be putting together a 100 percent open source workstation. Open source software has reached the same level of professionalism as paid software packages, which can often cost more than a thousand dollars each. By using open source software packages, you can put together VFX content development workstations (as many as you need) that will rival any paid application development software workstation. Let's get started!

Visual Effects Production: Fusion 8

BlackMagic Design Fusion 8.0 used to be called Eyeon Fusion, and it used to cost thousands of dollars. Now it is available for free, with a "Studio" version offered for around a thousand dollars. Fusion offers a professional visual effects software package that can be used for film and television, eBooks, smartphone applications, iTV set programs, smartwatch face design, virtual reality simulations, and other new media endeavors. Download Fusion 8.0 now at https://www.blackmagicdesign.com/products/fusion/.

Once you get to the Fusion 8 homepage, click the blue **Download** button, which can be seen in Figure 1-1, to start the download process. Once you have the software in your `Downloads` folder, copy it to the `Software` folder where you keep installers. Unzip the software package into its own folder.

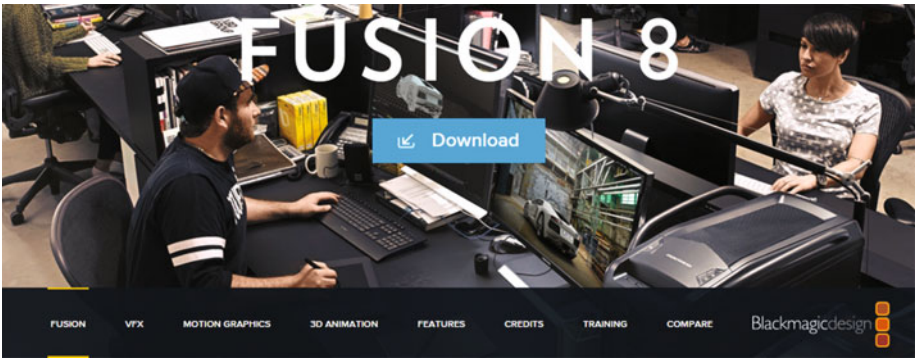

Figure 1-1. *Go to blackmagicdesign.com and download Fusion 8*

Figure 1-2 shows this `C:\Software\Blackmagic_Fusion_Windows_8.0` folder. Find the `Install Fusion v8.0.msi` file and right-click on it; select the **Install** option to start the installation process.

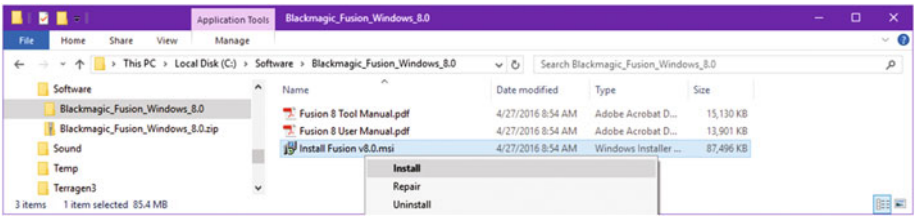

Figure 1-2. *Right-click Install Fusion .MSI, and select Install*

Accept the license and click **Install**, as seen in Figure 1-3.

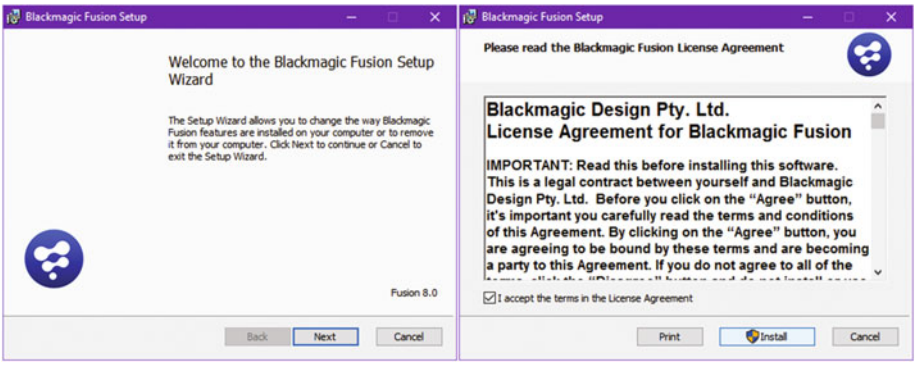

Figure 1-3. First two dialogs in the Fusion 8 install process

When install completes, click **Finish**, as seen in Figure 1-4.

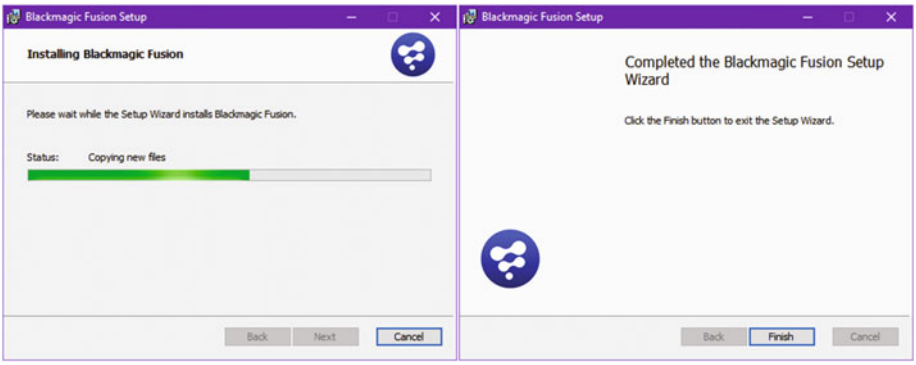

Figure 1-4. Last two dialogs in the Fusion 8 install process

Once Fusion 8 installs, pin the icon to your taskbar for use in Quick Launch and start the software. This will make sure that Fusion can load, and that the software can start and run successfully. Figure 1-5 shows a Fusion initialization screen.

Figure 1-5. Launch Fusion 8 to make sure it installed correctly

VFX Content Production: New Media Apps

Next, let's install new media content production applications, which can be used in conjunction with your VFX project pipeline, so that you have a complete set of working tools for your VFX content development work process.

GIMP 2.8: Digital Image Editing and Compositing

The GIMP project offers a professional imaging software package for digital image editing. Download GIMP at `http://www.gimp.org` and install it. GIMP is currently at version 2.8.18. The GIMP homepage, with its red Download button, is shown in Figure 1-6.

Figure 1-6. Go to `www.gimp.org` and download GIMP 2.8.18

If you want to learn digital image compositing, check out the *Digital Image Compositing Fundamentals* book (Apress, 2015).

Blender: 3D Modeling, Rendering, and Animation

The Blender Foundation project offers professional i3D software, allowing you to model 3D objects as well as render and animate them. Download and install Blender at `http://www.blender.org`. Blender's homepage and blue Download button can be seen in Figure 1-7.

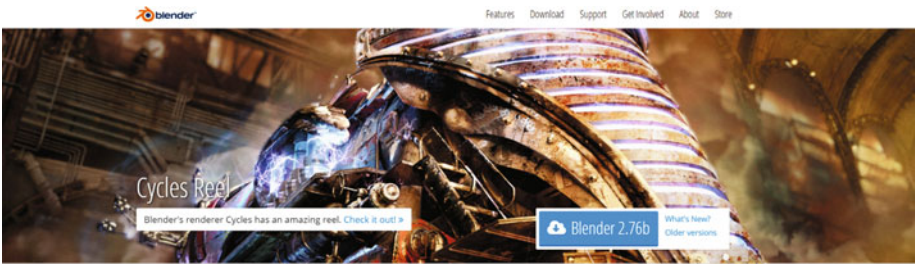

Figure 1-7. Go to blender.org and download the latest version

This is a professional-level software package with many of the same features as 3D Studio Max, Maya, XSI, and Lightwave.

Inkscape: Digital Illustration and Digital Painting

The Inkscape Project offers a professional digital illustration software package. Download and install the software at `http:www.inkscape.org`. Inkscape's homepage and **Download** button can be seen in Figure 1-8.

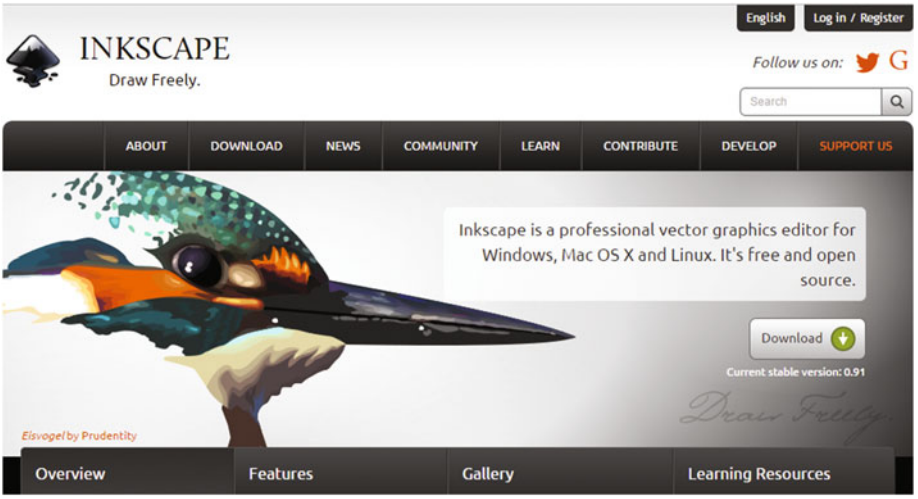

Figure 1-8. Go to inkscape.org and download the latest version

If you want to learn digital illustration, check out the *Digital Illustration Fundamentals* book (Apress, 2015), as well as *Digital Painting Techniques* (Apress, 2015).

Audacity: Digital Audio Editing and Special Effects

The Audacity Team offers a professional digital audio editing software package called Audacity for digital audio editing, sweetening, and special effects. You can download and install this software package at `http:www.audacityteam.org`. The Audacity homepage and "Download Audacity 2.1.2" link are shown in Figure 1-9.

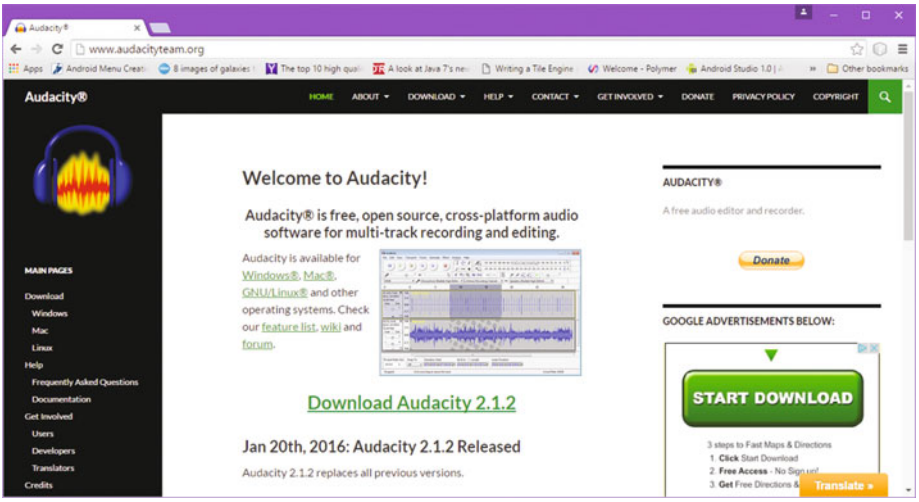

Figure 1-9. *Go to audacityteam.org and download version 2.1.2*

Audacity offers many of the same digital audio editing features as the professional audio editors have, and is adding 64-bit capabilities and professional features every month. The next major version will even have a more professional user interface look and feel. If you want to learn more about digital audio editing, synthesis, and special effects, check out the *Digital Audio Editing Fundamentals* (2015) title from Apress, available at `http://www.apress.com`, when you have a moment. Digital audio can greatly enhance the experience in your VFX content production pipeline, as audio can have a major impact.

Next, let's take a look at a free digital video editing software package called Lightworks 12.6 offered by Editshare. This software package has been used to create a large number of feature films, so I am not setting you up with just any software package, but rather with software that has been used for professional film, television, and digital VFX development.

Digital Video Editing: Editshare Lightworks 12.6

Editshare Lightworks offers professional digital video editing in a software package that can also do special effects. Download the software package at `http:www.lwks.com` after signing up for the download. Lightworks' homepage and Downloads tab can be seen in Figure 1-10. Select the proper version for your OS. I recommend using a 64-bit OS and 64-bit software so that you can use 16MB of memory. Like Fusion, Lightworks also uses your 2GB DDR5 GPU!

Figure 1-10. Go to lwks.com and download Lightworks for your OS

Office Productivity Suite: Apache OpenOffice 4.1.2

Apache OpenOffice, originally Sun Microsystems' StarOffice, was acquired by Oracle and released as open source software. This will provide your VFX development business with professional office and business productivity software support. Download this great software package at `http:www.openoffice.com`. The Apache OpenOffice homepage and **Download** button are shown in Figure 1-11. Select your OS, language, and software version. I recommend using a 64-bit OS and 64-bit software so that you can use 16MB (or even more) of memory, and thus can use multiple applications.

Figure 1-11. Download Apache OpenOffice 4.1.2 at OpenOffice.org

There are other professional software packages available that we have not covered here, especially if you are using the Linux OS (Ubuntu, SUSE, Fedora, Mint, Red Hat, etc.), which has some other awesome audio production packages, such as QTractor and RoseGarden. I have set you up with the top-tier multimedia production software packages in this chapter for each genre, so you have an excellent foundation. If you are using a laptop, be sure to also check out Ubuntu Mate 16.04 at `www.ubuntu-mate.org`, as it makes your system extremely efficient and hyper-fast!

Summary

In this first chapter we set up your open source VFX content production workstation. We did this by downloading and installing professional open source new media content development software packages. These included: **Fusion**, **GIMP**, **Inkscape**, **Blender**, **Lightworks**, **Audacity**, and the **OpenOffice** integrated office productivity suite software, which you can use to run your VFX content development business.

In the next chapter, we'll start to look at foundational concepts behind raster imaging, including the pixels and frames that are utilized for digital imaging, digital video, and VFX processing pipelines.

Chapter 2

The Foundation of Raster for VFX: Pixels, Color, and Alpha

Now that you have some VFX and new media content production software installed, covering areas like digital illustration, digital audio editing, digital imaging, digital video editing, and visual effects (VFX), it is time to get into all of the underlying concepts, like **raster**, **motion**, **vector**, and **audio**, that span these multimedia genres and come together as one inside of your VFX software package—in our case, Fusion 8.0. The concepts covered over the next few chapters will provide an overview of what is contained within your visual effects assets, projects, and effects-processing pipeline. We will cover motion concepts in Chapter 3 and audio concepts in Chapter 4.

Digital imaging software, such as GIMP and Photoshop, and digital video software, such as Lightworks, use **raster**, or **pixel-based**, technology. My book *Digital Image Compositing Fundamentals* (Apress, 2015) is useful if you want to delve deeper into **raster imaging**. There is also the **vector**, or **math-based**, technology that is used for illustration and 3D software packages, like Inkscape, CorelDraw, and Blender. We'll be covering vector in the next chapter, as it's different from raster imaging, although vector assets can be **rasterized**.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_2

I will start with low-level concepts during these first few chapters. In this chapter we will discuss **raster images** and foundational concepts behind raster imagery. I will cover pixels, resolution, aspect ratio, color space, color depth, masks, alpha channels, anti-aliasing, layers, blending modes, palettes, and dithering. We need to get all of this technology lingo out of the way during the first few chapters so that you will understand the concepts that are found in the remaining chapters. Bear with me if you already understand the fundamental terminology that is used for VFX and new media content production pipelines and workflows. I will try to cover all these topics in as few pages as possible!

Raster Image Concepts: Arrays of Pixels

The **raster approach** to digital imaging is completely different from the **vector approach**. Whereas a vector approach uses math to define imagery using illustrations, which are created by SVG commands, or 3D geometry, using polygons and splines, the raster approach uses tiny dots called *pixels* to create the image. You have to look at these pixels all at once—not close up, but from a distance. Imagine those LED (Light Emitting Diodes) signs in Las Vegas, where you can't see what is on a sign when you are right in front of it, but can see a brilliant image once you're the right distance away.

Picture Elements: Pixels Are the Raster Image Building Blocks

Digital imagery is made up of **two-dimensional** "arrays" or grids, which contain **pixels**. The industry term **pixel** is the combination of two terms: **pictures** (hip folks call these "pix") and **elements** (shortened to be just "els"). Thus, the foundation for any digital image that you will be working with are its picture elements. These pixels dictate everything about your digital image asset, like its file size, dimension, color, transparency, and shape. It is important to note that vector or digital illustration assets are **not** made up of pixels, at least not until the SVG command that composes the vector is converted to pixels using an algorithmic process called **rasterization**. The digital image asset used in digital video pipelines can be said to be "static" (that is, without motion), whereas digital video will be said to be "dynamic," because it does incorporate motion. We'll be covering what makes digital video dynamic in the next chapter when we cover frames and 4D (animation) concepts.

Image Resolution: The Number of Pixels in Your Raster Image

The number of pixels contained in your digital image asset will be expressed using a term called **resolution**. This term also applies to digital video assets, as each frame contains the same number of pixels along the X and Y dimensions. Resolution contains a **width**, which is usually denoted using a W, or alternatively using an X, which stands for your x-axis, as well as a **height**, which is denoted using an H or a Y, which stands for the y-axis. The image (or video) resolution gives you the digital image or video **dimensions**. For instance, the popular VGA resolution would be expressed by specifying **640 x 480**. Alternately, you could also use the word "by," like this: **640 by 480**. XGA resolution is **1024 by 768**, Blu-ray resolution is **1280 by 720** pixels, HDTV resolution is **1920 by 1080** pixels, and Ultra HD resolution is **4096 by 2160** pixels.

To find the total number of pixels that are contained in any 2D digital image, you will want to multiply width pixels by height pixels, or, in coding terms: **Resolution = Width * Height**.

To find the total number of pixels that are contained in any 2D digital video, you will want to multiply width pixels by height pixels, by the number of video frames, or, in coding terms: **Total Pixels in Digital Video = (Width * Height) * TotalFrames**.

Hopefully, you remember the **area of a rectangle** equation from grade school. Here it is again in a professional digital video editing project context. For example, an HDTV image with a resolution of 1920 x 1080 pixels will contain 2,073,600 pixels, which you can determine if you multiply the width and height together. If you like digital photography, you're probably familiar with the term **two megapixels**, which refers to 2 million pixels. This is essentially what this HDTV resolution is giving you in terms of the number of pixels.

The more pixels that are contained in the digital image or digital video, the higher its resolution will be said to be. Higher resolution imagery and video will give your viewers more detail, or image subject matter definition. This is why 2K HDTV is called **high definition**, and why the new 4K resolution UHD TV is called **ultra-high definition** (4096 pixels is called "4K").

Image Aspect Ratio: A Ratio of W:H Pixels in an Image or Video

Closely related to the resolution of the digital image or video is the **ratio of X to Y pixels** in a digital image. This is called the **aspect ratio**. This concept is more complicated than that of resolution, because it is the ratio of width to height, represented as **W:H**, that exists in an image or video resolution (dimensions). If you like to think in terms of an image x-axis and y-axis, this would be **X:Y**. This aspect ratio defines the shape for your image or video, and the concept also applies to the shape of a device's display screen. For instance, smartwatches will have a square aspect ratio of 1:1, and your ultra-wide iTV set will feature a rectangular 2:1 aspect ratio.

A digital image (or display screen) with a 1:1 aspect ratio can be said to be perfectly square. Since this is the aspect ratio, a 1:1 aspect ratio is the same as a 2:2 or a 3:3 aspect ratio. It is important to note that it is this ratio between the two numbers X and Y that defines the shape of the image or video, not the numbers themselves, and that is why it is called an aspect ratio—although it's often called **aspect** for short. A 2:1 aspect ratio would create a widescreen image. A 3:1 aspect ratio is usually called a **panoramic** aspect ratio.

Your image aspect ratio is generally expressed using the smallest set or pair of numbers that can be achieved (reduced to) on either side of the aspect ratio colon.

If you paid attention in high school when you were learning about the lowest (or least) common denominator, these aspect ratio calculations should be fairly easy for you to comprehend.

I would do this mathematical problem by continuing to divide each side by two. Let's take a fairly weird 1280 x 1024 (this is the "SXGA" or "Super XGA") resolution for our example.

Half of 1280:1024 is 640:512, and half of 640:512 would then be 320:256. Half of that is 160:128, and half of that is 80:64. Half of that is 40:32, and half of that is 20:16. Half of that is 10:8, and half of that is 5:4. Therefore, an SXGA resolution uses a 5:4 aspect ratio, which is fairly close to being square. The larger and closer your numbers on the two sides of the colon are to each other, the squarer the rectangle defined by the aspect ratio is. Thus, a 5:4 aspect is squarer than a 4:3 aspect, which is squarer than a 3:2 aspect, which is squarer than 2:1 aspect.

Interestingly, all the SXGA ratios in that example were the same aspect ratio and were valid. Thus, if you wanted to take the really easy way out, replace that "x" in the image resolution with a colon, and you have an aspect ratio

for that image. The industry standard involves distilling an aspect ratio down to its lowest format, as we've done here, as that is the most useful ratio. Next, let's look at how pixels define colors!

Digital Color Theory: Each Pixel Contains 3 RGB Color Channels

Within the array of pixels that makes up your resolution and its aspect ratio, each pixel will hold color values, using three **color channels**, in something called the **RGB color space**. RGB stands for the red, green, and blue color values each pixel can define to establish what color that pixel is. Since devices that are going to display your image or video assets use this RGB color space to do so, we will use this RGB color space throughout this book, rather than other color spaces, such as **YUV** or **YCbCr**, which are more frequently used with **analog** (broadcast, reduced color spectrum, or interlaced) **video**. The digital, 24-bit, non-interlaced color devices made these days do not require YUV color data, which is used to correct for, or optimize for, human perception of color.

There are advanced techniques that can be implemented by switching between different color representations, such as YCbCr and YUV to RGB and HLS (hue, luminance, saturation), but these are beyond the beginner scope of this *VXF Fundamentals* title.

Color channels were originally used in digital image compositing programs like GIMP for compositing digital imagery for use on display screens. The digital video editing software I'll be using provides a nearly identical compositing pipeline, as will the Fusion 8 visual effects software package.

Color channels are sometimes referred to as **color plates** in the printing industry due to older printing presses, which use metal plates. Some of these presses are still in use today.

In GIMP, color channels have their own **Channels palette** that allows us to work on just that color channel (or plate), which can be quite useful for special effects or other advanced image operations. Editshare Lightworks also has a Channels feature. There is a Red (light) channel with 256 levels of light or brightness, a Green (light) channel with 256 levels of light or brightness, and a Blue (light) channel with 256 levels of light or brightness under the Channels tab.

An opposite of RGB, or **additive**, color is **subtractive** color (CMYK), which is used in printing and involves using inks. Inks subtract color from each other, rather than adding color, which is what happens when you combine different colors using light.

Taking red and green as an example, using additive color, Red + Green = Yellow. Using subtractive color, Red + Green = Purple. As you can see, additive gives you brighter colors (adds light) while subtractive gives you darker colors (subtracts light).

To create millions of different color values using these RGB color channels, what you will need to do is vary the **levels** or **intensities** for each of the individual RGB color values. The number of red, green, and blue values, or levels of intensity of light, that you have available to mix together will determine the total number of colors that you will be able to reproduce. A color needs to be generated for every image pixel.

Every pixel in an image will contain **256 levels** of color intensity for each of the RGB (red, green, and blue) color data values. **Color intensity** (or brightness) data inside each of the digital image pixels is represented with a brightness level for each color. This can range between **zero** (brightness turned off, or black) and **255** (brightness fully on; fully saturated color). This controls the amount of color contributed by each pixel for each of these red, green, and blue colors. As far as digital video goes, each frame is a digital image, in essence, so everything that applies to digital imaging applies to digital video too.

Calculating a total amount of available colors is easy, as it is again simple multiplication. If you multiply 256 times 256 times 256 you get 16,777,216 colors. This represents the number of unique color combinations of red, green, and blue that you can obtain using these 256 levels (data values) per color that you have to work with across these three different additive color channels. If you are using 16-bit color you have 65,536 colors, and 8-bit color offers 256 colors. The multiplication involved is simple.

There are also new 48-bit color and 64-bit color models, which are used in VFX production pipelines where HDRI, or High Dynamic Range Imaging, is utilized. This is usually done with a 3D content production workflow, or with expensive film or 4K DV cameras, which have support for 16-bit color (so, 65,536 colors) per color channel, allowing for a much higher dynamic color range (65536x65536x65536).

Image Color Depth: Bit Levels Will Define Your Number of Colors

The amount of color available to each pixel in a digital image is referred to in the industry as the **color depth**. Common color depths used in digital images and video include 8-bit, 16-bit, 24-bit, 32-bit, 48-bit, and 64-bit.

The **true color** depth image will feature the 24-bit color depth and will thus contain 16,777,216 colors. **High color** depth images feature 16-bit color depth (256 x 256 = 65,536 colors) in an RGB565 (five bits red, six bits green, five bits blue) data configuration.

Image file formats that support 16-bit and 24-bit color depth include the BMP, XCF, PSD, TGA, and TIFF. True color–only image formats include JPEG (JPG), PNG24, or WebP (WebPhoto). For digital video, MPEG-4 H.264 and H.265 and WebM support 24-bits.

Using true color depth will give you the highest quality level. This is why I'm recommending the use of PNG24 or PNG32 imported image assets for visual effect compositing pipelines, or for your digital video editing compositing, or for a digital image editing and compositing workflow.

Figure 2-1 shows a true color image created with Autodesk 3D Studio Max, which uses 24-bit color space. Notice the higher quality of the image when compared to Figure 2-2, which uses a lower 5-bit color image quality that uses 32 colors instead of 16,777,216 colors to create the image.

Figure 2-1. *High-quality 24-bit true color digital image example*

Figure 2-2. *Photoshop Save for Web dialog showing indexed color*

Next, let's take a look at indexed color in case you need to use PNG8 or GIF assets in your visual effects pipeline, or in case you have to use existing digital image assets that do not have the advantage of 24-bits, 32-bits, 48-bits, or 64-bits of color.

Indexed Color Depth: Using Palettes to Hold 256 Colors

The lowest color depth exists in 8-bit **indexed color** images. These feature a maximum of 256 color values (2 to the 8th power, or 8 bits of Binary data, known as a "byte") and use an indexed "palette" of colors, which is why they are called indexed color images. Popular image file formats for indexed color include GIF, PNG8, TIF, BMP, or Targa. The indexed color palette is created by the indexed color **codec** when you export your file from an imaging software package, such as GIMP. Codec stands for **CO**de-**DEC**ode and is an **algorithm** that can optimize a file size to be smaller using **compression**.

The way you convert 24-bit, true color image data to this indexed color image format (GIF or PNG8) in Photoshop is to use the **File ➤ Save for Web** menu sequence. This will open your **Save for Web** dialog, which will allow you to set a file format (GIF or PNG), number of colors (from 2 up to 256), color conversion algorithm (perceptual, selective, adaptive, or restrictive), the dithering algorithm (diffusion, pattern, or noise), and a number of other advanced options, such as progressive interlacing. The **dithering algorithm** interleaves pixels of neighboring colors in an attempt to create the illusion of a smooth color transition.

To convert true color image data into indexed color image data using GIMP 2.8.18, you would use the **Image ➤ Mode ➤ Indexed** menu sequence. This will call up an **Indexed Color Conversion** dialog. This has fewer options than your Photoshop Save for Web dialog, but the important ones are there so you can specify color depth and diffusion dithering. I recommend using the GIMP **Floyd-Steinberg** diffusion dithering algorithm. There is even an algorithm that reduces color bleeding, thus keeping imagery edges clean and sharp.

As an example of color depth, if you selected two colors, that would be a 1-bit (PNG1) image, four colors would be a PNG2 (2-bit color depth) image, sixteen colors would be a 4-bit PNG4 color depth image, sixty-four colors would be a 6-bit PNG6, and one hundred twenty-eight colors would be a 7-bit PNG7 image. In case you're wondering, there are digital video formats that can support indexed color digital video, like the aGIF, or animGIF, format does, but with synchronized audio. I am a data footprint optimization aficionado, thus I'd suggest that the more mainstream video codecs add an 8-bit color optimization algorithm to allow video editors to have the capability to optimize faster, smoother video.

Next, let's take a look at the other color format, 24-bit true color, or 32-bit true color plus alpha channel transparency.

True Color Depth: Using 24-bit True Color Imagery

One of the most widely used digital image file formats in the world is the **JPEG** file format, and it only comes in one flavor: 24-bit color. Other file formats that support 24-bits of color data include Windows BMP, TIFF, Targa (TGA), Photoshop (PSD), and PNG24. Since the PNG format supports 8-bit (PNG8) or 32-bit (PNG32) color, I call a 24-bit PNG format PNG24 to be precise. The primary difference in the true color file formats comes down to a format characteristic: **lossy** versus **lossless** compression.

Lossy compression means that an algorithm, which is also called a codec (COde-DECode), throws away some of the data to achieve a smaller data footprint. For this reason, save your original, uncompressed file using a lossless format prior to applying any lossy compression—in this case, JPEG. It is important to note that MPEG-4 and WebM are **lossy video formats**.

Lossless compression, used by the PNG, GIF, BMP, TIFF and TGA formats, doesn't throw away any original image data; rather, it applies an algorithm that finds **patterns** that result in less data being used, and that can 100 percent reconstruct all of the original pixel values.

True color images are used for user interface design or for websites and application content. They can also be used for other digital content, such as that used for e-Readers, iTV set apps, games, smartwatches, e-Signage, and social media sharing forums that support digital imagery, videos, or illustration. We will be using true color images for visual effects pipelines, as we need a high-quality color space to be able to pull off an effects-processing result that's visually realistic in the end.

Using more than one image or video is called **layer compositing**. Compositing involves using more than one asset, arranged in several layers. Your background, or **backplate**, video uses 24-bit image data. All the other layers in a **compositing stack** above a background plate need to support **transparency**, and will need to use 32-bit, known as **ARGB** or **RGBA**, image or video data.

This transparency is provided by a **fourth channel**, known as the **alpha channel**. I'm going to introduce you to this next.

True Color Plus Alpha: Using 32-bit Digital Imagery

Besides 8-bit, 16-bit, and 24-bit digital imagery, there is also 32-bit digital imagery. Formats that support the 32-bit color depth include PNG, TIFF, TGA, BMP, and PSD. I like to use **PNG32**, as it's supported in HTML, Java, JavaFX, CSS, and Android, while other file formats are not used in open source platforms.

These 32-bits of image data include 24-bits of RGB color data, plus 8-bits of "alpha," or transparency, value data, held in what is commonly referred to as your alpha channel.

Since you now know that 8-bits holds 256 color values, it will make sense to you that an alpha channel will hold 256 different **levels of transparency** data values for each pixel in a digital image. This is important for digital video compositing, because it allows layers that hold this 32-bit image data to allow some portion (from zero to 255, or all of that pixel's color) of the color data to bleed through to (or to blend with) layers below.

Next, let's take a closer look at what alpha channels do.

Alpha Channels: Defining a Transparency Level for Each Pixel

Let's take a look at how alpha channels define digital image as well as digital video pixel transparency values, and how they can be utilized to composite digital video compositions. Alpha channel data provides transparency inside of the visual effects compositing pipeline in software such as Fusion, Lightworks, GIMP, Blender, and Inkscape. Alpha data can be used in both static (imaging) or dynamic (VFX, video) compositing pipelines to composite digital images and video together in real-time using Fusion VFX software, as well as open platforms such as Android Studio, HTML5, CSS3, Java, JavaFX, and PDF. I would term code compositing using these platforms as "interactive dynamic" use, as the code allows you to access the pixel transparency values interactively in mere nanoseconds of time. This allows you to animate the data in any way that you like, as well as make it respond in real-time to the user's control. This can be used in websites, games, animated user interface designs, iTV programs, interactive e-Books, digital signage, and smartwatch faces.

Visual effects compositing involves the seamless blending of more than one layer of digital image, digital video, and digital illustration assets. As you might imagine, per-pixel transparency, provided by each asset's alpha channel, is indeed an important concept when you have more than one layer involved in a digital video editing or visual special effects project.

Visual effects compositing needs to be used when you want to create a video on a display that appears as though it is one single video, but that is actually a seamless collection of more than one composited image and video content layers.

One of the principle reasons you would want to set up an image, video, audio, and animation composite is to allow you to exercise fine-tuned control over various elements by having different new media assets (components) on different layers. In digital audio, which we are going to cover in Chapter 4, these layers are termed "tracks." To accomplish a multi-layered composite—with the exception of digital audio, where silence would be the equivalent of transparency—you will always need to have pixels with alpha channel transparency data values located on layers that are above your backplate (or background).

Alpha channels can be utilized to precisely control the **opacity blending** of each pixel's color with the pixels in the same X,Y location on the layers below it.

Like the RGB color channels, the alpha channel will have 256 levels of transparency, from 100 percent transparent (zero) to 100 percent opaque (255). Each pixel will have different alpha transparency data, just like each pixel will have different RGB color values.

This is the reason that the **Layers** and **Channels** tabs for software packages such as GIMP, Photoshop, Painter, Lightworks, Pinnacle Studio, Inkscape and VideoStudio Ultimate are grouped together using a **floating palette**. You will observe this user interface design commonality as you use the different new media software packages over the course of this book.

Porter-Duff: Algorithmic Pixel-Blending Modes for Each Layer

There is another powerful aspect of layered compositing called **blending modes**. Any of you who are Photoshop or GIMP users have seen that each layer is able to be configured to utilize a different blending mode. Blending modes are **algorithms**. These specify how the pixels for each composited layer are blended mathematically with the compositing layers underneath that layer.

These pixel-blending algorithms take into account a pixel's transparency level. Between these two image compositing controls, you can achieve virtually any compositing result that you are trying to achieve using GIMP, Photoshop, Painter 2016, Inkscape, Lightworks, Blender or Fusion 8.

A major difference on dynamic platforms such as Android is that blending modes can be controlled **interactively** using custom **Java** or **JavaFX** programming logic. Some of these powerful **PorterDuff** Java class blending modes include **XOR**, **ADD**, **SCREEN**, **OVERLAY**, **DARKEN**, **LIGHTEN**, and **MULTIPLY**. The *Pro Android Graphics* (Apress 2013) title covers how to implement PorterDuff blending modes inside the complex image compositing pipeline, if you are interested in learning about Android Studio programming or Java-based layer compositing. Next, let's learn about anti-aliasing.

Smoothing the Edges in Digital Image Composites: Anti-Aliasing

Anti-Aliasing is a popular digital image technique that is also implemented using an algorithm. This algorithm finds where two adjacent colors meet along an edge and blends the pixel colors together along that jagged edge. Anti-aliasing will add **averaged colors** along the edge between two colored areas to visually smooth those two colors together along the (formerly) jagged edges. This will make jagged edges appear to be smoother, especially when your image or video is zoomed out, such that the individual pixels are not easily discernable.

Anti-aliasing tricks the eyes into seeing smoother edges, eliminating what's commonly termed **the jaggies**. Anti-aliasing provides impressive results using a few (usually fewer than eight) intermediary averaged color values for those pixels that lie along the edges within an image or video layer that contains assets that need to be made to look smoother.

By *averaged color* I mean a color or a spectrum of colors whose shade is part of the way between the shades of the two colors that intersect along that edge being anti-aliased. I created a visual example of anti-aliasing for you to show the resulting effect. I first created the seemingly smooth red circle, seen in Figure 2-3, against a yellow background. I zoomed into an edge of that circle, and then I grabbed a screenshot. I placed this next to the zoomed-out circle to show the orange anti-aliasing pixels. Notice there are seven or eight averaged colors that are used to create this visual effect, which is really not that many, if you think about it. As you see on the left, the result is a sharper edge.

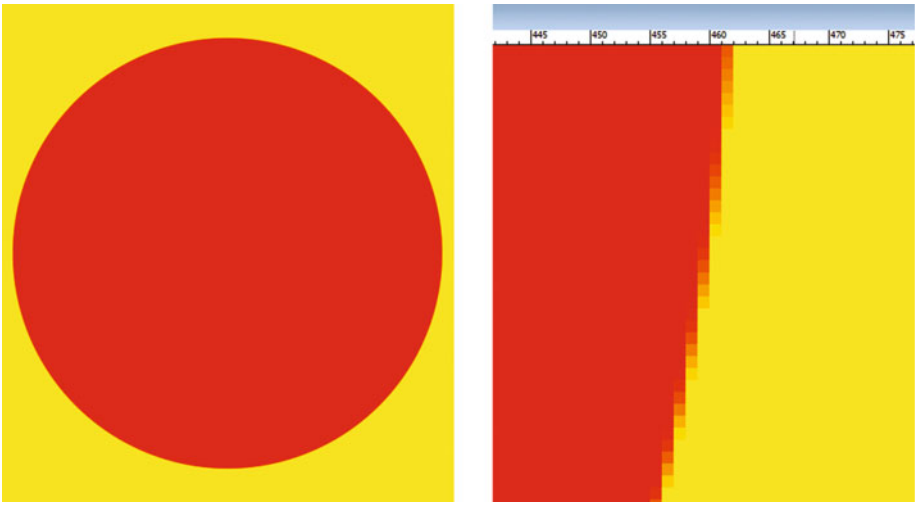

Figure 2-3. The zoomed-in right view shows anti-aliasing effect

One of the nifty tricks I utilize to implement my very own anti-aliasing effect is to use a **Gaussian Blur** filter on any jagged edges in digital compositing layers in Photoshop, VideoStudio Ultimate, GIMP, Pinnacle Studio, Painter, Fusion 8, Lightworks, and Lightworks Pro (the paid version of Lightworks).

Be sure to use **low blur values** (0.15 to 0.35) on subject matter (as well as on alpha channels, if you are anti-aliasing the transparency mask) that contains these jagged edges.

This will provide the same anti-aliasing effect that you see in Figure 2-3, and not only that, it will "blur" the alpha channel transparency values as well so you are smoothing the composite between any two layers in real-time. This is significantly more advanced. I'll try to cover advanced techniques whenever possible, even though this is a VFX fundamentals book and not an intermediate or advanced visual effects title.

Blurring the alpha channel will allow it to anti-alias a 32-bit image object with any background imagery you may be attempting to seamlessly composite it against, which I call real-time anti-aliasing. It is implemented using "static" alpha channel data contained within what is essentially a black and white image used as a **transparency blending guide**. The grey pixels on the edge in the alpha channel data will then tell the compositing pipeline to average the colors between the current layer and those pixels (colors) that are underneath it in your digital effects or digital image compositing project.

Summary

In this second chapter I made sure you had a solid foundational understanding of raster image concepts, since these concepts will be needed in VFX, digital video, and 3D software when 2D "static" new media components are imported for use in the digital image asset layer compositing pipeline for your digital VFX projects. We looked at the raster image concepts and terminology especially carefully, as raster image assets are used in just about every stage of the VFX content creation pipeline. We learned about pixels, resolution, and aspect ratio, and how to calculate each of these mathematically. Then we took a look at color modes, color channels, and color depth, and how they define color within your digital visual effects projects.

Next, we looked at more advanced concepts such as layer compositing, alpha channels, pixel-blending modes, PorterDuff algorithms, and anti-aliasing algorithms, all of which we will be using during this book so that you get hands-on experience in order to become a visual effects pro.

In the next chapter you'll take a look at **frames** and how these turn digital imagery into **dynamic motion digital video**.

Chapter 3

The Foundation of Motion for VFX: Frames and Codecs

Now that you have an understanding of the fundamental concepts, terms, and principles of digital image new media content, it is time to get into moving imagery, since **motion** is a major part of VFX project design. The **digital video** new media asset type has its own concepts, terms, and principles, and therefore I'm going to take a chapter to show how 2D raster images become 4D digital video assets. We will be looking at the concepts behind analog video (or film) and how it is digitized into digital video using popular "codecs," or **CO**mpressor-**DEC**ompressor algorithms, via digital video encoding software and plug-ins.

We will look at how digital video is created using image frames and then displayed at a rapid rate of speed, and we will also review digital video concepts such as frame rate and resolution. We will look at advanced digital video playback concepts you'll be using for your VFX content production workflow, such as bit-rates, video streaming, HD, UHD, compression, and digital video formats.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_3

Digital Video: Concepts and Terminology

Digital video is comprised of both digital imagery and digital audio. I covered digital imagery concepts in Chapter 2 and will cover digital audio concepts in Chapter 4. Digital video can be used for a number of things other than as a resource (or asset) that is input into a VFX project pipeline; indeed, it can be used on its own, such as you see on the Internet, or in Android Studio, Java, or JavaFX application development. You can also use digital video in your user interface design, as a content background, to hold a special effect you are applying to your visual effects project, or in conjunction with 3D or i3D project design, which we will be looking at in more detail during Chapters 5 and 6 when we look at the 2D and 3D vector new media asset types and their concepts and terminology.

Digital Video Is Animated: Frames and Frame Rate

Digital video extends digital imagery into **4D**, or the fourth dimension of **time**. This is done with digital images called **frames** in the digital video and in the film industry. A frame is contained in a collection of slightly different digital images or frames, making it a digital video asset.

The digital video assets are therefore comprised of this ordered sequence of frames that are displayed rapidly over time and serve to create the intended illusion of **motion imagery**.

The primary concept behind **frames** in 2D digital video assets builds upon the primary concepts for 2D digital imaging. Digital video frames also contain **pixels**, which means that the digital video asset also has a **resolution**, as well as an **aspect ratio** and **color depth**. The other concepts from Chapter 2 also apply to digital video. All the frames in a video use the same resolution, aspect ratio, and color depth.

Most digital video assets will use a **24-bit** color depth; this is supported in Fusion 8 as well. Indexed color video does exist, as Microsoft Video 1 codec operates either in 8-bit palettized color mode or in a 16-bit RGB_565 color space. Animated GIF (aGIF or AnimGIF) can also index the color palette for motion video graphics assets, but neither Microsoft Video 1 nor Animated GIF are recommended for use in VFX pipelines.

Like pixels do in digital imagery, digital video frames multiply the data footprint with each frame used. In a digital video, not only does the digital video frame resolution greatly impact the resulting file size, but so does the number of **frames per second** that the codec needs to analyze and compress during the video encoding process. More data to compress equals larger file sizes, even if the compression algorithm is reducing data.

This number of frames per second is commonly referred to as **FPS**, and is also referred to as the frame rate in the visual effects, digital video, and film production industry.

Common frame rates include **60 FPS** for i3D console games, **30 FPS** for digital video, **24 FPS** for feature films, and **20 FPS** for multimedia. There is new-media content that should support even lower frame rates than 20 FPS, such as 15 or 12 FPS.

To find out how long each frame is displayed based on a frames per second (FPS) value, divide the number 1 by your FPS value, move the decimal right four places, and add "milliseconds."

Thus, 20 FPS displays for .05 of a second, 30 FPS displays for .033 of a second, 24 FPS displays for .04167 of a second, or 416.7 milliseconds, and 60 FPS displays for .0167 of a second, or 167 milliseconds.

Next, let's take a look at how to calculate the raw video data footprint; that is, how much system memory will be needed.

Digital Video Mathematics: Doing the Multiplication

If you multiply the number of pixels in your image by the number of color channels, you will get the raw data footprint for that image. With digital video, you will multiply that number again against the number of frames per second that the digital video is set to play back at to get the data footprint per second. To get the total data footprint, you would then multiply that number by the number of seconds that represent the total duration of your video clip.

So, with a VGA or SD Video with 640 by 480 resolution with a 24-bit color depth, you would have **[(640x480) x 3] = 921,600** for just one frame of video. At 30 FPS this would be **921,600 x 30 = 27,648,000** bytes of data. If you divide this by 1024, you have 27,000 KB data, or 27 MB of raw data (or system memory), for one second of video.

You can see why having a powerful video codec that will compress this raw data footprint down by an order of magnitude, given the optimal compression settings, is extremely important! Let's take a look at digital video codec algorithms next so we can see how the codec is able to accomplish this.

Digital Video Algorithms: Digital Video Codecs

You will be amazed at some of the digital video data compression ratios that we will achieve using MPEG4 and WebM video file formats once you know exactly how to best optimize a digital video compression work process. This is done by using the correct bit-rate, frame rate, frame resolution, and color depth for digital video contents and specific VFX applications.

I will try to cover the concepts that apply to the digital video data footprint optimization work process for VFX projects in this chapter so we can get it out of the way!

That said, it is important to note here that you'll want to use uncompressed (raw) digital video assets in a VFX project pipeline so that all the algorithms performed on those pixels and frames have the original data to work with. Only after the VFX project is complete will you data optimize it using codecs.

Let's take a look at how the digital video codec differs from a digital audio or digital image codec. There will be some elements of both digital image and digital audio compression found in digital video compression algorithms, but it can be an order of magnitude more complex because it has to ascertain and compress **inter-frame** pixel movements. So, not only does a codec compress in 2D space (pixels and waveform), but it will also compress in 4D space; that is, any **changes between frames**. For this reason, talking head videos, or video with no panning or zooming, will compress better because there are very few moving pixels between frames.

Video Content Delivery: MPEG4 H.264 and WebM

Your VFX projects will most likely be delivered on popular consumer electronics devices such as smartphones, iTV sets, tablets, smartwatches, and e-Book readers, each of which have sold billions of units by now. The platforms that have the most market share are HTML5 and Android, with Windows, Blackberry, or iOS also having part of the market share. Android Studio supports the same two open source digital video formats that HTML5 supports: the MPEG4 H.264 AVC and the ON2 VP8 and ON2 VP9 codecs, which were acquired by Google from ON2 Technologies. Google renamed these WebM, then released them in an open source licensing schema. This cross-platform open source digital video codec support is great news from the content production standpoint, as VFX content produced and optimized by developers can be used in HTML apps, browsers, and hardware devices, as well as in Android Studio applications. Apple uses a proprietary QuickTime format, and Windows uses the proprietary Windows Media WMV format. Blackberry uses the same MPEG4 and WebM formats that are used in Android, Java, and HTML5 due to their widespread support, high quality, and popularity.

This digital video format cross-platform support affords visual effects content developers with a "produce once, deliver everywhere" production scenario. Every content production team and VFX programmer dreams about this. This could reduce content development costs and therefore increase revenues, as long as these economies of scale in content development can be taken advantage of by visual effects project developers.

Since consumer electronics devices currently have screen resolutions that are using medium 1280 x 720 to higher 1920 x 1800 resolutions, with some devices now utilizing ultra-high 4096 x 2160 resolution, you would utilize the MPEG4 H.264 AVC format. This is the digital video format most often used in the world today.

Digital Video Resolutions: Industry Standards

Let's start out by covering the primary resolutions used in commercial video. Before HDTV (high-definition television) came along, video was called "SD," or standard definition, and used a standard vertical resolution of 480 pixels. The original aspect ratio for SD was **4:3** (640 by 480). More recently, a widescreen aspect ratio was added, making SD video 720 by 480 resolution.

HDTV resolution video comes in two different resolutions of 1280 x 720, which I call "Pseudo HD," and 1920 x 1080, which the video industry calls "True HD." Both use a **16:9** widescreen aspect ratio, and they are now used not only in film, HD television, and UHD iTV sets but also in UHD smartphones, UHD tablets, and UHD e-Book readers. It's only a matter of time before smartwatches use HD as pixel pitches in OLED display technology approach 560 PPI.

The 1920 x 1200 resolution is, by the way, a less wide, or taller, **16:10** pixel aspect ratio. It is becoming more common as a widescreen device aspect ratio, as is the **16:8**, or 2:1 aspect ratio, with 2160 x 1080 screens having been available since 2013.

There is also 16:10 pseudo HD resolution, which features 1280 by 800 pixels. In fact, this is a common laptop, notebook, netbook, and mid-size tablet resolution. I would not be surprised to see a 16:8, 1280 by 640 screen offered at some point in time.

Generally, VFX content producers will try to match video content resolution to the target device resolution, and thus to the target device aspect ratio as well.

Similarly, manufacturers will try to match display screen resolution to popular content resolutions. Blu-ray is 1280 x 720, and so there are a lot of 1280 x 720 screens and 2560 x 1440 screens (two times 1280 x 720 on each axis scales up perfectly with a 4-pixel (2 x 2) matrix for each pixel in the Blu-ray content). You will also see 3840 x 2160 device screens as well, as two times 1920 x 1080 is 3840 x 2160, allowing upscaling to be precisely 100 percent or "2X," which gives users the highest quality visual results. Content scaling algorithms yeild perfect results only when up or down scaling by an even number of 2X (100%) or 4X (200%). Non-even scaling causes partial or fractional pixels, which are then blurred to simulate fractional pixels, thereby reducing visual quality.

Digital Video Compression: Bit Rates and Playback

Another important digital video concept that we need to cover in this chapter is the concept of **bit-rates**. Bit-rate is a key setting used in the video compression process. Bit-rates represent the **target bandwidth**, or the **data pipe size**, that is able to accommodate a certain number of data-bits that stream through it every second. Higher bit-rates will yield higher quality, because more of the original data is kept in order to reproduce the original (uncompressed) content. Therefore, if you have to use compressed assets in a VFX project, select the one that uses the largest (highest) bit-rate setting so as to get the best quality content into the visual effects processing pipeline.

In general, the smaller the video data file size you are able to achieve, the faster the data will travel through a data pipe, and the easier it will be to decode that data using the codec and the CPU, as there's less data to process.

There is a **quality to file size trade-off** where video asset compression is concerned. This will work against you (low quality) in the VFX project but in your favor (smooth playback) on the playback device as far as user experience is concerned.

Single-core CPUs in devices such as smartwatches may not be able to decode high-resolution, high-bit-rate digital video assets without dropping frames. This is a playback quality issue, so make sure to thoroughly optimize lower bit-rate video assets if you are going to target older (or cheaper) devices.

Digital Video Optimization: Encoder-Decoder Use

I am going to go over digital video optimization theory in the last part of this chapter, as it is an important subject when it comes to delivering your VFX projects. The **decoder** side of your digital video codec will always be optimized for speed, because **smoothness of playback** is the key issue, and the **encoder** side will be optimized to **reduce the data footprint** for your digital video asset that it is generating. For this reason, the encoding process may take a long time, depending on how many processing cores a workstation contains. Most VFX video content production workstations should support a minimum of eight, twelve, or sixteen processor cores so that encoding is faster and special effects are rendered more quickly (along with your powerful GPU processing pipeline and 4 GB to 8 GB of DDR5, of course).

Codecs on an **encoder** side are like plug-ins in the sense that they can be installed into different digital video editing software packages to enable them to encode digital video asset file formats. Since Android and HTML5 both support H.264 MPEG4 formats and ON2 VP8 and VP8 WebM formats for video, you need to make sure that you're using one of these codecs. For VFX output you may also want to consider the pristine MPEG-H H.265 HEVC.

More than one software manufacturer makes MPEG4 encoding software, and they could yield different (better or worse) results as far as encoding speed and file size goes.

One professional encoding solution that is available, if you wish to encode digital video assets, is Sorenson Squeeze Pro from Sorenson Media, which is currently at version 11.

Squeeze has a professional-level version, which I will be using in this book, that costs around $750 dollars, and whose value is in excess of the suggested list price amount if you consider what it will do. There are also less expensive versions of the software available at $200 and $550.

There is also the open source solution, called Editshare LightWorks 12.6, but the free version does not currently support output using an MPEG-4 AVC or WebM VP8 or VP9 codecs. The WebM codec is open sourced, so they could add that support if they wanted to. For now, I'll have to use Sorenson Squeeze Pro 11 for the book until the codec support for Android Studio and HTML5 is added to Editshare LightWorks 12.7, or maybe in V13. If you visit the LightWorks Forum, users are requesting WebM, as it is an open source codec and there is no good reason that it shouldn't now be included in Editshare LightWorks.

Digital Video Optimization: Encoder Settings

When optimizing for digital video asset file size using encoder settings, there are a number of important settings that directly affect the data footprint. I'll cover these in the order in which they affect the file size, from the most impact to the least impact, so you know which parameters to tweak in order to obtain the results that you're looking for. As with digital image compression, the resolution, or number of pixels, in each frame of video is the optimal place to start your data optimization process. If you are targeting 1280 x 720 smartphones or tablets, you do not need to use 1920 x 1080 resolution to get great visual results from your digital video asset.

With high-density, also termed fine dot pitch, displays (such as Android XHDPI, XXHDPI, and XXXHDPI) currently common in the devices market, you can scale 1280 video up 33 percent and it will look reasonably good. The exception to this should be iTV apps for Android TV, which has a medium MDPI dot pitch due to large 55- to 95-inch screen sizes. In this case, if you are developing applications for iTV sets, you would want to use a 1920 by 1080 True HD resolution so your content hits every pixel. This can be scaled up to 3840 by 2160 to support UHD with good results.

The next level of optimization would come in your number of frames used for each second of video (FPS). This assumes the actual seconds contained in the video itself can't be shortened through editing. This is known as the frame rate. Instead of setting a video standard 30 FPS frame rate, consider using the film standard frame rate of 24 FPS, or the multimedia standard frame rate of 20 FPS. You may even be able to use a low 15 FPS frame rate depending upon the amount of movement in your content.

Note that 15 FPS is half as much source data as 30 FPS, a 100 percent reduction of data going into the encoder. For some video content this will play back the same as 30 FPS content. The only reliable way to test how low you can get the FPS before it will start to affect your video playback quality is to set, encode, and review the result with different frame rate settings during your digital video (encoder) content optimization work process.

The next optimal setting you'll tweak or experiment with would be the bit-rate that you set for a codec to try to achieve. Bit-rate settings equate to the amount of compression applied, and thus set the visual quality for video data. It is important to note that you could simply use a 30 FPS, HD 1920 x 1080 video and then specify a low bit-rate ceiling.

If you do this, the results would not be as good looking as if you first experimented with lower frame rates, or with lower resolutions, using the higher (quality) bit-rate settings.

The next most effective setting to adjust in an attempt to obtain a small data footprint is the number of keyframes. The codec uses your keyframe settings to know when to "sample" your digital video. Video codecs apply compression by looking at a frame and then encoding only the changes, or offsets, present during the next few frames, so that a codec doesn't have to encode every single frame in your video data stream.

This is why a talking head video will encode better than a video where every pixel moves on every frame, such as videos with fast panning or rapid zooming, for instance. The keyframe setting in the encoder will force the codec to take a new frame sample of a video data asset every so often. There is usually an **auto** setting for keyframes; this allows the codec to decide how many keyframes to sample. There is also a manual setting that allows you to specify a keyframe sampling at set intervals, usually a certain number of times per second, or a certain number of times over the duration of the video (total frames).

The next most effective setting in obtaining a tiny data footprint is your **quality** or **sharpness** setting. This is usually implemented using a slider UI. Sharpness controls the amount of blur that codecs apply to your video pixels before compression. In case you are wondering how this trick works so that you can apply it yourself in GIMP during the digital image optimization process, applying a slight blur to your imagery or videos, which is usually not desirable, will allow for better compression. The reason for this is that a sharp transition in an image, such as sharp edges between colors, is more difficult for the codec to encode optimally—that is, using less data. More precisely (no pun intended), sharp or abrupt transitions in colors take more data to reproduce than soft transitions do. I would recommend keeping the quality or sharpness slider between an 85 percent and 100 percent quality setting. Try to achieve the data footprint reduction using the other variables that we've discussed here.

Ultimately, there are a significant number of variables that you'll need to fine-tune in order to achieve the best data footprint optimization for each particular video data asset. Each will be different (mathematically) to the codec, as each may contain a different array (collection) of pixel color data. For this reason, there is no standard collection of settings that can be developed to achieve any given result.

Your experience tweaking various settings may eventually allow you to get a better feel, over time, as to those settings you need to change to get your desired compression result with different types of uncompressed video source assets.

Summary

In this third chapter we took a look at motion imagery (digital video) concepts, principles, and formats that compress and decompress your digital image assets. We looked at how your resolution, color depth, frame rates, bit-rates, and codec settings can contribute to data footprint reduction.

In Chapter 4, you will learn about digital audio editing concepts and in Chapter 5 about 2D vector or digital illustration concepts and techniques.

Chapter 4

The Foundation of Audio for VFX: MIDI, Wave, and Sample

Now that you have an understanding of the fundamental concepts, terms, and principles of digital video new media content, the next logical topic would be digital audio new media content, since digital video media contains a digital audio component inside of its media container. This includes tracks for stereo music and mono tracks for vocal performances and sound effects. Audio can also be included in VFX projects in its own media containers, which we will be covering in this chapter as well. We will be looking at the concepts behind analog audio and how it is digitized into digital audio, as many of these analog and digital waveform concepts will apply across both mediums.

We will look at how waves of air create audio and analog audio concepts, such as amplitude and frequency, as well as look at digital audio concepts, such as samples and resolution. We will look at advanced digital audio concepts such as bit-rate, streaming, and HD audio. Finally, we'll look at all the powerful digital audio formats supported by the publishing platforms you will be targeting, such as HTML5, Android, Linux, Windows, Blackberry, Kindle and iOS.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_4

Digital Audio Concepts and Terminology

In this chapter I am going to get you up to speed on analog audio concepts and terminology, as well as on digital audio concepts and terminology. Digital audio is the more important of the two as far as your VFX production pipeline is concerned. We will see how the transition can be made between analog and digital audio using a process called **sampling**, which those of you who do sound design and music synthesis will be familiar with. We will look at **encoding** audio, using bit-rates, what **streaming** is, and the new 24-bit **HD audio** standard now being used for broadcast radio and satellite. We will also cover the mainstream digital audio codecs (audio file formats) supported in HTML, Android, Windows, Linux, Blackberry, Kindle and iOS.

Foundation of Analog Audio: Sound Waves of Air

Just like digital imaging and digital video, digital audio is quite complex, especially at the professional level. Part of this complexity comes from the need to "bridge" analog audio technology and digital audio technology together, because modern day consumer electronics devices are all digital. Analog audio used to be generated using speaker cones of different sizes that are manufactured using resilient membranes made out of one space-age material or another. These speakers can generate **sound waves** by vibrating, or pulsing, analog audio into existence. Our ears receive this analog audio, which is still used at live concerts, in exactly the opposite fashion, by catching and receiving those pulses of air, or vibrations with different wave lengths, and then turning them back into "data" our brain can process. This is how we hear these sound waves, and our brain will then interpret these different audio sound wave **frequencies** as different **notes** or **tones**.

A sound wave will generate a different tone depending on the **frequency** of that sound wave. Wide, long, or infrequent wave cycles will produce lower (bass) tones, whereas a more frequent (shorter) wavelength will produce a higher (treble) tone. It is interesting to note that different frequencies of light produce different colors, so there is a very close correlation between analog sound (audio) and analog light (colors), which will also carry through for digital production techniques and principles. Figure 4-1 shows a frequency component of an analog sound wave.

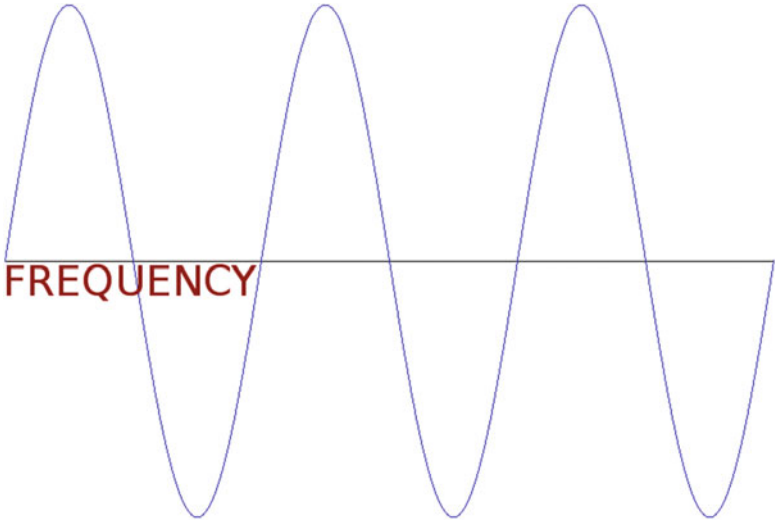

Figure 4-1. The analog sound wave frequency component

The volume of the sound wave will correlate with the **amplitude** of that sound wave, or the **height** (vertical size) of the wave. Thus, the frequency of sound waves equates to how closely together these waves are spaced along an x-axis, if you look at it in 2D, and amplitude equates to how tall these waves are, as measured along the y-axis. You can see the amplitude component of an analog sound wave in Figure 4-2.

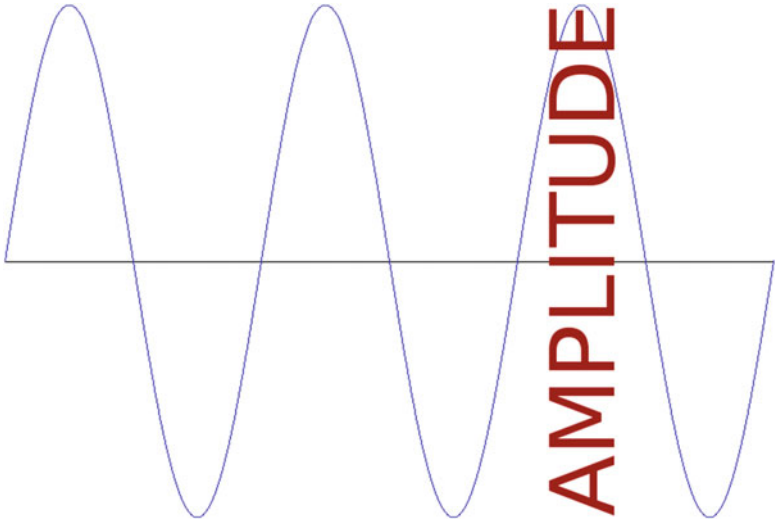

Figure 4-2. The analog sound wave amplitude component

Sound waves are uniquely shaped, allowing audio to mimic different sound effects. The baseline sound wave type is called a **sine wave**, which you learned about in math class with sine, cosine, and tangent math functions. Those of you who are familiar with synthesizer keyboards are aware of other sound wave shapes that are used in sound design, such as the **saw wave**, which looks like the edge of a saw, or the **pulse wave**, which is shaped using right angles, resulting in immediate on and off sounds (pulses).

Even randomized waveforms, such as **noise**, can be used in sound design to obtain an edgy sound result. As you might have ascertained using Chapter 3 knowledge, regarding data footprint optimization, the more "chaos" or noise that is present in your sound waves, the harder they will be to compress for the codec. This will result in a larger digital audio data footprint for that particular sound. Next, we are going to take a closer look at how this analog sound wave is turned into digital audio data using a process called sampling. This process is one of your core tools when it comes to sound design and music synthesis.

Digital Audio: Samples, Resolution, and Frequency

The process of turning analog audio (sound waves) into digital audio data is called **sampling**. If you work in the music industry, you have probably heard about a type of keyboard (or rack-mount equipment) called a sampler. Sampling is the process of slicing an audio wave into segments so that you can store the shape of that wave as digital audio data using a digital audio format. This turns an infinitely accurate sound wave into a discreet amount of digital data—that is, into zeroes and ones. The more zeroes and ones used, the more accurate the reproduction of the infinitely accurate (original) analog sound wave. The sample accuracy, or resolution, will determine how many zeroes and ones are used to reproduce the analog sound wave; we will be getting into that topic next.

Each digital slice of the sampled sound wave is called a **sample**, because it takes a sample of a sound wave at that exact point in time. The precision of the sample is determined by how much data is used to define each wave slice's height. Just like with digital imaging, this precision is called your resolution, or, more accurately (no pun intended), the **sample resolution**.

Audio sample resolution is defined as 8-bit, 12-bit, 16-bit, 24-bit, or 32-bit. In digital imaging and digital video the resolution is quantified by the number of color channels, and in digital audio the resolution is quantified by how many bits of data are used to define each of the audio samples being taken.

Just like with digital images, where having more colors yields better quality, with audio a higher sampling resolution can yield much better sound wave reproduction. Figure 4-3 shows a digital audio data sample in **Audacity** that uses a 32-bit floating point data representation, with a **Stereo 44100 Hz** data-sampling rate. I recommend using this as the minimum sample quality to be used with VFX project pipelines, with 48000 Hz, 32-bit being even better.

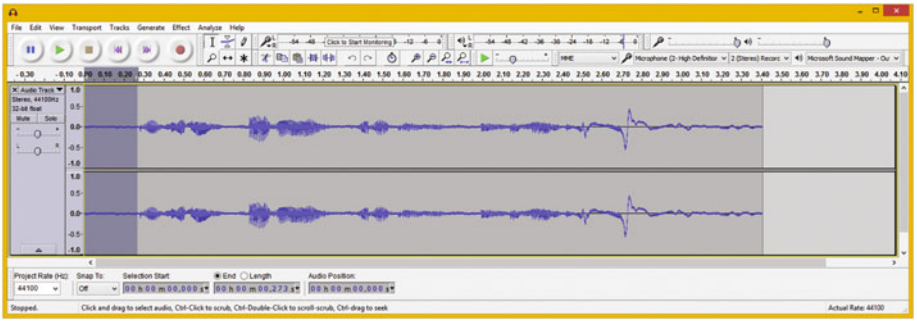

Figure 4-3. *An audio data sample in Audacity at 32-bit and 44100 Hz*

Higher sample resolutions use more data to reproduce any given sound wave sample, and will yield a higher audio playback quality, but at the expense of a larger asset data footprint.

This is the reason why 16-bit audio, known as CD-quality audio, would sound better than 8-bit audio, just like true color images will always look better than 8-bit indexed color images.

In digital audio there is now **24-bit** audio sampling, known as HD audio in the consumer electronics industry. An HD digital audio broadcast radio uses the 24-bit sample resolution so that each audio sample, or slice of the sound wave, contains 16,777,216 potential data samples of resolution. Consumer electronics devices should be moving to support 24-bit HD audio, like smartphones you see advertised featuring HD-quality audio playback. It will also be logical for iTV sets to ship with 24-bit audio playback hardware installed for home theater usage.

In order for a consumer hardware device to play back this HD audio format, it must have 24-bit audio hardware, so I would recommend using 16-bit audio for now, unless you are creating high-quality VFX content, which wouldn't be surprising! Most Smartphones, Tablets and iTV sets have 24-bit audio hardware capabilities these days anyway.

Besides a digital audio sample resolution, you also have a digital audio **sampling frequency**. This defines the number of samples, at any given sample resolution, that are taken during **one second** of sample time. In digital images, sampling frequency would be analogous to the number of pixels contained in an image. Sampling frequency can also be called the **sampling rate**. You are probably familiar with the idea of CD-quality audio, which is defined as using a 16-bit sample resolution and a 44.1 kHz sampling rate. This means taking 44,100 samples—each of which contains 16-bits of sample resolution, or the potential maximum of 65,536 data values—per second for the digital audio data held in each sample.

Let's do some math to find out how many bits of data might be held in one second of raw (uncompressed) digital audio data. This could be calculated by multiplying the 65,536 sample resolution by the 44,100 sample frequency. This yields 2,890,137,600 values that are available to represent one second of CD-quality audio. Audio codecs compress this down to an amazingly small file size, as not all the data is needed.

The point is, the exact same trade-off that you have in digital imaging exists using digital audio. If you include more data, you will get a higher quality result at the cost of a larger data footprint. Fortunately, audio codecs do a better job giving you even better quality-to-file-size results than digital image codecs or digital video codecs usually provide.

Common audio sample rates for the digital audio industry include: 8 kHz, 11.25 kHz, 22.5 kHz, 32 kHz, 44.1 kHz, 48 kHz, 96 kHz, 192 kHz, and, recently, 384 kHz. As you may have guessed, I like to use the ones that are evenly divisible by eight (power of two).

For this reason I'll usually gravitate towards 8 kHz for low quality (voice), 32 kHz for medium quality (background music or ambient sound effects), and 48 kHz for high quality (scores).

Lower sampling rates, such as 8 kHz, 11 kHz, or 22 kHz, would be optimal for sampling voice-based digital audio, such as a movie dialog track or an e-Book narration track.

Higher audio sample rates, such as 48 kHz or 96 kHz, will be more appropriate for music, and possibly sound effects such as rumbling thunder, which absolutely needs a high dynamic range for high-fidelity reproduction, especially if VFX work is involved.

Some sound effects can get away with using a lower 11 kHz or 22 kHz sampling rate, as long as the sampling resolution used is 16-bit quality. Ultimately, you'll have to use your ears as your guide during digital audio optimization to ascertain an aural-quality-to-digital-audio-file-size trade-off.

Digital Audio Publishing: Popular Audio Formats

There are considerably more digital audio codecs supported in popular devices and platforms than there are digital image or digital video codecs. Digital audio support includes .MP3 (MPEG3) files, .WAV (PCM, or pulse-code modulated) Wave files, .MP4 or .M4A (MPEG4) files, .OGG (OGG Vorbis) audio files, .MKS (Matroska) files, .FLAC (Free Lossless Audio Codec) files, and .MID, .MXMF, and .XMF MIDI (Musical Instrument Data Interface) files, which technically aren't even digital audio data at all. Let me explain what MIDI is first, since it is not a format that you are likely to be using in your VFX pipeline, but is one that underlies the history of digital audio. It is unique in its digital player piano performance playback approach, so I'm going to include it in this chapter, as it has some unique and highly data footprint optimized applications. After that, I'll cover the most popular sampled digital audio data formats.

MIDI: Musical Instrument Data Interface

MIDI stands for Musical Instrument Data Interface and it is one of the very first ways that you could work with audio using your computer. The origins of MIDI date all the way back to the 1980s, so MIDI has been around for several decades now.

The first computer to feature MIDI DataPort hardware was the Atari ST 1040. This computer allowed me to plug my keyboard synthesizer—at the time it was a Korg M1—into that MIDI port. MIDI allowed me to play and record performance data using the computer, which uses the MIDI data format along with audio software known as a MIDI **sequencer**, so called because it sequences playback data.

MIDI files contain no sample data; that is, they contain no audio data, only the **performance data**. When this performance data is played back into the synthesizer by the computer, the synthesizer generates the audio tones using the MIDI hardware (interface, cables, and ports).

MIDI records which keys on the synth or sampler keyboard were pressed and when, along with a keypress duration, how hard it was pressed (aftertouch), and similar performance nuances.

When MIDI files are played back through the synthesizer, it replicates the exact performance of a performer or composer; even though that person is no longer playing their performance track, their computer is. The way the MIDI data is used in MIDI sequencing software is that you can play your instrument track, recording your instrument play performance using MIDI data, and a MIDI sequencer will then play the performance for you, while at the same time you play a second instrument track alongside the first instrument performance track.

MIDI enables songwriters to assemble complex musical arrangements using only their computer. This is certainly less expensive than hiring a studio and musicians. If you want to be a songwriter, you can download open source MIDI software called **Rosegarden** at http://rosegardenmusic.com. Not only is Rosegarden a MIDI sequencer, but it also includes **music notation**, also known as **scoring**. This means that you don't have to know how to write notes and clefs on staffs in order to publish your music!

MIDI is beyond the scope of this book, and I cover it here only to educate you as to the history and scope of digital audio support for popular devices. MIDI holds a key role in the evolution of digital audio, as it is a major component of **music synthesis** and is an important (keyboard) tool for **sound design**.

MPEG3 Audio: The Popular MP3 Data Format

The most popular digital audio format worldwide is **MP3**. Most of you are familiar with MP3 digital audio files, which are found on music download websites like Napster. Most of us collected songs in this format to use on popular MP3 players and in our CD-ROM music collections. The reason the MP3 digital audio file format is popular is because it has a good **compression-to-quality ratio**, and because the codec needed to play MP3 audio can be found on every playback device.

MP3 can be an acceptable format to use in a VXF pipeline as long as you get the highest quality level possible out of it by using the optimal encoding work process. Because of patents, Audacity cannot include MP3 encoder software or distribute MP3 software from its website, so be sure to download and install a free **LAME encoder** for Audacity 2 if you want to work with MP3.

It's important to note that the MP3 codec will output a **lossy** audio file format, like JPEG does for images, where some of the audio data, and thus quality, is discarded during compression.

Some devices support an open source **lossless** audio codec called **FLAC**. This stands for **Free Lossless Audio Codec**. Support for FLAC is as widespread as support for MP3 due to its free nature (codec means encoder-decoder, as in **CO**de-**DEC**ode).

FLAC: The 24-bit Free Lossless Audio Codec

FLAC uses a fast algorithm, so the decoder is highly optimized for speed. FLAC supports 24-bit audio, and there are no patent concerns for using it. This is a great audio codec to use in VFX projects if you need high-quality audio with a reasonable data footprint. FLAC supports a range of sample resolutions, from 4-bit data per sample up to 32-bit. It also supports a wide range of sampling frequencies, from 1Hz to 65,535 Hz (or 65kHz), using 1Hz increments, so it is extremely flexible. From an audio playback hardware standpoint, I would suggest using a 16-bit sample resolution and either a 44.1 kHz or 48 kHz sample frequency, unless you are targeting HD audio.

OGG Vorbis: A Lossy Open Source Audio Codec

Another open source digital audio codec supported by most devices is the **OGG Vorbis** format. This lossy audio codec is brought to you by the **Xiph.Org Foundation**. The Vorbis codec data is most often held inside a file with a **.OGG** file extension, and thus Vorbis is commonly called the OGG Vorbis digital audio data format. OGG Vorbis supports sampling rates from 8 kHz to 192 kHz and supports 255 discrete channels of digital audio; it is supported in HTML5 as well as in Android OS.

Vorbis is quickly approaching the quality of MPEG4 HE-AAC (High-Efficiency Advanced Audio Codec) and Windows Media Audio Professional and is superior in quality to MP3, AAC-LC, and WMA. It's a lossy format, so the FLAC codec would still have superior reproduction quality over OGG Vorbis, as FLAC contains all of the original digital audio sample data.

MPEG4 Audio: Advanced Audio Coding (AAC)

All devices support MPEG4 AAC (Advanced Audio Compression) codecs, including the **AAC-LC**, **HE-AAC**, and **AAC-ELD**. These should each be contained using a MPEG4 file container (.3gp, .mp4, or .m4a file extensions). ELD stands for **Enhanced Low Delay**. This codec is intended for use in real-time two-way communication applications, such as a digital walkie-talkie or a Dick Tracy smartwatch app.

The simplest AAC codec is the AAC-LC (**Low Complexity**) codec, which is the most widely used. This will be sufficient for most digital audio encoding applications. AAC-LC will yield a higher quality result at a lower data footprint than the MP3 codec.

The most complicated AAC codec, HE-AAC (**High Efficiency)**, will support sampling rates from 8 kHz to 48 kHz, and both **Stereo** and **Dolby 5.1** channel encoding.

Because of software patents, Audacity will not include an MPEG4 encoder. Be sure to download and install the free **FFMPEG 2.2.2 encoder** for Audacity from http://lame.buanzo.org, and pronto!

PCM Audio: Pulse Code Modulated Codec

Finally, most popular platforms also support **PCM** (Pulse Code Modulated) codecs, commonly known as the Windows WAVE (.WAV) audio format, or Apple AIFF (Audio Interchange File Format). Most of you are familiar with this **lossless** digital audio format from one of these two popular operating systems. It is lossless because there is **ZERO compression** applied. PCM audio is commonly used for CD-ROM content as well as for telephony applications. This is because PCM WAVE audio is an uncompressed digital audio format, and it has no CPU-intensive compression algorithms applied to the data stream. Thus, decoding (CPU overhead) isn't an issue for telephony equipment or CD players!

For this reason, when you optimize a digital audio asset into various file formats, you can use a PCM as your "baseline" file format. You will probably also use PCM in your VFX project pipeline, as it is original (pristine) data and does not require any decoding (just like the telephony equipment and CD player).

Ultimately, the only way to find out which audio formats supported by popular devices have the best digital audio result for any given audio data is to actually encode digital audio in all the primary codecs that are well supported and efficient. I suggest you check out *Digital Audio Editing Fundamentals* (2015) from Apress, available at http://www.apress.com when you get the chance.

Summary

In this fourth chapter we took a look at both analog audio and digital audio concepts and principles, and at the data formats that compress and decompress your digital audio assets. We looked at sound waves, frequency, amplitude, sampling, bit-rates, codecs, MIDI, MPEG, OGG, Free Lossless Audio Codec (FLAC), Pulse Code Modulation (PCM), and High Dynamic Range (HDR) 24-bit HD audio.

In Chapter 5 you will learn about vector format new media concepts and terms and the principles behind 2D scalable vector graphics (SVG) and 3D OpenGL, both important asset types.

Chapter 5

The Foundation of 2D Vector for VFX: Point, Path, and SVG

Now that you have an understanding of the fundamental concepts, terminology, principles, and methods of data footprint optimization for your digital imagery, digital audio, and digital video new media content, it's time to get into 2D and 3D vector new media in this and the following chapter. Fusion supports one open source 2D vector format called **SVG**, or **Scalable Vector Graphics**, so that will make things easy. We can focus on SVG concepts and terminology in this chapter, as both GIMP and Inkscape support SVG. Then, in the next chapter, we can take 2D vector imagery into the third dimension with 3D vector imagery.

We'll look at how a digital illustration is created using points in 2D space, lines and curves connecting those points to create shapes, and color fills, gradients, or patterns inside of these 2D vector shapes. We will look at the open source digital illustration SVG data file format, which is supported by Fusion as well as other open platforms, like Android, Java, and HTML5.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_5

Digital Illustration Is Rendered, Not Stored

As you learned in Chapter 2, pixel-based digital imagery is technically called **raster imagery**, because an array of pixel values is rasterized to a screen, thus displaying the image created using these pixels. Digital illustration, or **vector imagery**, is not stored as an array of image elements (pixels). It's instead **drawn**, or rendered, to the screen, just like you would draw it if someone were watching you draw, only using instructions that the computer uses to do exactly what you did when you created it. This is the equivalent of the **MIDI** concept you learned about in Chapter 4, where the performance—in this case it is drawing; in that case it was composing music—is recreated by the computer processor, which renders it to the screen (SVG) or synthesizer (MIDI) using **playback instructions**.

With vector images, this is accomplished with coordinates in the 2D X,Y space, along with mathematics that define curves, which are conveyed using SVG instructions that look a lot like code. We will cover the basic constructs of vector illustration first so you understand how it all goes together. After that, we will look at how the SVG format turns these into instructions that can be processed in Fusion, Android, Java, or JavaFX using SVG-compatible classes. SVG can also be processed by HTML5 and CSS.

Vector Components: Vertices and Curves

Digital illustration vector images are composed of **points** using **coordinates** in 2D space, and **lines** or **curves** that connect those points together. We will be looking at concepts and terminology for these points and lines during this section. If you create a closed shape—that is, one where there are no openings through which a **fill** (color, pattern, or gradient) can escape—you can fill the vector shape so that the shape looks solid instead of empty.

The Vertex: The Foundation for 2D Shapes

The foundation for any 2D or 3D vector asset is called a **vertex**. Multiple **vertices** are required to create a line or **arc**, which require two vertices, or a **closed shape**, which requires at least three vertices, unless you are using curves, in which case you can do it with only two. Vertices are used in both 2D vector (SVG) data processing and 3D vector (OpenGL) data processing, both of which are integrated into Fusion 8.

Vertex data is outlined in SVG using **X,Y coordinates**, as you might have guessed, which tell the processor where a vertex is located in 2D space. Without these vertex coordinates, lines and curves cannot be drawn, as they must have an **origin** as well as a **destination** vertex coordinate as part of the line-drawing operation. Lines and arcs are examples of **open shapes**.

When we get into creating and looking at SVG data, you'll notice that these X,Y numeric pairs make up the majority of the SVG data, which can be contained using the XML format, or in a Java SVG object for Android. SVG data can be used in your JavaScript (HTML5) code as well as in JavaFX (Java 8 or Java 9) code; thus, it is compatible across each of these open-platform application development workflows, including your VFX processing pipeline.

An X,Y coordinate all by its lonesome is what is termed one dimensional, or **1D**. It takes two vertex coordinates to be considered two dimensional, or 2D, so a line or a curve—that is, an open shape—or a closed shape will be considered a 2D object.

The Path: Connecting Vertices to Create a Shape

A path is defined in SVG using a "path" data element. Both an open shape and a closed shape are technically a path, according to the SVG specification. An **SVG path** represents the outline of an open or closed shape that can be filled, stroked, or used as a clipping path. We will be covering these concepts in detail during this chapter, but in short a **fill** deals with the interior of a path, a **stroke** deals with the lines or curves that make up a path, and a **clipping path** is used for Boolean operations.

In SVG data an SVG path object represents 2D geometry used to outline a path object. In fact, in JavaFX, the class is actually called the `SVGPath` class! SVG path data can be defined in terms of SVG **commands**, which I will outline later in the chapter using Table 5-1. Some of these include a `moveto` command, which sets your current point; a `lineto` command, which draws straight lines; a `curveto` command, which draws **cubic Bézier** curves; the `elliptical arc` command, which draws an elliptical arc; and the `closepath` command, which closes a current shape, drawing a line to its starting point.

There are also advanced SVG commands.

Compound paths are also possible in SVG; these allow you to create complex, Boolean-shape special effects. For instance, you could use a compound path to create a hole in your shape.

These SVG paths can be used in Fusion to create a vector asset or to define other, more advanced things like interpolation.

Lines: The Simplest of the Path Components

The simplest way to connect point coordinates along a path is to use straight lines. Different shapes such as triangles, squares, pentagons, and hexagons can be created using the **lineto** (L) commands. There are three lineto commands: lineto, horizontal lineto, and vertical lineto, as outlined in Table 5-1.

To code an **octagon** using SVG, you would use a **moveto** (M) command from a point (vertex) at 60,0 and then draw seven lines using lineto commands. This would look like the following:

```
M 60 0 L 120 0 L 180 60 L 180 120 L 120 180 L 60 180 L 0 120 L 0 60
```

Next, let's take a look at the elliptical arc, which is a simple curve by nature with a complex set of specification data for its SVG command. This is why I usually stick with curves, from a modeling perspective, as you can create 2D paths using fewer data points (vertices) than with elliptical arcs. This is far more data optimized. This also holds true for 3D models, where polygon models that use straight lines (edges) are more data intensive, as they use more points and lines to represent a curved surface than mathematical formulas for an actual curve will utilize.

Elliptical Arcs: Circular and Elliptical Arcs

The elliptical arc uses a capital A (absolute) or lower-case a (relative). The arc command draws a segment of an ellipse. It takes the largest number of parameters of any of the curve drawing-related commands and takes the following basic format:

```
M x,y   A rx,ry   x-axis-rotation   large-arc-flag   sweep-flag   x,y
```

Here, M (moveto) x,y is your **starting point** for the arc, rx is the **x-radius** for the ellipse, ry is the **y-radius** for this ellipse, and x-axis-rotation is the number of degrees to **rotate** the x-axis. There are two on/off **flags** for large/small arc and sweep/no-sweep arc, and the final x,y coordinate is the **end point** for the arc.

It is important to note that setting both the x radius and the y radius (the rx and ry values) to identical values will create a **circle** instead of an ellipse, as this makes the curvature **symmetrical**.

The example I found online that is shown in Figure 5-1 allows you to modify whether your arc coordinates are absolute (A) or relative (a) to the starting point (defined by the red blob). An example of the following `M 125,300 A 225,100 0 1 1 375,300` elliptical arc command and coordinate sequence shows how an arc can be defined. If you want to include the missing segment for the ellipse at the bottom, you will deselect, by setting to zero, the large arc flag and the sweep flag, which should draw the smaller part of the arc and mirror it around an x-axis. The command for this should look like the following: `M 125,300 A 225,100 0 1 1 375,300`.

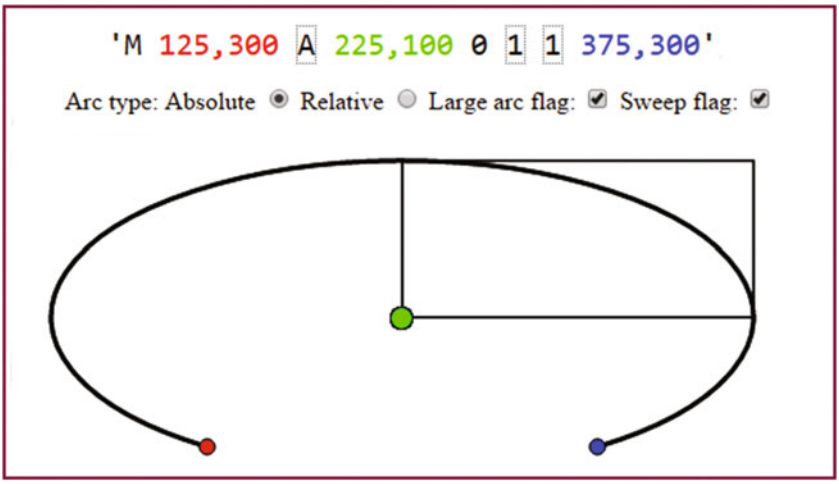

Figure 5-1. Elliptical arc with sweep and large arc flags on

As you can see, these `rx,ry` parameters create different angles, which distort your ellipse from being a circle into being an elliptical shape. There are several SVG curve generators on the Internet if you want to experiment with these parameters more.

Cubic Bézier Curve: Two Curvature Control Points Are Available

If you have ever used the Pen tool in Photoshop or GIMP, or used Inkscape, or any of the 3D modeling tools out there, then you are probably familiar with **Bézier curves**. I am not going to go into all of the math behind how curves are constructed mathematically, as this is a VFX fundamentals book and not an advanced book, but we will be looking at how to use open source tools to generate the digital illustration vector assets you will need for your Fusion VFX project pipelines. In a nutshell, you would draw **cubic Bézier curves**, using SVG commands, by defining your start and end points as well as two control points—one control point for the start point and one control point for the end point. These control the curvature of a curve (also called a **spline** in the industry) that is going away from the first data point and coming into the second data point.

The cubic Bézier curve command would utilize the following format:

```
M x,y   C (or c) x1,y1 x2,y2   x,y
```

The starting point is defined by `M x,y,` and the `C` (or c) defines an absolute or relative cubic Bézier curve type. The `x1,y1` is your control point for beginning the curve, and `x2,y2` is your control point for end of curve. Finally, your `x,y` coordinate at the end of the command string is an end point for the cubic Bézier curve. You can create this Bézier curve in real-time by using the popular open source Inkscape software. A Bézier curve can also be used in Fusion 8 for controlling data related to timing, transitions, and similar effects applications.

Quadratic Bézier Curve: A Single Curvature Control Point Used

Your inclination will be to assume that **quadratic Bézier** curves are more complicated, given that *quad* means four, and thus there are even more control points for this type of curve. However, the **exact opposite** of this is actually the case, because a quadratic Bézier curve actually has only **one control point** that connects to both the start and end point of the curve segment! Moving this point controls how a curve is shaped between the two points. So, if you are looking for a coordinates data reduction of 100 percent, as far as control-point specification goes, use quadratic Bézier curves. An SVG command specification for a quadratic Bézier curve will thus look like the following:

`M x,y  Q (or q)` **`x1,y1`** `x,y`

Note how the quadratic Bézier command requires only one single control point, which is then used as the control point for both start and end points. It's like the two control points in a cubic Bézier curve are connected as one control point that moves the curvature from the start point and into the end point at the same time. There are numerous SVG curve generators on the Internet if you want to experiment with the parameters.

The Fill: Filling Your Closed Shapes with Colors

Once you have defined your shape using lines, arcs, and curves, you can fill it to make it solid rather than hollow or empty. A **fill** can be a **color**, a **gradient**, or a **tiling image pattern**. You can fill an open shape if you like, and the (imaginary) line connecting the start point with the end point will define the fill boundary so that the fill does not go all over the place in your digital illustration. The fill operation, as well as the stroke operation, which we'll cover next, is what is known as a **paint** operation. Fusion 8 has extensive painting capabilities, so you may want to use SVG for 2D curve creation and Fusion for the painting operations.

Color Fill: Filling Your Shape with a Solid Color Value

To fill the octagon we looked at earlier in the line section with green, a `fill="green"` statement would be added after the data statement that creates the shape. Using SVG XML, these two declarations would be inside of a path XML tag using the following XML markup SVG data structure:

```
<path>
d = "M 60 0 L 120 0 L 180 60 L 180 120 L 120 180 L 60 180 L 0 120 L 0 60"
fill = "green"
</path>
```

Solid fill color is not as useful as gradients, however, as careful use of gradients can simulate a 3D result using 2D SVG graphics. Defining a gradient is more complex, so let's take a look at linear and circular gradients next.

Gradient Fills: Linear Gradients and Radial Gradients

There are two types of gradients in SVG, a **linear gradient** and a **radial gradient**. A linear gradient is the most common type of gradient that you will encounter, so we'll start with that type first. Much of what applies to how a linear gradient is set up will also apply to the radial gradient, which simply uses a different XML tag. I will show you how to set up a `<linearGradient>` tag using SVG in XML, and you can simply change it later to be a `<radialGradient>` to change your gradient type.

Gradients are defined in the `<defs>` or definitions tag in SVG XML. The `<defs>` tag goes inside the parent `<svg>` tag, and has the `<linearGradient>` tag as its child tag. Inside the `<linearGradient>` tag are two `<stop>` child tags. **Stops** are used to define the colors in the gradient, what percentage they take up in the overall gradient, and alpha (transparency) values for that section of the gradient. There must be at least two stops, and you can use any number of gradient sections that you need.

Make sure that your stop offset values add up to 100 percent in the end. Here is how you would fill your octagon with a red and yellow linear gradient; as you can see, it's much more complex than the octagon example:

```
<svg xmlns='http://www.w3.org/2000/svg' height="300" width="300">
  <defs>
    <linearGradient id="LinearGradient" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
```

```
  <path>
    d="M 60 0 L 120 0 L 180 60 L 180 120 L 120 180 L 60 180 L 0 120 L 0 60"
    fill="url(#LinearGradient)"
  </path>
</svg>
```

You wire the gradient into your fill using the `id="name"` parameter inside of the `<linearGradient>` tag and then reference that name inside of your `fill="url(#name)"` parameter inside of your `<path>` tag, as can be seen in the previous SVG XML markup.

Pattern Fill: Filling Your Shape with a Tileable Image Pattern

Patterns are also defined in the `<defs>` or definitions tag in SVG XML. The `<pattern>` tag goes inside the parent `<defs>` tag, and it has the `<image>` tag as a child tag. Inside the `<image>` tag is a reference to and specifications for the image asset. **Patterns** are seamless image tiles that fill a shape with a 2D **texture map**. We will be learning more about texture maps in Chapter 6, which will cover 3D vector concepts and terminology.

Make sure your pattern width and height values match up with your image width and height values. The image x and y position the start of the pattern at the upper-left corner, which is always location **0,0**. Here is how you would fill your octagon with an eight-pixel tiling image pattern; as you will see, it's an even more complex definition than your gradient:

```
<svg xmlns='http://www.w3.org/2000/svg' height="300" width="300">
  <defs>
    <pattern id="pName" patternUnits="userSpaceOnUse" width="8" height="8">
      <image xlink:href="data:image/filename.png"
             x="0" y="0" width="8" height="8">
      </image>
    </pattern>
  </defs>
  <path>
    d="M 60 0 L 120 0 L 180 60 L 180 120 L 120 180 L 60 180 L 0 120 L 0 60"
    fill="url(#pName)"
  </path>
</svg>
```

I am showing you how to do this because even though this is a VFX fundamentals book I want to include some complexity, as VFX is a complex undertaking and topic. A large percentage of Apress titles are targeted for programmers or developers, so I'm covering SVG, XML, HTML5, JavaFX, Java, and Android. Many of you would use software such as Inkscape to create vector artwork, and then export it to SVG.

At the end of the workday, however, you'll need to know how to bridge the SVG command structure with the Fusion project pipeline, and possibly later with application code, so I'm going to cover the basic SVG commands during the chapter so that you have knowledge of SVG basics for VFX applications development.

The Stroke: Controlling How Lines and Curves Look

Finally, let's take a look at how to **stroke**—or **color**, **style**, or **thicken**—the lines, arcs, or curves that you created using these SVG commands. The stroke parameters allow you to define stroke **color**, **opacity**, **width** in pixels, **dash array pattern**, and how lines will be **capped** or **joined** together using **round**, **square**, or **bevel** constants. Let's add these stroke-related parameters to a `<path>` we created earlier for the octagon and give it a three-pixel-thick black border with round corners, a dashed line, and 50 percent opacity. This is accomplished via the following SVG XML markup:

```
<path>
d = "M 60 0 L 120 0 L 180 60 L 180 120 L 120 180 L 60 180 L 0 120 L 0 60"
fill = "green"
stroke = "black" stroke-width = "3" stroke-dasharray = "5, 10, 5"
stroke-linecap = "square" stroke-linejoin = "round" stroke-opacity = "0.5"
</path>
```

Next, let's finish up by looking at the primary SVG data commands that you'll be using when creating 2D vector assets for your Fusion 8 projects, probably by using GIMP and Inkscape.

SVG Format: Coding Vector Shape Data

There are ten different letters that can be utilized with the numeric (X,Y data point coordinate location) data in SVG data strings. Each has an uppercase (absolute reference) and lowercase (relative reference) version. As you can see in Table 5-1, SVG data commands provide you with a great deal of flexibility for defining custom curves for your VFX project pipelines. You can even combine all of these scalable vector-graphics SVG commands with code to create **interactive** vector artwork that has never before been experienced. You can then add the VFX content using the same Java, JavaFX, HTML5, and Android code.

Table 5-1. SVG Commands You Will Use for Creating SVG Path Data

SVG Command	Symbol	Type	Parameter	Description of the SVG Command
moveto	**M**	*Absolute*	X, Y	**Define a Start** of Path, at the X,Y coordinate, using **absolute** coordinates
moveto	**m**	*Relative*	X, Y	**Define a Start** of Path, at the X,Y coordinate, using **relative** coordinates
closepath	**Z**	*Absolute*	None	**Close SVG Path**, by drawing line, last to first point
closepath	**z**	*Relative*	None	**Close SVG Path**, by drawing line, last to first point
lineto	**L**	*Absolute*	X, Y	**Draw Line** from the current point to the next point
lineto	**l**	*Relative*	X, Y	**Draw Line** from the current point to the next point
horizontal lineto	**H**	*Absolute*	X	Draw **Horizontal Line** from the current point to the next point
horizontal lineto	**h**	*Relative*	X	Draw **Horizontal Line** from the current point to the next point
vertical lineto	**V**	*Absolute*	Y	Draw **Vertical Line** from the current point to the next point
vertical lineto	**v**	*Relative*	Y	Draw **Vertical Line** from the current point to the next point
curveto	**C**	*Absolute*	X,Y, X,Y, X,Y	Draw a **cubic Bézier** curve from the current point to the next point

(*continued*)

Table 5-1. (*continued*)

SVG Command	Symbol	Type	Parameter	Description of the SVG Command
curveto	**c**	*Relative*	X,Y, X,Y, X,Y	Draw a **cubic Bézier** curve from the current point to the next point
Short and smooth curve	**S**	*Absolute*	X,Y, X,Y	Draw a **short cubic Bézier** curve from the current point to the next point
Short and smooth curve	**s**	*Relative*	X,Y, X,Y	Draw a **short cubic Bézier** curve from the current point to the next point
quadratic Bézier curve	**Q**	*Absolute*	X,Y, X,Y	Draw a **quadratic Bézier** curve from the current point to the next point
quadratic Bézier curve	**q**	*Relative*	X,Y, X,Y	Draw a **quadratic Bézier** curve from the current point to the next point
short quadratic Bézier	**T**	*Absolute*	X,Y	Draw a **short quadratic Bézier** curve from the current point to the next point
short quadratic Bézier	**t**	*Relative*	X,Y	Draw a **short quadratic Bézier** curve from the current point to the next point
elliptical arc	**A**	*Absolute*	rX, rY, Rot	Draw an **elliptical arc** from the current point to the next point
elliptical arc	**a**	*Relative*	rX, rY, Rot	Draw an **elliptical arc** from the current point to the next point

An optimal way to see how to use these powerful SVG data path-drawing commands is to learn a work process for creating SVG data with vector illustration tools and then export work flows. This is possible using GIMP and Inkscape. If you're a VFX artist, you can use the SVG constructs as control data for non-graphical uses. If you're interested, I cover 2D vector and SVG further in *Digital Illustration Fundamentals* (2015, Apress).

Summary

In this fifth chapter we took a look at **2D vector illustration** and its related concepts and principles, as well as at the SVG format. We looked at how vertices, lines, arcs, curves, fills, patterns, gradients, and strokes contribute to 2D vector content creation. In Chapter 6, you will learn about **3D vector concepts**.

Chapter 6

The Foundation of 3D Vector for VFX: Models and OpenGL

Now that you have an understanding of the fundamental concepts, terminology, principles, and methods of data footprint optimization for 2D digital imagery, digital audio, digital video, and vector digital illustration new media content for VFX project pipelines, it is time to get into the more complicated area of 3D new media assets that can be used with Fusion's advanced 3D feature set.

Fusion uses an open source, real-time 3D vector rendering platform called **OpenGL**, which stands for **Open Graphics Library**. Unlike SVG, which is relatively simple in comparison with OpenGL, I can't cover 3D in just a couple of chapters—or even within a couple of books, for that matter. I'll cover most of the topics that are important for 3D- and GPU-centric Fusion 8, however.

In this chapter, I'll give you as much of an overview of 3D as I can, but it is far more expansive than your 2D new media assets, and not just because it has the third dimension (depth) or fourth dimension (animation), which digital video and digital audio also have. Vector 3D can be used to create photorealistic environments out of thin air. It has been primarily driven by the 3D game industry as well as by the Hollywood film industry. You can take advantage of all these 3D advancements in Fusion.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_6

We'll look at the basic concepts, principles, and formats used in 3D vector images, 3D animation, 3D modeling, 3D texture mapping, 3D particle effects, 3D physics simulations, character animation, and all of the similarly complex, 3D-related topics.

3D New Media Assets: 3D Vector Content

3D vector assets are primarily comprised of 3D vector **geometry**, which is made up of vertices in 3D space that are **surfaced** with raster images, which you learned about in Chapter 2. 3D can be animated using **keyframes**, which we covered in Chapter 5 and will also cover in this chapter. Characters can be created in 3D by defining skeleton rigging, facial animation, lip-syncing, skin, muscles and cloth.

Everything we have covered in the first five chapters is available for use in 3D, and therefore in Fusion VFX pipelines. Digital images can be used to create **shaders** that texture map a 3D model's geometry. Digital video can provide animated texture maps, and SVG geometry will be used to create 3D geometry using 2D. Or, SVG can be rendered on 3D surfaces for texture map effects, such as transparency (opacity) maps, and digital audio could be used for sound effects or to allow your 3D character to speak.

Let's start from the vertex on up, like we did with 2D vector illustration. I will show you various attributes that take a 3D asset from being **3D geometry** to being a **3D model** to being a **3D hierarchy** to being a **3D animation** to being an **i3D object**. This is the least commonly used new media asset type to be found in HTML5 (websites, HTML5 OSes, tablets or smartphones) using **WebGL 2.0**. Android 7 uses OpenGL ES 3.2 and a new Vulkan rendering engine, and Java 7, 8, and 9 via JavaFX all use **OpenGL ES 3.1**.

The Foundation of 3D: The Geometry for the Model

The lowest level of 3D new media assets is, just as with 2D SVG new media, the **vertex** and its connection with other vertices. With 3D, connections between vertices become more complicated, not only because these occur in three dimensions, but also because 3D introduces **triangular faces** commonly called **polygons**, or *polys*, or **rectangular faces** called **quadrilaterals**, or *quads*. The connections between vertices are called *edges*. Faces are created using vertices and edges. Before 3D geometry is texture mapped, it's referred to in the 3D industry as a *mesh* or a *wireframe*, since that is what it looks like before it is **texture mapped** or *skinned* using digital imagery. We will cover this in the next section.

3D Starts with Data Points in 3D Space: Origins of the 3D Vertex

Just like the 2D vertices—*anchor points*, as they're called in Illustrator, or *path nodes*, as they are called using Inkscape—the vertex is the foundation of your **3D geometric** or **3D organic** modeling. Just like 2D polygons use straight lines, 3D polygonal or geometric models also use straight lines, whereas 3D organic models use 3D algorithmic curves defined using NURBS, Catmull-Rom Splines, Bézier Splines, or Hash Patches curve algorithms. A vertex defines where your model's infrastructure, whether that is edges or splines, is going to be located in 3D space. 3D scanners generate "point clouds" of vertices, and the **Alembic** format supported in Fusion tracks vertex characteristics, including 3D spatial animation. Alembic is a data format that defines all of its 3D data using vertices, and is supported in Fusion 8.

In 3D geometry, vertex data can hold **surface color** data, **surface normals** data, **UVW texture mapping** data, as well as the vertex **X,Y,Z location** data. This is similar to pixels for 2D, which hold X and Y data, RGB color data, and alpha channel data.

Connect the 3D Vertices: Edges Connect the 3D Vertices

3D geometric models use something called an **edge** to connect two vertices. An edge is a **vector**, or straight line, so it looks like the edge of a razor in 3D space, as you can see in Blender in Figure 6-1. Three edges are needed to form polygons, and four edges are needed to form quads. Polygons and quads are what are called *faces*, and we're going to cover those in the next section. When you're modeling 3D geometric objects, you can select components of the model, such as vertex, edge, or polygon.

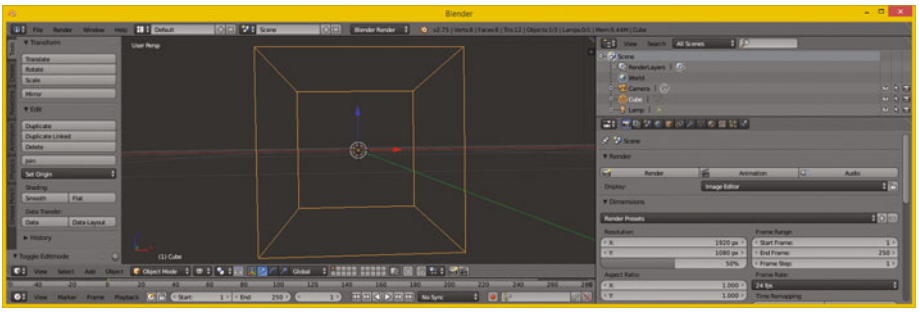

Figure 6-1. *Blender 3D geometry showing eight vertices and twelve edges*

If you've created your 3D geometry using a more advanced, organic, spline-based model paradigm, such as that used to create characters, like NURBS using MoI 3D, or quads using SILO 2.3 (which is only $159), or using Hash Patches in Animation:Master (which is only $80), then you will then need to "decimate" your model into polygons (triangles). This is what the GPU hardware used with Fusion does during 3D rendering if you are using spline-based or quad-based 3D models. Fusion does support quads, NURBs, and splines internally, as it comes very close to being a full-fledged 3D software package itself, as you will see during the course of this book.

The algorithmic process of **decimation** from other spline- or patch-based representations of 3D geometry into triangles or polygons turns infinitely smooth curves used in these modeling paradigms into a data-heavy collection of triangular faces that have straight edges and have to use more (and smaller) triangle data to approximate smooth curves. This works for VFX pipelines that are rendered out to frames, but not in interactive games.

Decimation is done by using a decimation or **smoothness** numeric factor via a slider or dialog setting. This will be the decimation function inside of the 3D software, or it can sometimes be supplied in a **File Export** function, which outputs the spline-modeling format from your curve-based modeler into a polygonal geometric model format. A great example of this can be found in the **Moment of Inspiration** V3 NURBS modeling software. Now, let's take a look at how polygons, quads, and splines form 3D **surfaces**.

Surfaces: Three Edges Form a Polygon, Four Edges Form a Quad

Once you have three edges together in the form of a triangle, you have a polygon. This can be used as a **surface** and can host a skin, or texture, to make the 3D data look more realistic. The rule of thumb is, the more uniform (square) a triangle is, the better it will render. **Slivered**, or long and thin, triangles, may cause rendering *artifacts*, or visual anomalies, but often do not. You feeling lucky? Then use slivered polygons. Modelers often prefer modeling with quads and keeping them as square as possible using quad modelers such as Silo 2.3 from NeverCenter, which is only $159 and can be used for 3D character modeling.

Character modeling is usually accomplished using organic spline modelers, such as SILO 2 or Hash Animation:Master, which uses highly efficient, proprietary **Hash Patch** curve algorithms.

Once you have a surface—which, for use on devices using OpenGL, will need to be a triangle—you will have also defined your **surface normal**, which you will be learning about next.

Direction a Surface Is Facing: The Concept of Surface Normals

The function for this surface normal is fairly simple. A surface normal tells a rendering engine the direction a surface is facing—inward or outward. The same logic would apply to a vertex normal; it would show the rendering engine which side of your 3D geometry to process for surface rendering.

If you know how to turn the "show normals" feature on in the 3D software, you can see the 3D face **surface normals**, which should be displayed at the exact center of the face, as you can see shown using tiny short lines in light green in Figure 6-2.

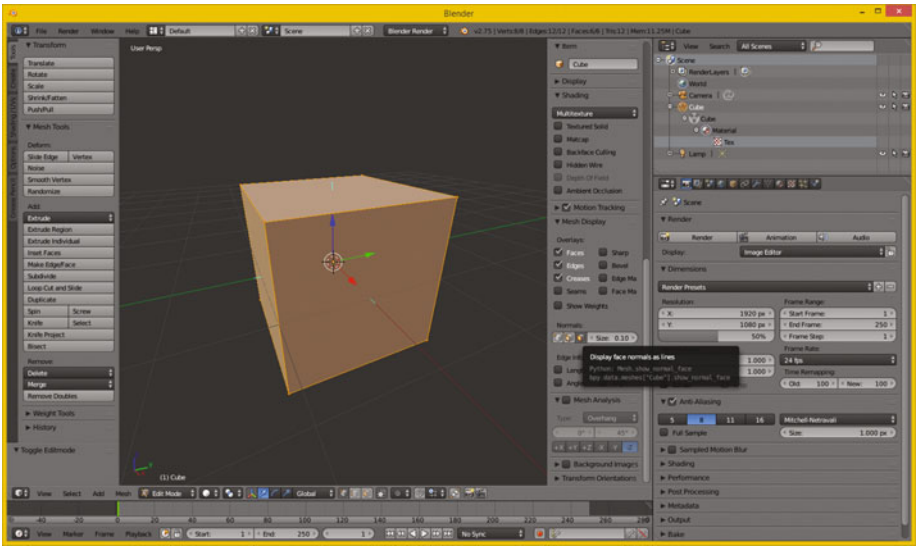

Figure 6-2. *Displaying face normal, as short green line on face*

As you can also see in Figure 6-2, two of the normals shown are actually aligned with the x (red) and y (green) axes, which intersect the cube at 90 degrees. Your axis guide, in the lower left corner of the 3D Edit Mode view, shows you which axis is x and which is y and which is z.

There are buttons in Blender for showing **vertex normals**, which point outward from the vertex, so for this model vertex normals point out diagonally from the corners of the cube at 45 degrees, the exact opposite result from the face normals, which point straight up, at 90 degrees, from the center of each face.

If the normals for this cube geometry had been pointing inward instead of outward, this cube would not be visible at all when rendered. There's a **flip normals** operation (algorithm) in 3D software that is used to reverse your normal directions. This is done for the model universally; all normals are flipped 180 degrees. If you ever can't see your 3D model in Fusion, use the flip normals feature to correct this file export anomaly.

Flip normals would be utilized when you render the scene and your imported 3D object is not visible after you render the scene. This is almost always because a 3D import utility points (flips) the surface normals for the imported 3D geometry in the wrong direction. An exporter from a 3D modeling tool might have exported the surface normal in the wrong direction, relative to the software you are importing them into. This is a fairly common occurrence, so expect to use flip normals regularly if you are going to be working with 3D new media assets in VFX software.

If you need two-sided geometry, like a house, for instance, where the 3D geometry has to render from the outside as well as from the inside because you have to be able to navigate through the models (this is common in virtual worlds, for example), you will have to create geometry faces that are **double-sided polygons**. You'll also need to apply a double-sided texture map, which I'll be covering during the next section, which covers 3D texture mapping concepts and terminology.

It is important to note that for i3D, using double-sided geometry and double-sided textures will require significantly more real-time rendering-engine processing by your CPU and FPU. This is because Interactive 3D, such as is found in game engine applications (as well as in the user interface for Fusion 8) renders the 3D geometry, textures, and animation in real-time. This requires a powerful GPU, which is why I recommended nVidia and AMD GPUs in Chapter 1 when we covered hardware requirements for Fusion 8.

Smoothing a Faceted 3D Surface: Using Smoothing Groups

You have seen 3D models that are rendered as solid (instead of wireframe) but look like they are chiseled; that is, you can see the polygons (faces) rendered as if they were flat, as in Figure 6-3. This is often called **flat shaded**, and no smoothing is being applied by the rendering engine. If you render the same geometry with smoothing turned on, this effect disappears, and the 3D geometry looks like it was intended to look, which is infinitely smooth, like it was created using splines, when it is in fact using polygons. It is more efficient to have your rendering engine do the smoothing than to have a ton of mesh (polygon) data. The renderer applies smoothing using something called a **smoothing group**, which is applied to each face to tell the renderer when to smooth between two faces, and when not to smooth

between faces, which leaves what is commonly referred to as a seam. A smoothing group uses simple integer numbers. If the numbers match on each side of an edge—that is, for each adjacent face on the opposite side of that edge—it renders as a smooth color transition. If the numbers are different, it renders a seam, and the edge will be clearly visible, as color gradients on each side of that edge will be different, so the color gradient is not seamless across the two faces (polygons).

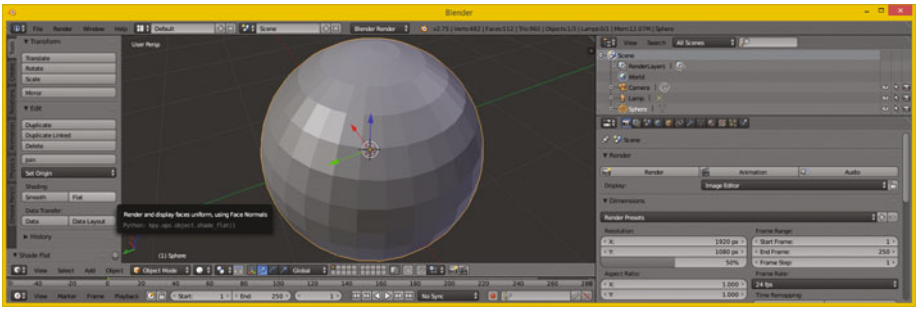

Figure 6-3. Set faceted shading via Transform ➤ Shading ➤ Flat

In some 3D software packages, such as Autodesk 3D Studio Max, you can see this smoothing group numbering schema right in the user interface. In others such as Blender, the numbering is hidden and the smoothing groups function is exposed using commands such as Mark Seam, Clear Seam, Mark Sharp, Clear Sharp, and so on. These can be found in the Blender Edges Menu.

Some 3D modelers (people, not software) will make the mistake in Blender of trying to expose a seam or a sharp edge in the 3D geometry by actually splitting an edge in the 3D geometry itself. This will achieve the right visual effect, but might cause a problem down the line during a 3D geometry **topology refinement**.

The term *topology* references how the surface of the mesh for your model is designed—whether it uses triangles, quads, or splines, for instance—and how these come together, especially at the seams of where model components attach, such as arms to a torso, legs to the waist area, or head (neck) onto shoulders.

Having to split geometry edges to achieve the seam can be avoided by instead using the Mark Seam, or a Mark Sharp, edge-modifier operation in Blender. This should affect only the smoothing group—that is, how geometry is rendered—and not the model topology. As you can see, 3D is a very complex media type.

These Mark Seam, Clear Seam, Mark Sharp, and Clear Sharp Blender seam modifiers are smoothing-group-based and therefore achieve their smoothing (or edge-seam) effect without actually affecting the 3D geometry topology itself.

A modifier in Blender is applied right before rendering and therefore doesn't affect the actual mathematical topology for the underlying 3D geometry. If you are familiar with the term *topography* used in mapping, know that topology is very similar but is a 3D term referring to how the surface for 3D geometry has been constructed during a 3D modeling process.

Using the Blender modifier is always a more flexible i3D content-creation approach as it applies a smoothness, or other desired effect or result, at the **rendering-engine level**, and not at the 3D geometry–topological level. Decimation would perform this geometry smoothing at a topological level directly on the mesh geometry. Using a modifier will leave a 3D mesh intact, and is ultimately simpler. Fusion 8 works in much the same way.

You can apply smoothing to a 3D model globally—that is, to all faces at the same time—by using the **Transform** panel and the **Shading** panel area, and then clicking on the **Smooth** button, as is shown in Figure 6-4. This applies the smoothing groups to the entire model, resulting in a smooth surface color gradient.

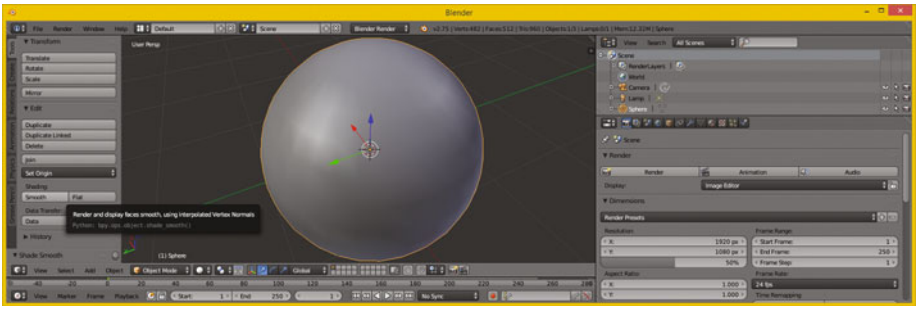

Figure 6-4. Set smoothing using Transform ➤ Shading ➤ Smooth

Next, let's take a look at how to apply a skin to your mesh geometry surface using something called texture mapping.

Skinning the 3D Model: Texture-Mapping Concepts

Once your 3D geometry, which is the foundation for your 3D model, is complete, you can apply **texture maps** to it to create a **solid appearance** for your 3D model. Texture maps can also be used to add detail and special effects to a 3D model, making it appear more realistic. If you are wondering what the difference is between 3D geometry and a 3D model,

3D geometry is just the mesh or wireframe, whereas a 3D model should have texture maps already applied. If you purchase third-party 3D models, you expect them to look like what they are supposed to be when you render them complete with texture, instead of just being flat gray, which is what a rendered model will look like without texture maps applied, as can be seen in Figure 6-4. Luckily, Fusion has the ability to texture 3D models!

Texture Map Basics: Channels, Shading, Effects, and UVW Maps

Texture mapping is almost as complex an area of 3D as creating topologically correct geometry is. In fact, each area of 3D is equally complex, which is what makes 3D the most complex new media type overall, with the possible exception of visual effects! This is why feature films employ 3D VFX artists to specifically focus on and handle each of these areas we are looking at in this chapter. Texture mapping is one of the primary areas in 3D new media asset production that is able to use 2D vector as well as 2D raster image and video assets.

It's important to note that there is also a more complex area of 3D texture mapping that uses **3D texture algorithms**. It is commonly termed **volumetric texturing**, and it uses algorithms to create true 3D texturing effects that go all the way through the 3D object as though it were solid and not hollow mesh. That is, it is a 3D object that requires double-sided texture maps.

A basic concept behind texture mapping is taking assets such as those you learned about during the first five chapters and applying them to the surface of 3D geometry. The question is, how do we align these to a model?

This is accomplished by using **U,V,W mapping coordinates**. These 3D mapping coordinates show how you want the 2D image (plane) oriented to, or projected on, the 3D geometry surface topology.

UVW is different than XYZ, but they represent the same dimensions (width, height, depth). Letters other than XYZ are used to avoid confusion, so they used the three letters in the alphabet before XYZ; that is, UVW. You use UVW to reference applying your texture map orientation, and you use XYZ to reference your 3D geometry orientation.

You can add more than one texture map to the surface of your 3D geometry. This is accomplished using **texture channels**.

These are analogous to the color and alpha channels that you use in 2D images to define each pixel's characteristics. OpenGL ES platforms, such as Android, HTML5, and JavaFX, currently support four of the most important texture channels. Fusion 8 supports numerous texture channels, which you'll see later on.

Primary texture channels include the **diffuse** texture map (basic RGB color values), a **specular** texture map (where surface looks shiny or dull), an **illumination** texture map (called a **glow** map), and the **bump** texture map.

3D software supports other advanced texture map channel types used for additional texturing effects. To bring them into Fusion, you will want to use the FBX or Collada 3D file format.

Baking texture maps involves rendering all the texture's channels into a primary diffuse texture map, since this is what devices support. This provides a similar visual result to what you will get when you render your 3D object in your 3D package or in Fusion 8, and you want to use that texture map in OpenGL.

As you can see in Figure 6-5, Blender uses a SceneGraph, just like most modern day i3D software packages will. In fact, Java's JavaFX offers the same SceneGraph functionality. A Scene Graph is a visual graph of the 3D scene construction, as seen on the right in Figure 6-5.

This sphere geometry and its texture mapping are grouped together in the SceneGraph hierarchy, which I have expanded for you on the top-right of Figure 6-5. This texture-map definition contains several texture channels; in this case diffuse (color) and specular mapping. These control the surface characteristics—such as shininess, metallicity, and similar looks—that you want in order to achieve your photorealistic, 3D-rendered output result.

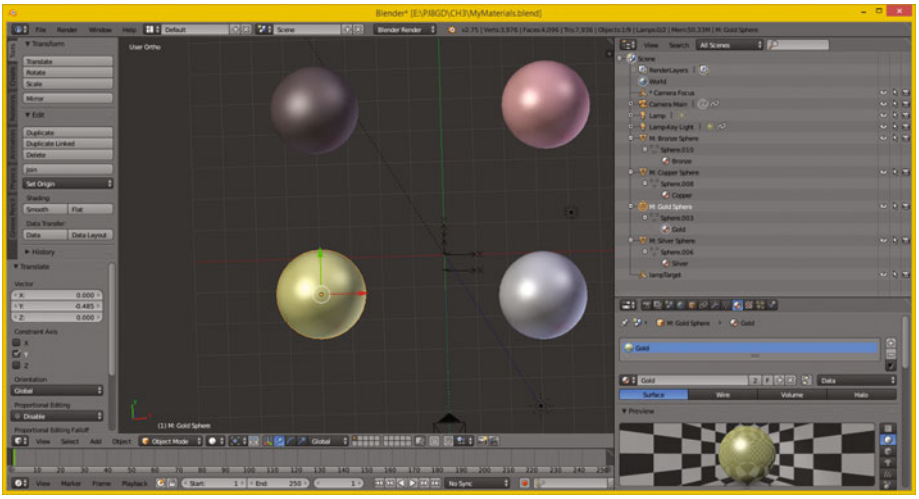

Figure 6-5. *Using the Blender 3D SceneGraph to apply a gold texture map and shader*

As time goes on, OpenGL ES 3 could add even more texture channel support. This would give developers increased visual flexibility regarding their 3D new media asset usage, as transparency areas (opacity maps) and surface details (normal maps) are two of the most important areas of advanced texture mapping support that needs to be added to Android, Java, JavaFX, and HTML5.

A collection of texture channels, and any code governing these channels' relationship to each other, as well as how they will be composited together and applied and rendered relative to each other, is called the **shader definition**.

Shaders are also commonly referred to as *materials* in the industry. We will be covering shaders and shader languages in the next section of this chapter, as this is another specialized (and complex) area of 3D and VFX project design for application development. As you can see, there are a lot of layers to VFX and 3D production, and, in fact, where shaders are concerned it can get fairly complex, as it involves blend modes and channels.

Shader Design: Shader Channels and GLSL Shader Language

As are each of the areas of 3D object creation covered in this chapter, texture-map **shader design** is an art form, in and of itself. Hundreds of shader artisans work on 3D movies, popular console games, VFX, television shows, and the like ensuring that the shaders used to texture or skin the 3D geometry make the resulting 3D model look as real as possible. This is often the primary objective of 3D and i3D—to replace more expensive video camera shoots and subsequent re-shoots by creating virtual worlds and having computer render farms create all the camera movement for you, then turning the 3D data into pixels (imagery), frames (video), or experiences (games, apps).

The basic shader is made up of a series of vector shapes and raster images, or algorithmic, volumetric textures, held in different types of texture channels. These texture channels use the vector and raster assets to apply various types of effects, such as **diffusion** (RGB color channels), **opacity** (transparency), **glow** (self-illumination), **specularity** (surface characteristic), **environmental** (surroundings), **bump** (height), **normal** (topology), and similar detail-effects channels that increase photorealism.

On top of this, advanced shader languages, like the **Open GL Shader Language**, or **GLSL** for short, use code to define just how these channels will interrelate with each other, how to apply or process the data contained in these channels, and how to perform other, more complex applications of the data within these channels based on complex factors such as time, orientation, or an X,Y,Z position in 3D space.

The complex nature of shaders also means that a render-time processing of the shader is more time consuming, and more processing-cycle consuming, the more complex a given shader becomes.

This is probably the primary reason OpenGL ES3 currently supports the four basic, easiest-to-process shaders. As hardware becomes more powerful (4- or 8-core CPUs in consumer electronics products), OpenGL ES will probably add in the last two important shader channels: opacity (alpha channel) and normals mapping.

Once texture channels are defined inside of the shaders, you will need to **orient** the 2D assets to the 3D geometry, which is done using **texture mapping coordinates**. Each channel will have its own coordinate system by which to apply each of the effects to the mesh (geometry) as needed to achieve the desired effect.

This is accomplished using **UVW mapping**, which we'll cover further in the next section before we move into the fourth dimension and cover 3D animation principles and terminology.

Texture Map Orientation: Projection Types and UVW Coordinates

It is important to **align** the detail features in your 2D texture map channels—especially the foundational diffuse color channel, as it paints or colors the surface of your object—to your i3D geometry correctly. If you don't do this correctly, some fairly odd, or at least visually incorrect, results appear when the 3D object is rendered. The alignment needs to be done in 3D space—for texture mapping this is **U,V,W**—especially with a volumetric "true 3D" texture, and also for 2D textures, in order to define how they will project onto, on top of, or envelop around i3D geometry.

One of the easier ways this can be done is by applying a **texture map projection type** and its related settings. This will then automatically set your **UVW mapping** numeric values for you. These **UVW map coordinate** values will define how your 2D imagery plane maps onto the 3D geometry in 3D space. This provides your "mathematical bridge" between your 2D space and your 3D space. UVW map **floating-point** values can be set or tweaked manually in order to fine-tune the visual result of the texture mapping.

The simplest type of projection is **planar projection**, as a plane is a simple 2D square. You would visualize this type of mapping as if your 2D texture map image were in front of the 3D object and you were shining a light through it; it would look like the colors in your diffuse texture map were projected onto the 3D object. A planar projection is also the simplest for the computer to process, so use this if you can get the result that you will need for the VFX project pipeline. You can also create planar mapping inside of Fusion 8.

Planar mapping is often used for static 3D objects, such as rendered 3D imagery. Once you move the camera around to the sides of the 3D model, this type of projection mapping will not provide photorealistic results for dynamic (animated) projects.

Camera projection is very similar to planar projection, as camera projection projects the texture from the camera lens onto a 3D object surface much like a slide projector would do. The projection is 100 percent parallel to the front of the lens. This should be used for projecting video backgrounds onto your scene so that you can model, or eventually animate, your 3D assets in front of the projection. If a camera moves, camera projection mapping will stay parallel with the front of your lens. This is sometimes termed *billboard mapping mode* (or billboard projection).

Your next simplest type is **cylindrical projection**, which provides more of a 3D application for your texture map than do the inherently 2D planar or camera projection of a texture map onto a 3D object from a single direction. A cylindrical map surrounds the object in your up and down (a 3D z-axis) dimension, projecting your image all the way around your object! If you walked around the 3D object, there would be unique texture details in another dimension, which planar or camera projection do not provide. Make sure your texture maps **tile seamlessly** along their y-axis!

An even more complex projection mapping type is called a **spherical projection**. This can provide an even more complete 3D application of your texture map than the cylindrical projection will. Spherical projection attempts to address all three—x-, y-, and z-axis—projection directions. Again, you want to make sure your texture map is tileable to avoid any visual seams.

Similar to a spherical projection is a **cubic projection**. This is like having six planar projections in a cube format and gives a result similar to spherical projection. It uses a special **cubic texture map** data format.

When you apply the cubic projection mapping type to your i3D object, the object's faces are assigned to a specific face in a cubic texture map, as you may have guessed. This is based on the orientation of each of the 3D object's polygon normal, or by the proximity of the face to the cubic texture map (coordinate) UVW mapping space. The cubic texture is then projected from the faces of the cubic texture map using planar projection methods.

If you use volumetric textures, the spatial projection is a three-dimensional, UVW texture projection that is projecting through the 3D object's volume. It is typically used with procedural or volumetric textures, or textures that need to have an internal structure, such as wood, marble, sponge, agate, and so forth. If you slice a 3D object or transform texture map coordinates relative to the 3D object, different parts of the volumetric or procedural texture will subsequently be revealed.

There's also a simpler texture mapping method called UV mapping (no W dimension). This applies your textures in two dimensions, instead of three, and is easier to process as it has less data. You can also use Fusion 8 to texture map your imported model. Now, we are ready to take a look at some 3D animation concepts.

3D Animation: Keyframes, Motion Curves, and IK

After you have created your 3D geometry and texture mapped it using shaders and UVW mapping coordinates, you may want to make it move in some fashion, such as flying a helicopter model. Concepts you learned for digital video assets in Chapter 3 apply equally as well to 3D animation assets, as you might guess, as both use **keyframes**. 3D software packages have what are generally termed "track editors," which allow you to add keyframes and a motion curve to tracks. In fact, Fusion has exactly the same advanced functionality for creating visual effects, as you will see during this book. Each **track** will relate to a 3D model, and if a 3D model uses sub-component grouping, then there can be tracks for groups and sub-groups as needed to achieve any complex animation or simulation necessary for your VFX project, application, game, or i3D virtual world.

Simple Linear Animation: Tracks, Keyframes, Loops, and Ranges

The simplest type of 3D animation, and 2D animation for that matter, is **linear animation**, which is useful for many types of 3D animation. Linear animation will use the least amount of processing power, so it is the most efficient. If you can use linear animation to accomplish your 3D animation objective, use the fewest number of tracks you can and the fewest number of keyframes, as this will use the least amount of system memory.

If animation motions are repetitive, use a **seamless loop** instead of a long range containing duplicate keyframe data. One seamless motion loop can take up less memory than a long range, especially when that range contains multiple copies of the same motion, which would be heavily redundant. So, using **looping** is a great optimization principle where linear 3D animation is used.

Next, let's take a look at some of the more complex types of animation, including those that aren't linear. They will not be in a straight line, with evenly spaced keyframes. We'll also take a look at character animation and procedural animation, which is used for things like physics simulations and particle systems. Fusion 8 has advanced features for all of these powerful visual effects capabilities, and can bridge to 3D software for any capabilities that it doesn't feature natively.

Complex Non-Linear Animation: Motion Curves and Interpolation

A more complex type of non-linear animation, which is less regular and often looks more realistic, especially where human motion and simple physics simulation is concerned, will implement a **motion path** for the animated 3D object or element (a sub-object in your hierarchy) to move along. To add even more complexity to the motion along that path, it is possible to use a **motion curve** so that the movement itself can speed up or slow down, simulating things like gravity, friction, or bouncing. The mathematical algorithms that are represented visually using these motion curves are called **interpolators**. JavaFX has an **Interpolator** class that contains a wide variety of the most standard (yet quite powerful, if used effectively) motion curve algorithms, and Fusion features this capability as well, as you may have guessed. We will get into Fusion in the next chapter.

A good example of non-linear, irregular-motion keyframing would be a rubber ball bouncing down the windy road. The curved path of the road would use a **motion path** to make sure the ball stays on the road curvature and that the ball floor conforms to the slope (angle) of that road. The bouncing of the ball should use a **motion curve**, also called a **motion interpolator**, in order to make each bounce look more realistic regarding the timing of the acceleration or deceleration of its movement through space. In this case, this is how your ball should react to the ground.

Complex physics simulations could be done using keyframes if you wanted to spend months of your time, as could character animations. However, it is actually easier to write algorithms and code routines to achieve this than it is to lay down motion paths, curves, or keyframes to try and simulate these phenomena.

Therefore, we will cover character animation principles and procedural animation principles next, as we are progressing from the less advanced concepts to more advanced concepts.

Let's get an overview of character animation next, as it is the next type of 3D animation that is supported in Fusion 8.

Character Animation: Bones, Muscles, Rigging, and IK

An even more complex type of animation is character animation, and character animators are one of the most popular positions on a 3D film, game, or television content production team. Character animation involves a number of complex layers, including setting up **bones** using **inverse kinematics** (IK) to control a **skeleton** that attaches to the **muscles** and **skin**, which attach to clothing featuring cloth dynamics simulations—so

things with character animation are about as complex as they can get without using code, which as you now know is called *procedural animation*. Fusion can bridge over to these features using **FBX**.

At the lowest level of character animation you have your **bone**, which uses an **inverse kinematics algorithm**. This controls the range of movement (rotation) so you don't have elbows that bend the wrong way or heads that spin around like in *The Exorcist*.

Bones are connected together in a hierarchy that forms a **skeleton**. This is what you will animate (keyframe) later on to bring a character to life, and is commonly termed *rigging* in the Character Animation. You can also simulate **muscles** and **skin** by attaching these to each adjacent bone and defining how a bone movement should flex each muscle and stretch the skin for the character. The muscle flexing and skin deformations are complex algorithms as well, just like the **IK** algorithm that controls a range of movement for the skeleton's component hierarchy.

There is also an area called *cloth dynamics* that then controls how clothing worn by the character would realistically move, or deform. This is also controlled by algorithms guided by **control settings**, which are applied by artisans who set up the **cloth dynamics rigging**.

As you can imagine, setting all of this up can be a very long and complex process, and is an area of character animation summarized using the general term *rigging*.

Fusion 8 doesn't have many character animation features, but does use the Alembic, Collada, and FBX formats to be able to seamlessly connect with character animation software, such as Autodesk 3DS Max and Maya, Lightwave, and Cinema 4D XL.

As I have mentioned, character animation and rigging is its very own job area if you plan to work on rigging for the film, VFX, or console game industry. The area of vector 3D is complex, so each of the areas we are covering is its own specialization.

Procedural Animation: Physics, Fluid, Hair, and Particle Systems

The most complex type of animation is procedural animation, because it needs to be done 100 percent using algorithms; all control specifications done by the artist are actually done in the code for the algorithm. In 3D packages this is done using C++, Python, Lua, or Java; Fusion 8 currently uses Lua and Python. The procedural 3D animation for Android applications would be done using a combination of Java and JavaFX, and HTML5 would use JavaScript.

This is the most complex and also the most powerful type of 3D animation, and it is the reason why "procedural animation programmer" is currently one of the more popular 3D job openings in the 3D film, games, VFX, IoT, and television production industries.

You are beginning to see why i3D is the most complex new media genre, along with VFX design using Fusion 8, and sound design using Reason 8.3.

There are a lot of features in 3D modeling and animation packages such as Blender, A:M, Silo, or 3D Studio Max that were actually procedural animation algorithms at one time. These are eventually recoded via the user interface that allows artisans to specify control settings so that all users, not just coders, can use them. These become what is popularly termed a *plug-in*.

The plug-ins expose a user interface to the user in which to specify **parameters**. These parameters control the result of a procedural animation once it is applied to 3D models or to a complex 3D model hierarchy created using the 3D software. This same paradigm also applies to VFX procedural algorithms, which eventually are made into plug-ins and sometimes into hard-coded VFX features that actually appear on the software menu structure as part of the program itself.

Examples of procedural animation algorithm–simulated and –controlled features—many of which include real-world physics simulation support—that are often plug-ins added to advanced 3D animation software packages include 3D particle systems, fluid dynamics, cloth dynamics, rope dynamics, hair and fur dynamics, soft body dynamics, rigid body dynamics, video motion tracking, and many other advanced features that require algorithms in order to be implemented with a reasonable amount of effort. 3D as well as VFX production is all about leveraging the computer processor and memory to create compelling content!

Summary

In this sixth chapter, we took a look at vector 3D concepts and terminology. We looked at components of 3D mesh geometry and how adding texture mapping makes this a complete 3D model. We looked at UVW mapping and how to easily apply this using projection mapping, at what a shader and shader language is, and at different types of 3D animation, from simple linear animation to complex animation types such as character animation and procedural or algorithmic animation. We saw how much complexity this area of 3D new media has due to its ongoing support of advanced console games, VFX pipelines, and feature film and television production.

In Chapter 7, we'll start to learn about **Fusion 8 features and user interface** and how to create a VFX project's foundation, which we'll be building upon during the rest of the book.

Chapter 7

Professional VFX Software: Blackmagic Design Fusion

Now that you have an understanding of the fundamental concepts, terms, and principles of new media content design and assets, it is time to get an overview of what a leading, free VFX software package can do for your VFX content production business. We'll be looking at features of and the user interface for Fusion within this chapter so as to get a high-level overview of the core areas we will be looking at in the remaining chapters of this book.

We'll first look at how Fusion Studio, which costs $995, differs from Fusion's free version, since we won't be covering the Studio features during this fundamentals title. We'll take a look at the Fusion 8 user interface after that, and we'll then take a look at those features that are in the free Fusion 8 version, most of which I am going to attempt to cover during this book.

Fusion vs. Fusion Studio: Two Versions

Blackmagic Design Fusion used to cost thousands of dollars for its professional version, which many of you may remember as Eyeon Fusion. The Pro version of this software, called **Fusion Studio**, is now less than one thousand dollars ($995), which is a steal! Once you see the features that are in this software, which we will be outlining here for you, you will see this software is easily worth at least **ten times** that amount. I commend

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_7

Blackmagic Design for also releasing a **free** version of Fusion (which allows me to write this book, as users can now obtain the software), and for only taking the optical flow (time-warping), network rendering, studio management, stereoscopic 3D, Avid Connect, and third-party plug-in support out of their Pro version, leaving essentially the entire VFX production pipeline toolset **fully intact**. This allows me to write a book covering all of these key VFX fundamentals while allowing readers to fully experience them with full-blown VFX production software that they can use on real-world projects!

Fusion Studio: Flow, Stereo 3D, Nodes, and Plug-ins

First, let's take a look at the Studio features found in the $995 version of Fusion Studio 8, as that is what we will **not** be covering during this book—so let's get that out of the way first.

If you want to see an exact feature-by-feature breakdown of the differences between Fusion 8 and Fusion Studio 8, visit the following URL:

`https://www.blackmagicdesign.com/products/fusion/compare`

I will summarize what these Studio version areas provide in this section of the chapter, then we will look at the Fusion features that we will be covering during this book after that.

Optical Flow: Algorithmic Time-Remapping with Motion Vectors

Fusion Studio's most advanced feature is based on **optical flow** algorithms that were expensive to develop (and probably to patent) and that are well worth the $995 purchase price alone. Optical flow will allow advanced motion image (pixel by pixel) analysis or advanced time-remapping, allowing everything from steady-cam simulation to fixing color variation and simulating film grain.

Those *Matrix* fans out there will want to purchase Studio just to get the **time-remapping** features so that they can play content faster, slow content down, ramp speeds up or down, and completely control the rate of speed throughout a VFX project. You can use optical flow to achieve a fluid result on extremely slow motion shots. You can control the amount of interpolation on frame blending as well, giving you even finer-tuned control.

Optical flow allows Fusion Studio 8 to automatically repair problems with source footage using advanced optical flow image analysis. You can do cool things like remove scratches, remove dust, smooth color variations, and adjust grain.

Fusion Studio uses their advanced optical flow algorithm to generate **motion vectors** so you can fix shots, retime, steady imagery, and morph imagery with amazingly professional accuracy.

Stereoscopic 3D: Produce Stereo 3D Film for Theatrical Release

Stereoscopic 3D film production is another niche area that this Studio version supports but that is not included in the free version. This is fine with me, because I do "real" 3D content production using 3D Studio Max, and the free version of Fusion 8 supports 3D animation and effects content production in their VFX pipeline. Stereo 3D is a faux-3D simulation of 3D using two camera lenses. However, you cannot walk behind a character in a Stereoscopic 3D production, while in a "real" 3D content production you can, because it's really created entirely in a 3D virtual world using 3D software.

Fusion Studio provides you with a complete, stereoscopic 3D pipeline solution for conversion of content to stereo format for use in stereo film theaters. You will "rotoscope" different elements for 3D displacement, add a 3D camera to your scene, and then adjust for different eye separation factors. You can even adjust convergence distance for amazing 2D-to-3D conversions.

Fusion Studio includes specific tools to solve intricate issues that result from live action stereoscopic 3D production. The Studio version of Fusion allows you to repair polarization, adjust color, selectively defocus, and adjust timing differences between two camera lenses (which represent the viewer's eyes).

Network Rendering: VFX Using Massively Parallel Processing

The Studio version of Fusion allows **unlimited network rendering** power to be assembled by adding multi-core workstations to your 3D and VFX **render farm**.

Fusion Studio claims to have the very fastest production-quality render engine available anywhere in the world. A studio can distribute their rendering tasks across an unlimited number of networked computers, allowing their artisans to achieve some amazing performance gains as well as **real-time effect previews**.

What this means is that for $995 a studio would be able to render complex scenes rapidly, allowing artists to meet their production deadlines. With a Studio version you can place the rendering engines on as many workstations or servers as you like and use them all together for a parallel processing scheme that allows you to render your VFX pipeline even faster than using a free Fusion version on an 8-, 12-, or 16-core machine.

Third-Party Plug-ins: Expand Your VFX Toolset and Algorithms

The free version of Fusion 8 includes literally hundreds of built-in tools and algorithms that can be combined to create amazing VFX projects. Fusion Studio allows you to add in even more tools and algorithms using third-party plug-in filters. This enables you to create an even larger number of visual effects using plug-in collections from companies such as Boris FX, which specializes in VFX plug-in filter collections.

Studio Management Tools: Generation Multi-User Management

Studio Fusion 8 also includes **studio management tools** that allow collaboration with larger teams of artists and producers using the **Generation** multi-user management software package. If you're an individual learning VFX fundamentals, which is what a majority of this book's readers are, you won't need these tools!

Generation is a collaboration tool that helps you manage and track versions of every shot in your visual effects project pipeline. In many VFX projects, each shot or scene is an entire sub-project that has more than one artisan working on it.

Generation allows you to assign tasks and then track the tasks across your studio artists. You can post notes and track each VFX artisan's progress to help balance the workload, which will allow a team to finish VFX projects under tight deadlines.

Fusion Studio also features a **bin server** to let everyone share and access common assets that are needed on the job. This means artisans don't have to copy or track multiple VFX project pipeline assets between many computers across studio facilities.

Avid Connect: Bridge Fusion Studio 8 to Avid Media Composer

Another feature of Fusion Studio that individual users are not likely to miss is the Avid Connect plug-in that ties Fusion 8 together with the Avid Media Composer system of NLE (non-linear editing) hardware and software. This plug-in allows you to send any clip or stack of clips from your artisan's Avid Media Composer timeline directly to Fusion Studio 8. Your Avid editors can have access to Fusion Studio's powerful visual FX, 3D compositing, and VFX animation tools and algorithms.

Next, let's take a look at the Fusion 8 user interface.

Fusion 8: Visual Effects Software User Interface

Let's fire up Fusion and take a look at how the user interface for the software is laid out, as the UI is very logical, and helpful in the VFX content production work process. Like most new media content production tools, the top of the software has a **title bar** with the program name and current project name. Notice that in Fusion, projects are **composites**, and are called **compositions**, as the software is at its core a **compositing engine**. Under this is a **menu** system, and under that are your docking **toolbars**. You can undock these if you like. At the right there's another dock that holds **control panels** for tool settings, and underneath that there is the project **statistics** area so you can keep an eye on your system resource usage. At the very bottom there is a **time control** area where you can control where you are in the project as far as the fourth dimension is concerned. The primary **work area** is the bottom half of the Fusion user interface, with tabs in the top-right corner that open up **editing panels**. There are two preview areas in the top half of the Fusion user interface; the left is primarily used for your project's visual feedback, and the right is primarily used for the visual editors in which you'll edit technical or mathematical concepts using things such as spline curves, spreadsheets, tracks, charts, or nodes.

These various functional areas are shown in Figure 7-1.

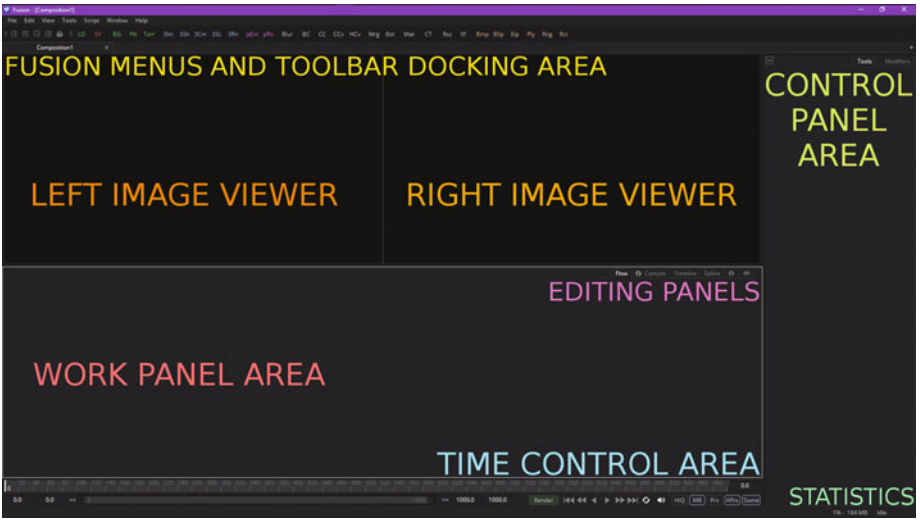

Figure 7-1. *Your primary eight functional areas for Fusion 8*

Next, let's take a look at the features in Fusion 8—the free one!—that I will be able to cover within this book.

Fusion 8: Free Visual Effects Software Features

Let's take a look at the plethora of VFX features that Fusion 8 offers in its free version, just in case I don't have room in the book to cover everything. After all, Fusion has thousands of pages of documentation and is one of the most complex software packages ever designed, and this is a VFX fundamentals title. I am attempting to teach you as much as possible in a few hundred pages, while also teaching you the foundational new media assets information, which we covered in the first half-dozen chapters.

Flow Node Editor: An Innovative Way to Create VFX Pipelines

The Fusion 8 **flow node editor** is central to its visual effects compositing interface. It uses a visual programming approach, where you can "wire together" nodes, to create a **schematic flow chart** structure that is both powerful and fun to create—and to refine. Most competing software packages use a layer-based or SceneGraph approach to creating a composition pipeline. Flow node editor nodes are also called **tiles**, and they represent **images**, **tools**, and **visual effects** that build up your VFX compositing pipeline.

To create VFX compositing pipelines you draw connections between tiles. These links dictate the **order** in which effects processing operations will be applied. By viewing the connections between tiles, you should get a clearer overview of the processing that each image (or video), tool, and applied effect will go through to create the final VFX compositing pipeline for your project.

Using a visual wiring-schematic view, which is sometimes called a "tree" view, affords VFX artists increased flexibility in creating complex VFX compositing pipelines. For instance, an artist can "branch" a tile's output, creating separate branches that are easier to assimilate (understand) visually and that will eliminate any need to group, pre-compose, or nest layers. The Flow Node Editor tab is shown selected in Figure 7-1 over the Editing Panels tab label in the middle-right (in purple).

We will be looking at the flow node editor in Chapter 8.

Timeline Editor: Keyframes, Segments, Guides, Filters, and More

The Fusion **timeline editor** can be used to adjust timing for VFX projects. You can perform NLE functions such as trimming clips, adjusting timing for animation using a spline, or trimming a duration for a tool or effect. You can rearrange the order of tools in a timeline without affecting their layer Z-order in a composite. The timeline editor manages timing for your compositions, while compositing operations are performed with the flow node editor. The **Timeline** Editor tab is shown in Figure 7-1 over the Editing Panels label to the right of the Scripting Console tab.

We'll look at Fusion's timeline editor in Chapter 9.

Spline Editor: Fine-Tune Motion Control Using Advanced Curves

The Fusion spline editor allows motion assets to be controlled, edited, and tweaked using advanced curve mathematics, such as Cubic Bézier and Quadratic Bézier splines, as well as Natural Cubic splines. The spline editor visualizes the change in data values for selected parameters over time using a spline. A keyframe sets a data value for a parameter on any given frame, whereas a spline curve will interpolate a value between two keyframe values. Once a keyframe is set, a spline can be created and you can make precise refinements to an animation. The spline editor can also visualize other VFX pipeline functions that may not be visualized via splines, like changing the characters in a string of text, or mathematical expressions that control animation. The **Spline Editor** tab is shown in Figure 7-1 over the Editing Panels label, to the right of the Timeline Editor tab. We'll be looking at Fusion's spline editor in Chapter 10.

Animated Polyline Masks: An Advanced Pixel Isolation Workflow

When polygon or B-spline masks are added to the flow, they are automatically ready to be animated. All you have to do to animate a mask is move the playhead to a new frame and make the required changes to the mask. A new keyframe is added in both the spline editor and the timeline editor. This one keyframe controls the position of all of the points that comprise the mask for that frame. The polygon or B-spline is automatically morphed between the two keyframes. To adjust the overall timing of the mask animation, you edit the keyframe horizontal-position spline using the spline editor or timeline editor. Additional points can be added to the mask at any point to refine the shape as areas of the image become more detailed.

We will look at animated polyline masking in Chapter 11.

Motion Paths: Precise Spatial Positional Control for VFX Assets

Motion paths are polylines that define the movement for positional controls of layers, effects, and masks. You add motion paths to coordinate controls, such as a merge tool's center or published polyline points on a mask. Coordinate controls are represented onscreen with a crosshair, or an X.

We will be taking a look at motion paths in Chapter 12.

Motion Tracker: Asset Position Control by Tracking Pixel Patterns

Motion tracking is one of the most powerful automation tools available to a compositor. A tracker can follow a pattern of pixels in an image over time, generating a motion path from the pattern's movement. This information is then used to perform a variety of tasks, including image stabilization, matching the movement of one image to another, and driving the positions of points on a mask.

We'll be taking a look at motion tracking in Chapter 13.

True 3D Compositing Environment: Seamlessly Merge 2D and 3D

Traditional image-based compositing is a two-dimensional process. Image layers have only the amount of depth needed to define one as foreground and another as background. This is at odds with the realities of production, since all images are either captured using a live-action camera with freedom in all three dimensions, in a shot that has real depth, or created in a true 3D modeling and rendering application. Within Fusion's flow node editor you have a complete OpenGL-accelerated 3D compositing environment, including support for geometry, point clouds, and 3D particle systems. This makes tasks such as the ones listed next much easier to perform.

Fusion's i3D environment allows you to do simple things, such as mapping 2D imagery on 2D planes in 3D space, or complex things, such as importing synchronized (also called "matched") cameras with 3D "point cloud" vertex data from 3D matchmoving software, such as PF Track or SynthEyes. Because the Fusion VFX editing environment is 3D, you can easily import your 3D scene's cameras, lights, and material library from 3D applications such as Autodesk 3DS Max. You can also cast shadows across imported 3D geometry using your other VFX pipeline assets and elements.

There are 3D features inside of Fusion that allow you to create **basic primitive geometry** (cube, sphere, and cylinder), or you can **import** complex mesh geometry in **FBX** or **Alembic** formats. You can extrude and bevel 2D text inside of Fusion, creating 3D text assets, without using any external 3D software whatsoever.

You can create realistic surfaces inside of Fusion using Fusion-supplied illumination models or Fusion shader compositor pipelines, and then render these 3D VFX right inside of Fusion using realistic depth of field, filmic motion blur, and supersampling.

We'll take a look at compositing using 3D in Chapter 14.

Auxiliary Channels: Bridge 3D Image Characteristics with 2D VFX

As you learned in Chapter 2, images have four channels: red, green, and blue with an alpha channel for transparency. 3D software is capable of exporting other types of data regarding 3D objects in a scene. The data uses **auxiliary channels** that contain **depth** information, mapping coordinates, normals orientation and other data used in VFX assigned to each pixel in a 3D rendered scene. Fusion uses auxiliary-channel data for depth-based compositing, for creating masks based on objects or material IDs, or for texture replacement. Fusion's tools are designed to work with auxiliary-channel data. We will not be looking at auxiliary channels in detail in this book as it involves other expensive software.

3D Renders and Texturing: Shaders, Materials, and Texture Maps

Fusion also has an integrated 3D rendering engine that can deliver photorealistic 3D scenes and special effects using the OpenGL rendering engine and your workstation's powerful nVidia or Radeon GPU hardware. Fusion 8 now supports advanced surface-illumination models in the form of material definition tools as well as a shader definition language. These models use Fusion's visual node-based programming language along with 2D texture maps supplied by the VFX artisan. We will take a look at 3D rendering and shader definition using materials and 2D texture mapping in Chapter 15, and dissect some of the Fusion bin-supplied shaders, so that you can learn how to create your own advanced 3D shaders.

3D Modeling: Advanced 3D Text Models for Dynamic Text Titling

Fusion also has an integrated 3D modeling engine that can create 3D text-titling VFX assets. We'll take a look at 3D text asset creation in Chapter 16, since titling sequences are often a part of the work that VFX artisans are required to do. We'll look at extrusion, beveling, chamfering, and advanced profile 3D modeling using Fusion's advanced spline-editing capabilities.

3D Animation: 3D Animation for Animated Titling Sequences

Fusion also has an integrated 2D and 3D titling engine that can create animated titling effects, so we will continue our work in Chapter 16 and animate the 3D text logos we create using a Fusion 3D text-modeling workflow. We'll take a look at Fusion modifiers, lighting, and animating 3D text assets in Chapter 17, and learn how to add custom imagery as textures to 3D text bevels and faces. We will also look at how to color 3D text assets, or any combination thereof that is needed for your 3D titling VFX.

3D Particle Systems: Introduction to Particle Physics and Dynamics

Fusion also has an integrated 3D particle systems engine that can create effects such as fog, fire, smoke, explosions, and the like, and integrates the particle effect with the 2D and 3D VFX assets. We'll take a look at 3D particle systems tools such as particle emitters, particle effects, and particle rendering in Chapter 18, where we will create our own particle system VFX simulation and integrate it with 2D imagery, creating a digital video asset for our imaginary client.

3D Particle Physics: Advanced Particle Physics and Dynamics

Fusion also has an integrated 3D particle physics engine that can apply physics effects such as gravity, friction, drag, turbulence, vortex, and the like. We'll take a look at 3D particle physics in Chapter 19, adding complexity to our Chapter 18 project so that you can see what a professional-level VFX project that uses all of Fusion 8's powerful tools takes to set up. At the end of Chapter 19, you'll have a Fusion project with twenty nodes that implement dozens of powerful algorithms, effects, and mattes in concert with each other to achieve a cinematic effect that uses a simple still image of the Toronto skyline as a starting point.

Publishing VFX Content: Popular Platforms and Device Hardware

In Chapter 20, we will take a look at where you can go with the Fusion content you learned how to create in Chapters 8 through 19. We will look at digital content publishing platforms and device hardware genres, which few if any VFX artisans are currently publishing their content on. This represents an incredible opportunity for the VFX artisans who ply their trade using this book and can deliver never before experienced VFX on new and emerging digital devices and platforms, such as HTML5 OS iTV set products like Panasonic's Firefox iTV and Android Wear 2 smartwatches.

Summary

In this seventh chapter we took a look at **Fusion Studio** features, the free Fusion user interface, and some of Fusion's features that we'll be looking at during the rest of this book. In Chapter 8, we'll look at Fusion's **flow node editor**, which is used to implement the visual tile-based programming language that Fusion is famous for and that other software packages have emulated**.**

Chapter 8

VFX Pipeline Composition: Using the Flow Node Editor

Now that we have gotten an overview of Fusion, it is time to get into the nuts and bolts of what makes it one of the leading VFX tools available today. We'll be looking at the user interface for the Fusion flow node editor during this chapter, as that is the core way that VFX processing pipelines are built in Fusion.

We will look at how to add tools to the flow node editor and how to connect them to each other to achieve VFX objectives such as color correction. We'll look at the different flow node utilities, such as cut, copy, paste, rename, alignment, grouping, and so on.

Flow Node Editor: VFX Compositing Tool

As you saw in Figure 7-1 in the previous chapter, the flow node editor is the **default tool panel** when you open Fusion or start a new project, as it is central to this VFX software—which is why we are covering it first! For this same reason, we'll also be covering it in most every other chapter in the book, so you will get getting a lot of experience with it, especially for a fundamentals book! If you are in the Timeline or Spline tabs and want to look at the "flow" of the VFX pipeline composition, use your **F5 key** or click on your **Flow tab**, as shown in Figure 8-1.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_8

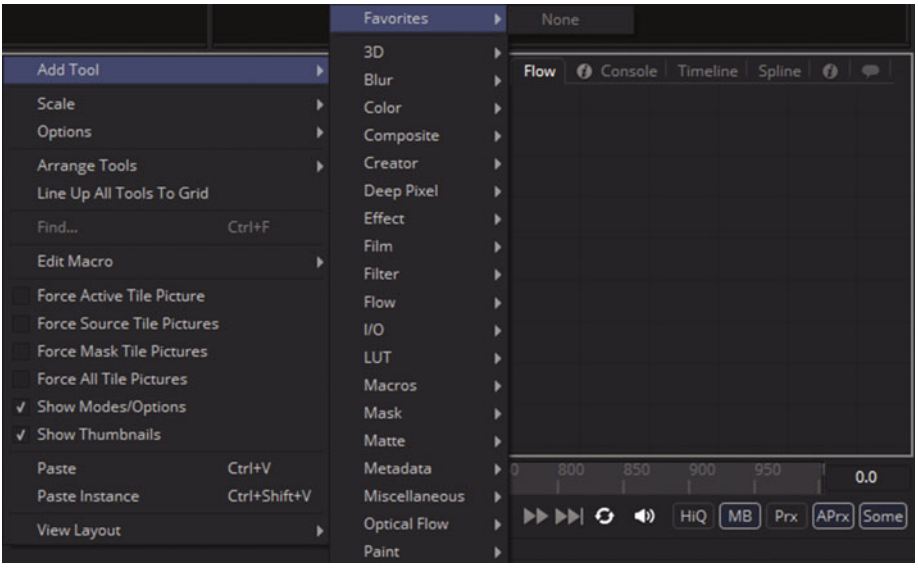

Figure 8-1. *Use a context menu to access flow tools and options*

To add a tool to this flow node editor, you can use your **Tools** menu at the top of Fusion or the **Tool Icons** at the top of Fusion, as was seen in Figure 7-1. I like to use a context menu approach, which is shown in Figure 8-1. Context menus are great for advanced software like VFX and 3D, as you simply right-click on an object or in an area and a menu shows only the functions or commands that will work in that context. Since using context menus also accelerates the workflow, I will be using the context menu approach whenever possible during this book. If you select the **Add Tool** context menu option, a **Tools** sub-menu will appear.

This is the same Tools menu that's at the top of Fusion, and many of these tools are available in the tool icon bar as well.

If you select a Tool sub-menu, another sub-menu will appear with Tools found in that Tool category. You can see this for the Favorites category in Figure 8-1, although initially you'll see **None** in that sub-menu, until you start using the software.

You can also use the context menu to **Scale** your flow node editor view (25%, 50%, 100%, 200%, 400%) or access flow node editor **Options** or **Utility** settings. You can also use this menu to affect a flow node **View Layout**, **Arrange Tools** (or tiles), or **Edit Macros**, as you can see in Figure 8-1. Context menus are key to an effective VFX workflow!

The Fusion Bin: Using Predefined VFX and Tools

You can also add tools, as well as entire pre-defined VFX flows, using the Fusion **Bin**, which as you can see in Figure 8-2 can be accessed using the **File ➤ Bins** menu sequence. You can open bins using the arrow icons on the left. As you can see, I opened the Tools bin as well as the Color Correction Tools bin. To add one of the tools to a flow node composition, simply drag it out of the bin and drop it onto your flow node composition where you want it to be inserted into your VFX compositing pipeline.

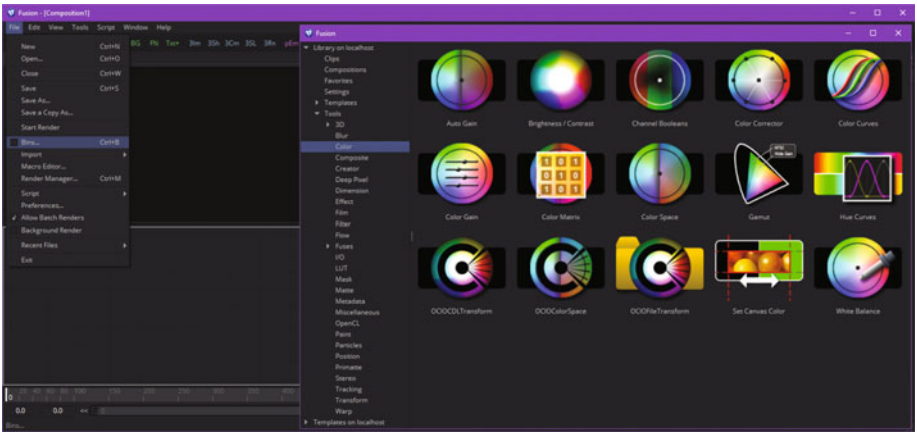

Figure 8-2. *Use a File ➤ Bin menu sequence to open Fusion Bins*

After we add an image to work on in the next section, we will be using the color corrector tool seen in Figure 8-2 (top row, fourth icon) to lighten the shadow areas in an image to bring out more details in a model's hair and hat, and in the background areas of the image.

Take a moment to peruse through these bins to see all of the amazing tools and presets that the free version of Fusion gives you. The Studio version of Fusion allows you to access a centralized repository bin structure across your entire studio.

You can also drag and drop from the OS's file management software. This is a great way to add still images, sequences of images, or digital video assets. Let's take a look at that next.

Adding Imagery: Drag and Drop with File Manager

There are a couple of ways to add 2D assets into the flow node editor. Using the context menu shown in Figure 8-2, you should select **Tools ➤ I/O ➤ Loader**. This will add an image loader tile with which to load and reference an image from the system hard-disk drive. An even easier way to do this is to drag and drop the image you want to load from your operating system file management utility into your flow node editor directly, as is shown in Figure 8-3. Once you drop this into the flow node editor it is shown using a collapsed tile, which you can change using the **File ➤ Preferences ➤ Global Settings ➤ Flow ➤ Tile Pictures** setting. I like to turn tile pictures on, as it gives you visual feedback as to the results that each tile is giving you in your VFX pipeline flow. If you want to see the **image loader (Niki.png) tile** in a figure, take a look at Figure 8-4 in the next section, which shows your image loader wired into the color corrector tool from your bin.

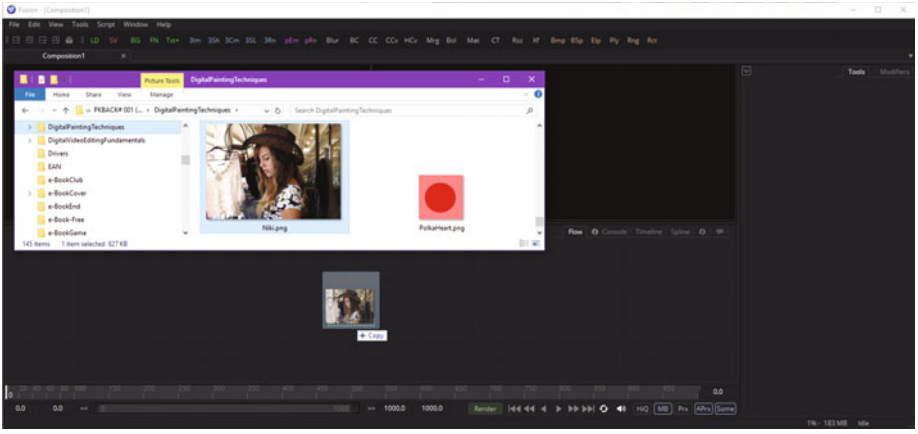

Figure 8-3. *Drag the Niki.png image from file manager into flow*

In case you're wondering how to delete a tool (or tile), simply click on it to select it, then hit your **Delete** key!

The Color Correction Tool: Lighten Shadow Levels

Let's use Fusion's flow node editor to pull some details out of the shadows in the model's hair using the **color corrector** tool. Open the Fusion Bin and drag the tool into the flow node editor area, then drag the red output box from the image loader onto the orange input triangle on your color corrector node, as is shown in Figure 8-4.

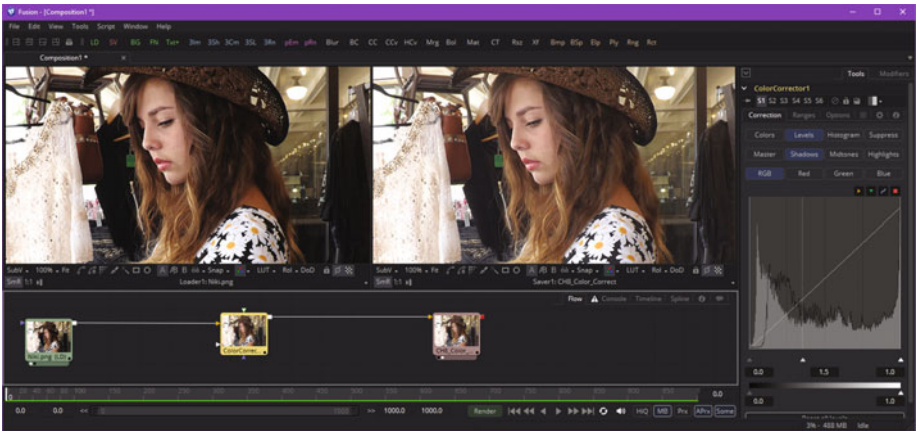

Figure 8-4. Wire image loader to color corrector to image saver

Click on the color corrector node and select that tool; its **Options** control panel dialog will appear in the control panels area at the right side of your software, as you can see in Figure 8-4. Select the **Levels** tab, and under that the **Shadows** tab, and set the midpoint to a value of **1.5** (or drag your arrow indicator at the bottom of the data diagram to suit your shadow-lightening tastes). As you can see in the right view pane, this allows the user to now see more detail in the hat and hair area of this image. **View indicator dots** at the bottom of your loader or saver tiles will indicate which of the two viewing panes are showing pre- (left) and post- (right) image-processing results. You can also click a node (tile) and use the **1** and **2** keys on the keyboard to select which view you want that node to use.

Next, right-click to the right of your flow pipeline, select **Add Tool ➤ I/O ➤ Saver**, and add an **image saver** node tile. Name the output file **CH8_Color_Correct**, and drag the output box from the color corrector node onto the input triangle on your image saver tile, as is also shown in Figure 8-4.

Now that we have a basic image color correction pipeline in place in the flow node editor, let's save your composition.

Saving a VFX Project Pipeline: Using "File ➤ Save As"

To save your VFX compositing pipeline at any time, use the **File ➤ Save As** menu sequence and open a **Save File** dialog, which is seen in Figure 8-5 on the left (menu) and middle-right (dialog). I named the composition **CH8_Composition_Niki_Levels_Shadows**, and Fusion will give that file name a **.comp** Fusion file extension.

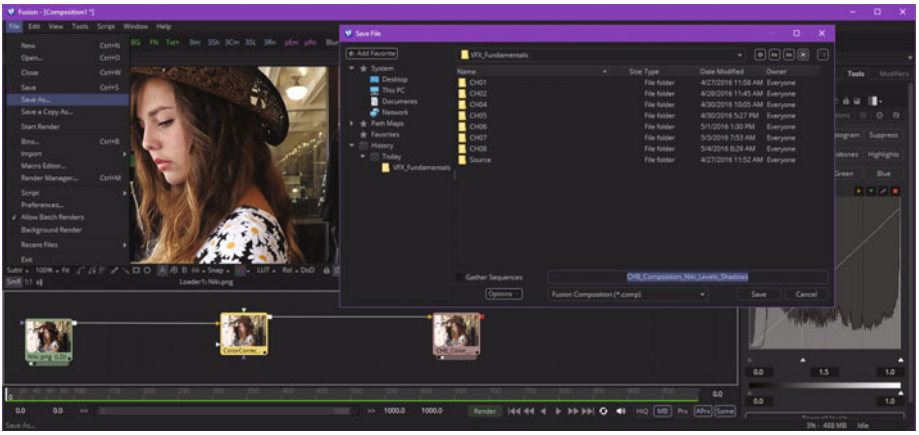

Figure 8-5. A File ➤ Save As menu sequence opens a Save File dialog

Flow Branches: Multiple Flows from a Single Node

You can have more than one branch coming out of a single node. Let's use the image loader to do more than one thing by having two branches come out of the Niki.png source PNG24 image. Right-click at the right of the flow node editor and select the **Add Tool ➤ Warp ➤ Vortex** context menu sequence, shown in Figure 8-6. We can use this effect to add some variation into the hair, allowing you to style the hair after the fact using VFX tools.

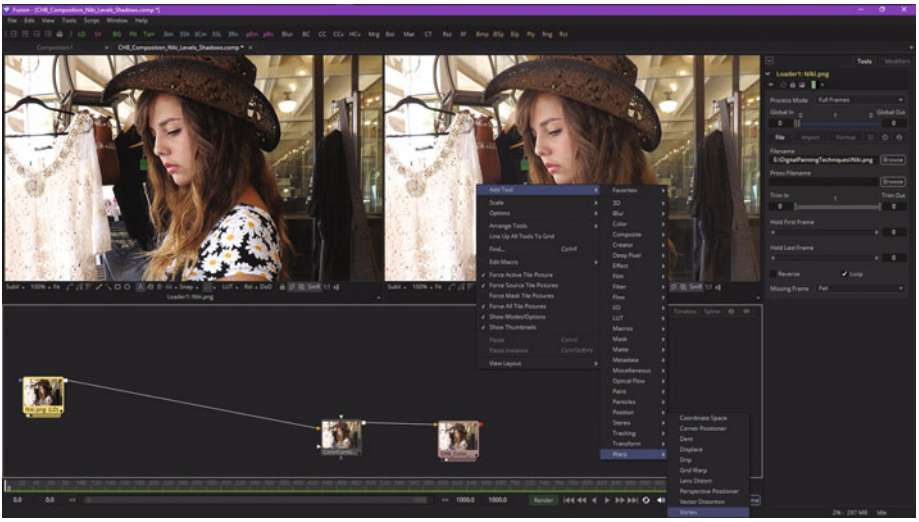

Figure 8-6. Right-click and choose Add Tool ➤ Warp ➤ Vortex algorithm

Drag the vortex warp node so that it is over the color corrector node, as shown in Figure 8-7, and drag a wire—or "pipe," as they are called in Fusion—from the image loader output (red) square to the vortex warp input triangle.

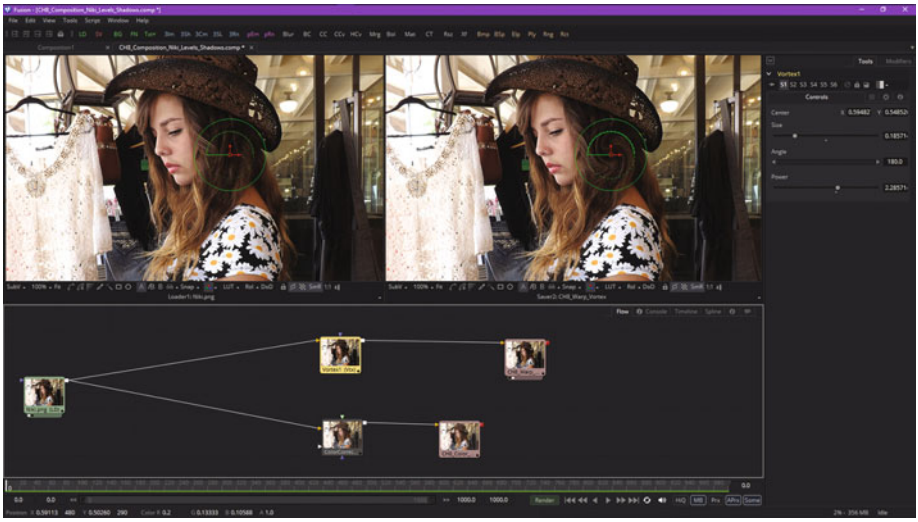

Figure 8-7. *Wire the image loader to vortex warp to image saver*

Right-click and use your **Add Tool ➤ I/O ➤ Image Saver** menu sequence to add an image saver (output) node, then drag the red output to your image saver input, wiring those together as well. As you can see in Figure 8-7, you now have multiple image-processing workflows coming out of the same image loader node.

The VFX tools that come out of the image loader node can be said to be "downstream" from the image loader, and therefore the image loader can be said to be "upstream" from the VFX tool nodes, which are upstream from the image saver nodes. This term is logical if you think about how the VFX processing flow is travelling.

If you want to unwire nodes, you simply mouse-over a pipe, and click on the blue half of the pipe, as is shown in Figure 8-8.

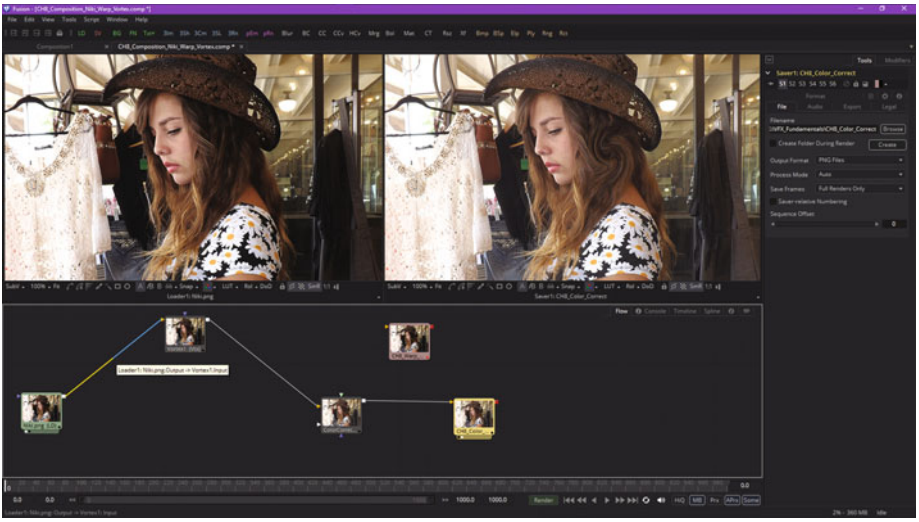

Figure 8-8. Rewire the vortex before your color corrector node

As you can see in Figure 8-8, I have inserted the vortex warp in the same VFX processing pipeline workflow as your color corrector so you can better visualize the vortex effect on the hair. Also shown is the mouse-over on a pipe to remove it from the node. Select the extra image saver and press the **Delete** key to remove that flow node tile from the VFX pipeline.

To see which pipes are connected to a given node, simply place your mouse over that node. To remove a pipe from both the input and the output in the same drag operation, hold the **SHIFT** key down while you drag to disconnect the data pipe.

You can also insert a tool between two nodes by dragging it from the Bin onto the pipe that connects those nodes, or by selecting an upstream node (a node prior to where you want it) and then using the **Add Tools** context menu or primary **Tools** menu.

You can also rename the nodes by right-clicking on them and selecting the **Rename** function. You can align all nodes with a **Line Up All Tools to Grid** function from the context menu, as seen in Figure 8-1. There is also an **Arrange Tools ➤ To Grid** option, which when selected turns on a “snap to grid” feature.

Grouping Flow Nodes: Organizing a VFX Pipeline

Next, let's add another unsharp mask branch to show you a cool trick regarding how to sharpen only the green channel of an image, and then look at the Group feature. Drag the **Blur ➤ Unsharp Mask** tool out of the Bin or right-click and use the **Add Tools ➤ Blur ➤ Unsharp Mask** context menu to add a VFX sharpen node. Now, wire it to the image loader, as shown in Figure 8-9. Also add an image saver and wire the unsharp mask tool into that as well. Select the **green** channel checkbox to get a more subtle (professional) sharpening result, as the contrast in a digital image is primarily contained in the green color channel.

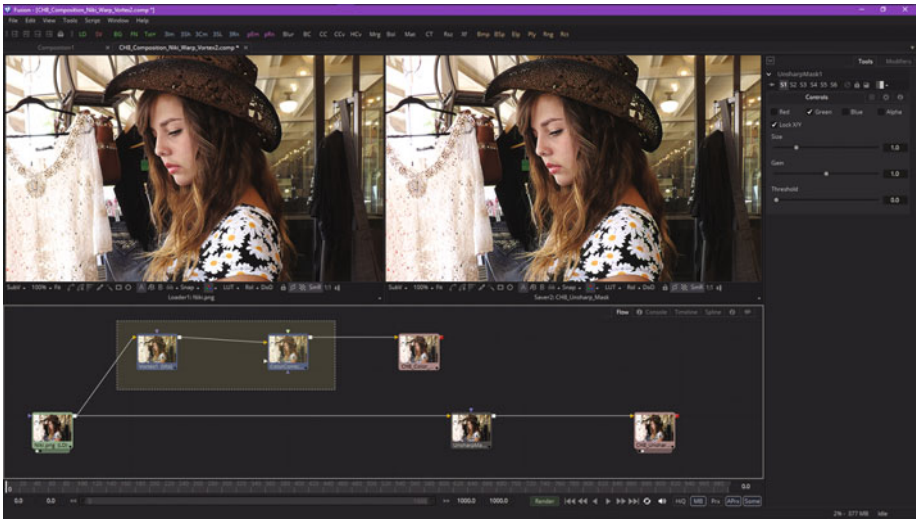

Figure 8-9. *Add an unsharp mask tool, and sharpen only the green color channel*

Using all (R, G, B, A) channels in this operation makes the sharpening effect too pronounced. Notice I have also **dragged** my **marquee** around the two nodes I want to **group** in Figure 8-9, to save on the number of figures I'm using in the chapter. To invoke the Group command, right-click on one of the selected nodes and select **Group** from the context-sensitive menu options, as shown in the middle of Figure 8-10.

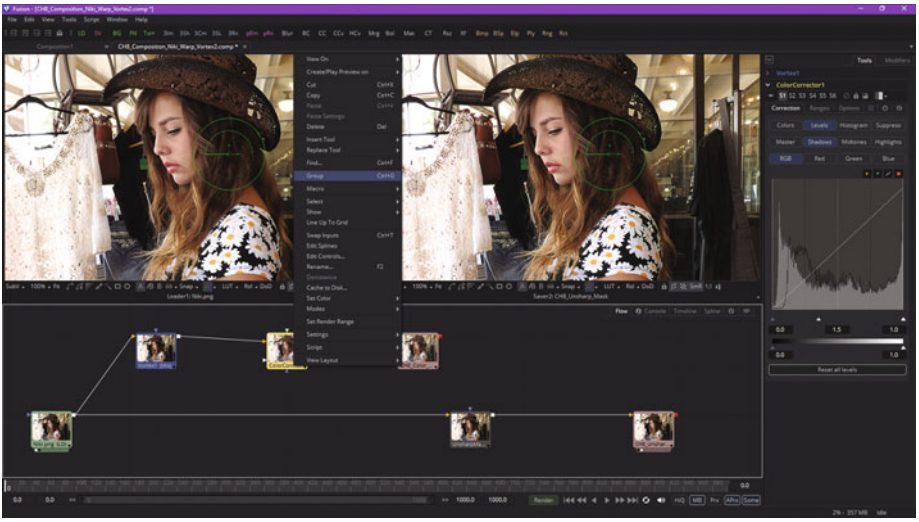

Figure 8-10. Right-click on one member of your group, and select Group in context menu

As you can see in Figure 8-11, a warp and lighten shadow VFX pipeline is now one tile named **Group1**, with the group symbol at the bottom-right. Right-click the tile, and **Rename** the group **WarpLevels**, so that you know which tools are grouped inside it.

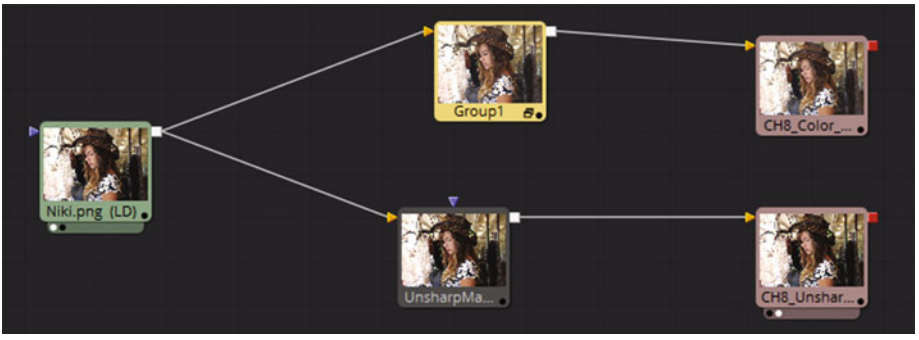

Figure 8-11. Vortex and color corrector are now a VFX Group

You can also use your **CTRL-G** keyboard shortcut to group selected tiles together using that work process, if you wish. I also clicked on the **group icon** in the WarpLevels group tile, to show you how to **expand** a group, which is shown in Figure 8-12.

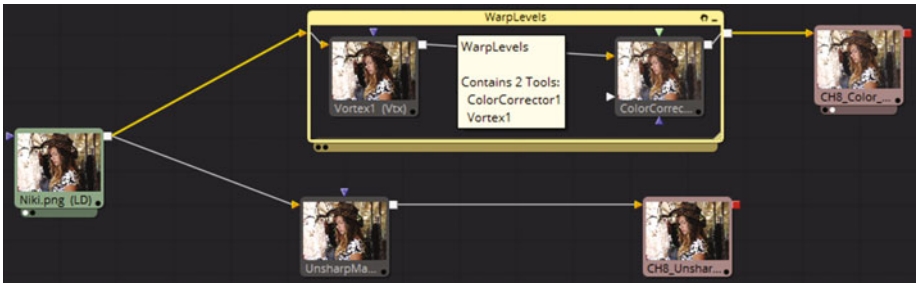

Figure 8-12. Click WarpLevels node group icon, and expand the view

As you can see, if you mouse-over the WarpLevels group's title bar, you will also get the group information popup helper as well as a viewport selector tray at the bottom. You can also save your WarpLevels VFX pipeline tool group as its own setting by right-clicking the title bar and selecting **Settings ➤ Save As** from the context-sensitive menu as is shown in Figure 8-13. You can also use this in other VFX projects. As you can see, Group can be used to design your own custom Fusion 8 VFX tools!

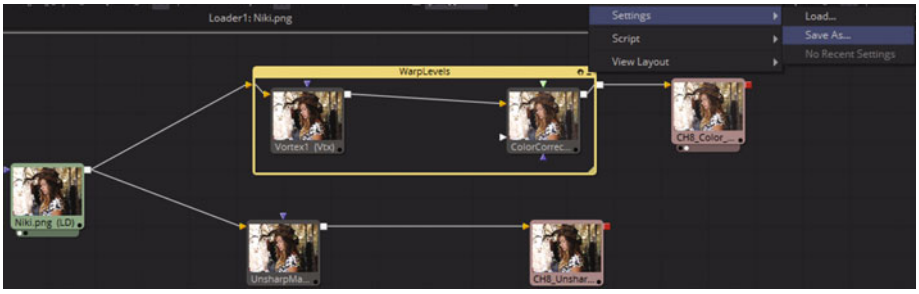

Figure 8-13. Right-click in WarpLevels title bar and Save setting

You will get more experience with flow node editor soon!

Summary

In this eighth chapter, we took a look at the Fusion flow node editor, and how to use its core feature set in real-world usage. In Chapter 9, you'll look at Fusion's **timeline editor**.

Chapter 9

VFX Pipeline Animation: Using the Timeline Editor

Now that you have a basic understanding of the flow node editor, which we will also be using in this chapter so you get some practice, let's switch gears from 2D compositing space to the 4D time dimension. We will take a look at the **timeline editor** in Fusion, as well as the closely related **time ruler** area, which is located at the bottom of the Fusion user interface. We will cover all of the 4D aspects of Fusion in this chapter and then learn how to expand our control even further in the next chapter, which will cover drawing motion curves using the Fusion 8 spline editor.

For now, we will look at how to animate digital still imagery, as this is the best way to allow you to learn the timeline editor features without having full-blown video assets.

Timeline Editor: Creating Animated VFX

Let's create a project from scratch to see how the flow node editor and timeline editor work together. Start Fusion 8 and then right-click, selecting **Add Tool ➤ I/O ➤ Image Loader**. Select the **Niki.png** image and then right-click in the flow node editor again. Select **Add Tool ➤ Blur ➤ Blur**. Rename the node (tile) **NikiBlur** and wire to it from the image loader tile. Set the **Blur Size** to **100%** and the **Blur Blend** to **1.0** (100%) for the Blur node tile. You are going to animate the model into clarity from a fog, so we are going to start with these maximum blur and blend settings.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_9

Notice in Figure 9-1 that you do not have to insert your image saver node in order to start working with the time ruler or the timeline editor portions of your Fusion 8 VFX software.

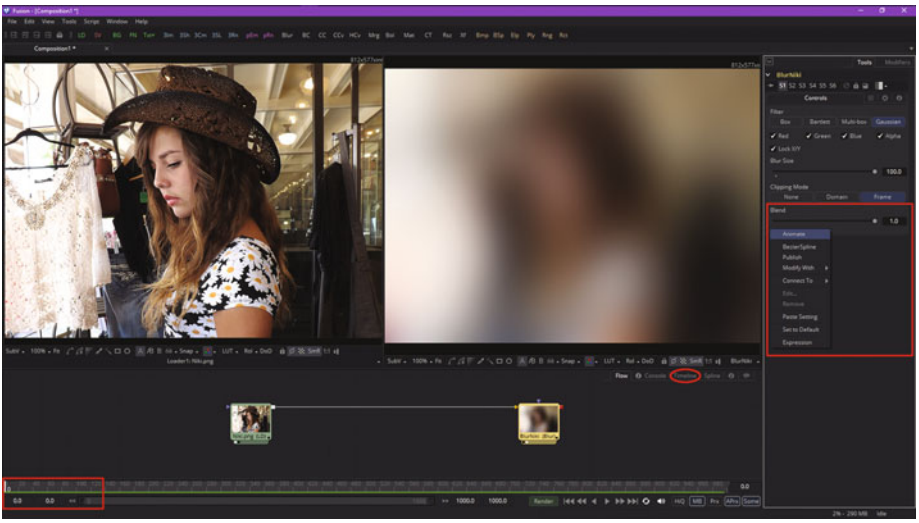

Figure 9-1. *Right-click the Blur Blend node and enable the Animate function*

As you can see in the time ruler, found in the bottom left part of Figure 9-1, you are at keyframe zero, so these maximum blur settings will be set as your starting values. To start keyframe operations, right-click the **Blend** setting of 100% (1.0 is 100%) and select the **Animate** option from the context-sensitive menu.

To enter the timeline editor, click the Timeline tab seen circled in red in Figure 9-1. As you can see in Figure 9-2, you have timeline "tracks" for both of these nodes ready to go! Move the **playhead** vertical time-position bar to **30**, by dragging it with your mouse to the right, as can be seen in Figure 9-2.

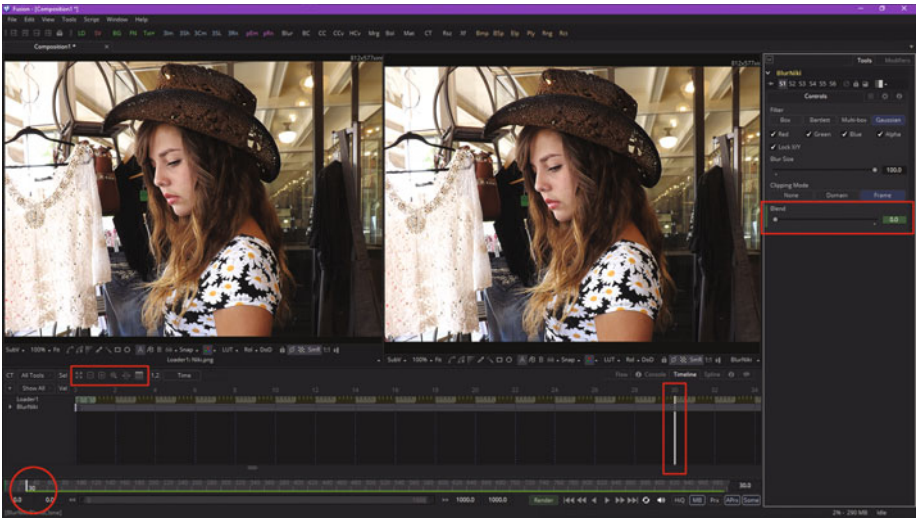

Figure 9-2. Move playhead to 30, and set the Blend value back to 0.0

Also notice that the time ruler shows your current frame position at 30 with a white tick, seen circled in red in Figure 9-2. I have also highlighted the **timeline utility icons** for **fit**, **zoom out**, **zoom in**, **zoom selection** (marquee), time **stretch**, and the show/hide **spreadsheet** mode toggle. These icons are located at the top-left of your timeline editor pane.

Next, drag the **Blend** slider back to **zero**, or type **0.0** in your number field, at the right of your user interface control. Notice that this automatically **sets a keyframe** for the value at frame 30; you do not have to right-click and select Animate, as the animation feature was enabled by doing this the first time for this node. By doing this, your blur will animate the next time that you drag the playhead back toward frame 0.

Fusion highlights the control and numeric entry field in green when a keyframed range has been put in place. If you drag the playhead vertical bar back toward zero, you will observe a change in the numeric Blend settings and in the blur preview as well. Also seen in Figure 9-3 are the keyframe left for the node at frame 30 and the “reset to Frame Zero” shuttle control that allows you to go immediately back to the start of the animation timeline. It also shows how to set a **working range** for a timeline animation, which I have set to **120** frames, or 4 seconds. I circled in red the **set range end** field and working range green tickmark on the time ruler, to highlight where they are in Fusion.

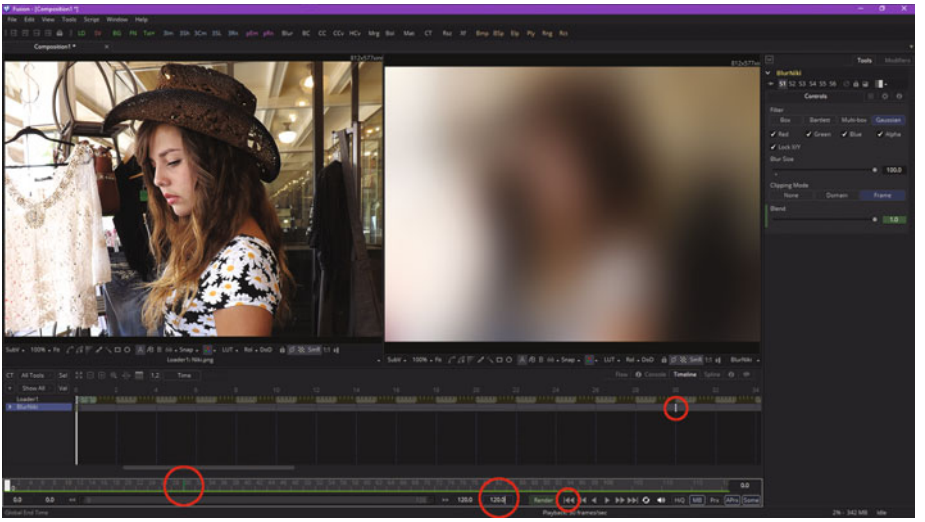

Figure 9-3. Drag the playhead back toward zero to animate the blur

I also show how you select a node in the timeline editor using the **timeline header**, which is the area to the left of the timeline editor. When selected, a node will turn a **blue** color.

Next, let's take a look at how to animate multiple nodes in more complex compositing pipelines, and also at some more timeline editor features.

Animating Multiple Nodes: Asynchronous Motion

Let's add another node with a **drip** effect after the blur effect, as you want the blur to be processed first. Right-click and select **Add Tool ➤ Warp ➤ Drip** and wire a pipe from the blur node output square to the drip node input triangle, as is shown in Figure 9-4, which shows only your flow node editor project area. Notice that the drip node tile preview shows the output of your blur node tile at keyframe zero, since that's where the playhead is.

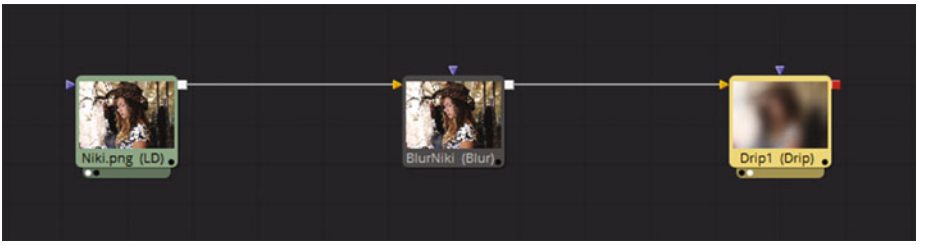

Figure 9-4. Add a drip effect node, and pipe the blur node into it

Click the **Timeline** tab to switch over into your timeline editor. Click the **minus (-) icon** to zoom out, as seen in Figure 9-5. Notice that you can have **negative** time values, and that if you click and drag using your middle mouse button, you can pan the timeline itself to any time value range, even a range prior to frame 0! We will see why this is important in the chapter covering spline (curves) editing, as you may want some settings for the effects algorithms to start numeric processing prior to frame 0 in order to achieve a smoother interpolation result.

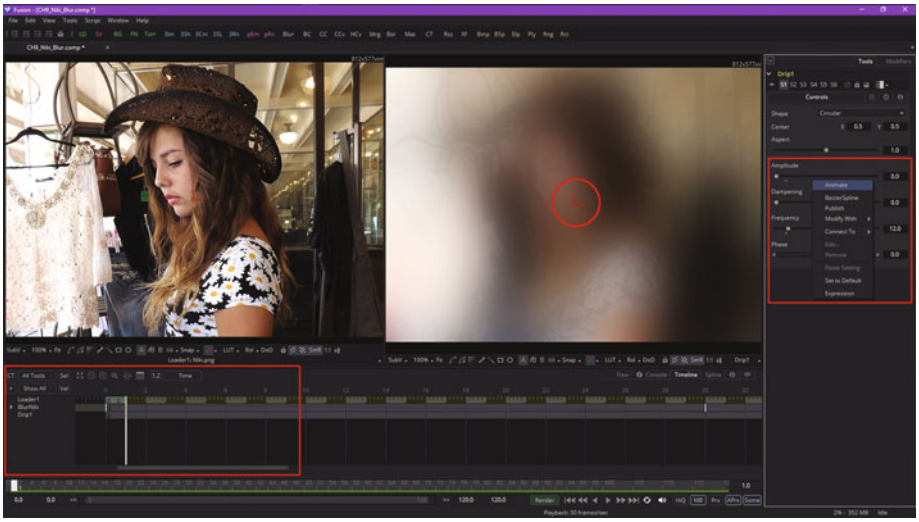

Figure 9-5. *Drag playhead to Frame 1 and right-click Amplitude, then select the Animate option*

Position the playhead at frame 1 and leave the drip tool settings at the default (no effect) setting. Right-click on the **Amplitude** slider and select **Animate** to enable the keyframing capability. You can use the **drip center widget**, seen circled in red in Figure 9-5, to position the center of a drip.

If you want to animate your drip center widget, you simply move it to another location when you set the destination keyframe, and Fusion will move it for you over time using the X and Y values, shown at the top-right in the Control Panel area in Figure 9-5. The only way to get a feel for what the settings do for Fusion visual effects is to experiment with their settings!

Now drag the playback head to frame **32** and type **0.16** in the Amplitude data field, as seen in Figure 9-6. As you can see in the preview area, this gives you a professional drip effect. Your amplitude keyframes are now visible in a **Spreadsheet view**.

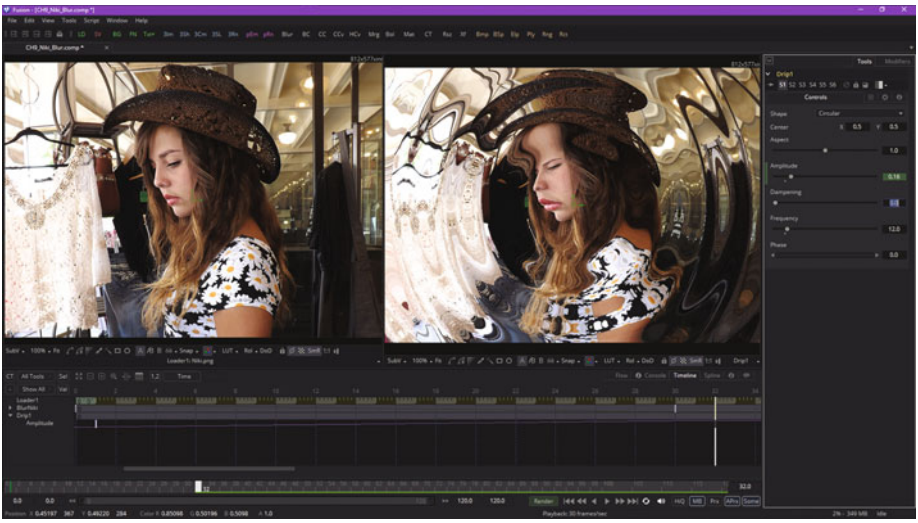

Figure 9-6. Drag playhead to frame 32, and set Amplitude to 0.16

Next, zoom out some more using the minus (-) icon, then drag the playhead to frame **60** and set the Amplitude back to **0.0** to remove the drip effect. Now, when you **drag** your playhead from 0 to 60, as you can see in Figure 9-7, the blur and drip VFX are combined, using the flow node editor (algorithmic) to define 2D compositing, and in 4D (time) using your timeline. As you can see, the number value field in the control will also animate, so you can see the value used on every single frame. I also opened up the spreadsheet views for both nodes, shown in a timeline header area on the far left. The timeline playhead position will also show in your time ruler, indicated by a **white block**, and a number at the bottom of the Fusion software. These, along with the keyframe you set at frame 60, are all shown in red.

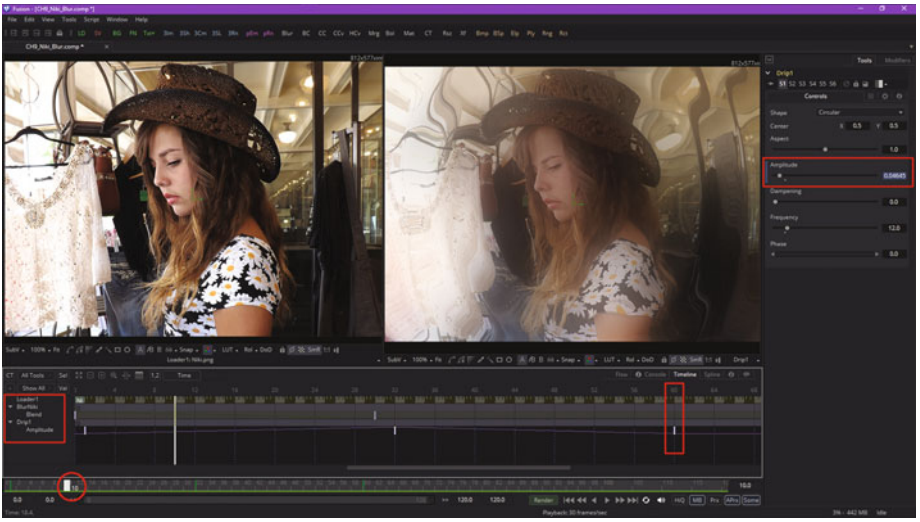

Figure 9-7. Set an ending keyframe at frame 60, and then preview by dragging playhead

Finally, notice in the spreadsheet keyframe (and curves) view, that there are linear (straight-line) **motion curves**, which control a **rate of change** for the motion between your keyframes.

Motion Paths: Keyframing 2D Spatial Animation

You can also keyframe elements moving in 2D space on a screen, in addition to visual effects timing. When you do this, you get a **motion path**, a spline path on your composite, that allows you to control the movement of the elements on one of the layers—or, in the case of Fusion, one of the nodes—in your VFX compositions. Add an image loader, and go into the **Open File** dialog, as shown in Figure 9-8.

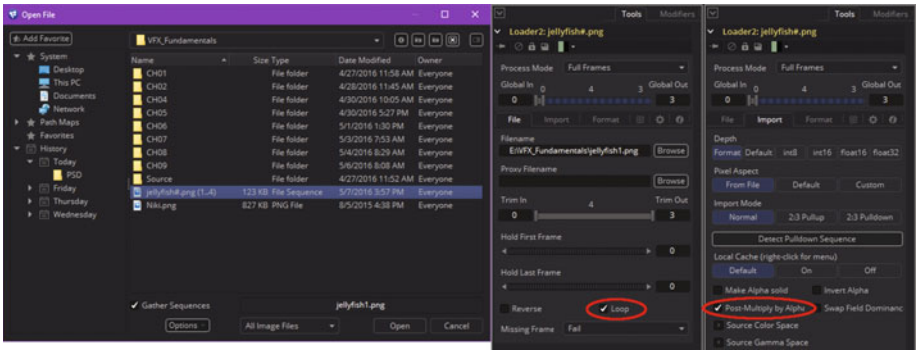

Figure 9-8. Load image loader node, and select a PNG32 image sequence

Find the PNG32 four-image jellyfish animation series, so we can learn how to work with animated image elements in Fusion. Select the image series, delineated automatically by Fusion as **filename(number...series)**, by selecting the series in blue, and clicking on the **Open** button. Set the Loader2 Control to **Loop** in the **File** tab, and to **Post-Multiply by Alpha** in your **Import** tab. The result can be seen in Figure 9-9 in the far right (green) node.

To composite one image (or series) with another, right-click in the flow node editor and choose **Add Tool ➤ Composite ➤ Merge**, which is shown selected in yellow in Figure 9-9.

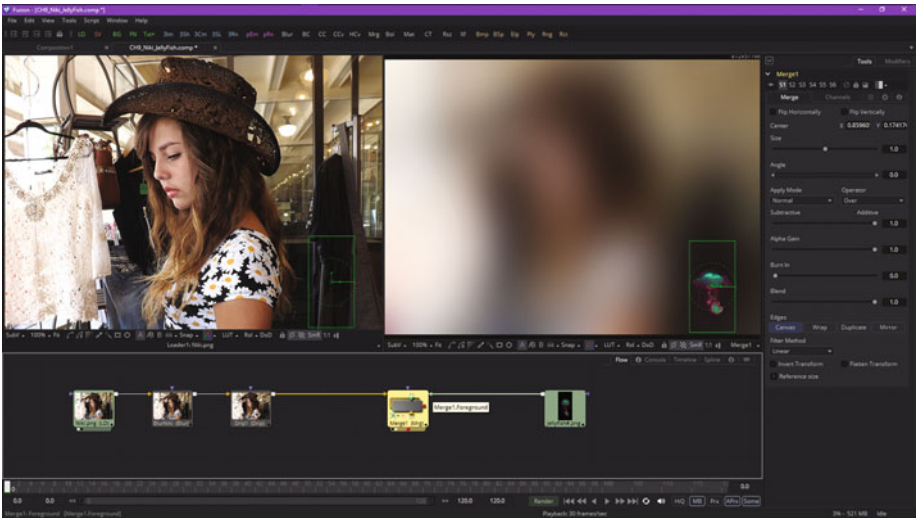

Figure 9-9. *Add a merge composite node, and connect Niki.png to the background input, and the jellyfish sequence to the foreground input*

Next, wire the drip output into the background input for the merge, and then wire the jellyfish image loader series into your merge node foreground input, as seen in Figure 9-9. You can put the mouse over any input, and Fusion will tell you what that particular input does.

You can drag your jellyfish using the center control, as seen in Figure 9-9, or position it using the X and Y data entry controls, as I have done in Figure 9-10, by setting Y to zero. The jellyfish is now seen emerging onto the composite, and due to how we have set up the flow node editor, it will swim on top of the blur-plus-drip composite effect that we created earlier.

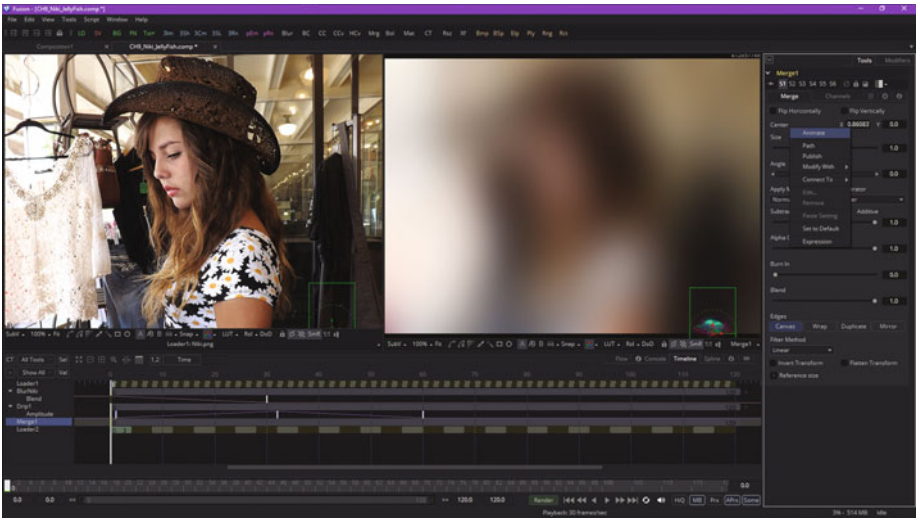

Figure 9-10. *Set Jellyfish Y=0, right-click Center, and Animate*

Also seen in Figure 9-10 is the fact that the merge node is selected (because it was selected in your flow node editor), the fact that the playhead bar is set at frame 0, and the fact that I right-clicked on the **Center** control and enabled it to **Animate**.

Now we are ready to create the **motion path** by animating the jellyfish. This involves setting a target keyframe. I chose frame 90, as you can see in Figure 9-11. Then set the Y location for the center of the jellyfish to 1 (or 100%, or top).

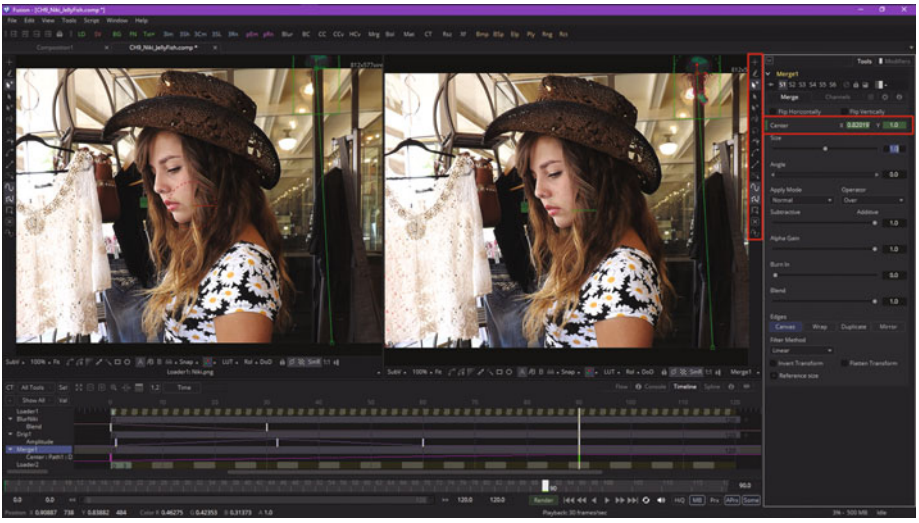

Figure 9-11. Set keyframe to 90 and set the Y location for the jellyfish to 1

As you can see in Figure 9-11, Fusion now shows you your motion path, as well as a new toolbar to use to edit the motion path, which is shown highlighted in red on the right-top of the right view pane. We will be covering these spline editing tools in Chapter 10, when we cover your Fusion spline editor. Also seen in red is the fact that the Center Y control data field is highlighted in green (animated).

Now, when you drag the playhead position bar to the right, you will see your jellyfish swimming upward, and the VFX on the Niki model image will play as designed behind it. You have a perfect compositing pipeline that is now animated using still imagery on frame 18 of the animated composite, as shown in Figure 9-12.

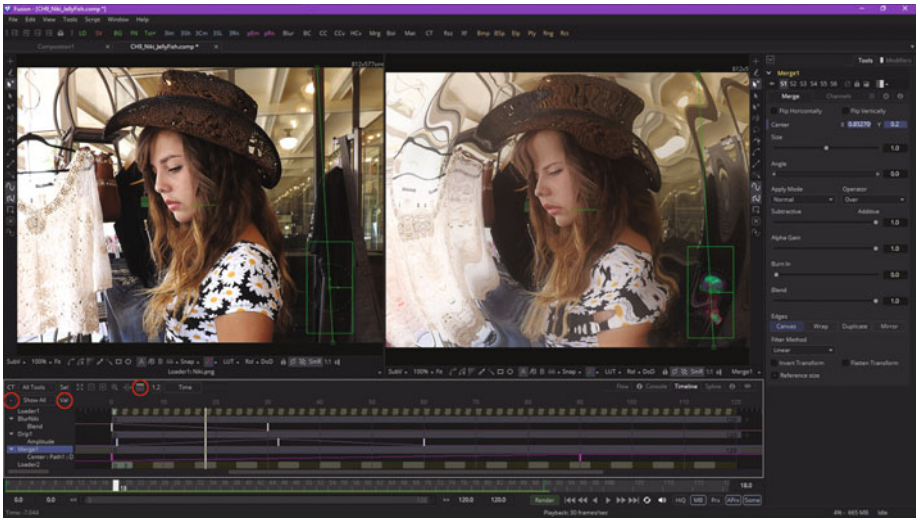

Figure 9-12. Test project by dragging the playhead to the right

Also highlighted in red in Figure 9-12 are your **collapse** (-) and **expand** (+) icons on the far left, your **spreadsheet** show (or hide) option at the top, and the **VAL** (show data value) icon on the right. Play with these so that you become familiar with all the timeline editor tools, options, and settings. Figure 9-13 shows the result of using the VAL icon toggle, which lets you see the numeric data values that you entered in the Control Panel area.

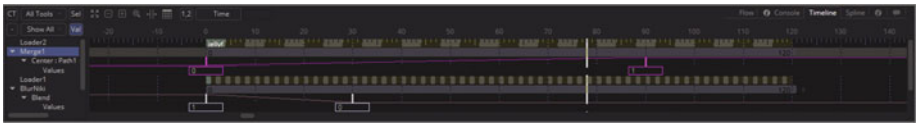

Figure 9-13. Use a VAL UI button to toggle the data values to show

We will look at the **time stretch** tool (icon) in the next chapter on spline editing. Figure 9-14 shows how I use the time **zoom marquee** tool to allow my project timeline keyframes to fill my timeline editor viewport.

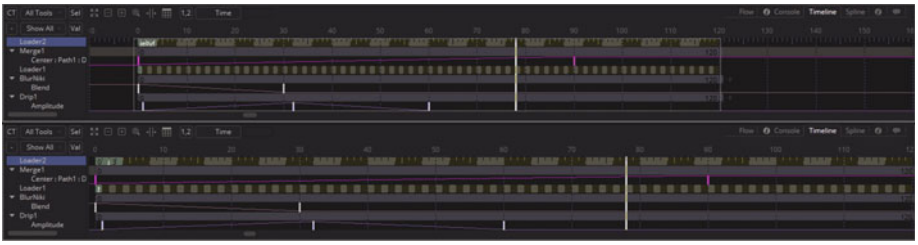

Figure 9-14. Using the zoom marquee tool (top) to fill viewport

Summary

In this ninth chapter, we took a look at the Fusion 8 timeline editor. We looked at how to enable animation (keyframing) for a flow node control, how to position your playhead to show Fusion where you want a keyframe, how to switch back and forth between the flow node editor and timeline editor for your workflow, how to import a 32-bit alpha channel compositing image series for use in animation, how to create motion path animation in a flow node editor preview, and how to practice using the various tools in the time ruler utility at the bottom of the Fusion UI as well as in your timeline editor, where you can work in the fourth dimension!

In Chapter 10, you will learn about the Fusion **spline editor** and how to algorithmically control the change in the rate of movement over time using complex mathematical curves, such as cubic and quadratic Bézier curves or natural B-splines.

Chapter 10

VFX Pipeline Motion Control: Using the Spline Editor

Now that you have an understanding of the Fusion flow node and timeline editors and how they work together in a VFX workflow, it is time to get into how you fine-tune the motion over time using complex spline constructs to control the "rate of change" in time and using curves to define **time interpolation mathematics**.

We'll continue working on the project created during the previous chapter in the **spline editor**, changing the complexity of a linear (no changes over time) movement into a more complex second-order derivative (change in rate over time) mathematical representation. We will accomplish this by using **spline curves**, a brilliant application of one mathematical topology to another complex endeavor (controlling time), in my most humble opinion.

Spline Editor: Control Time Using Curves

The Fusion **spline editor** allows you to edit the motion curves that you created in the timeline editor when you defined ranges of movement by turning on **Animate** and setting keyframes. To see these, click the **Spline** tab next to the Timeline tab, as shown in Figure 10-1 on the far right. To see a spline, select a node or node parameter that you keyframed; you'll see motion curve data. If you check a parameter, you can edit your spline; if you put a dot in this checkbox, you will be shown a **reference spline**. If you

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_10

leave this checkbox empty, Fusion will hide that spline. A reference spline is a spline that is visible but locked from editing.

Figure 10-1. Click the Spline tab, then select the Amplitude control

I'm also showing the **Show Key Markers** icon/toggle option turned on in Figure 10-1, and I highlighted the resulting keyframe triangles, shown in red. I will try to cover as many spline edit features during this chapter as I can, starting with the UI (user interface) tools and widgets. Since no nodes, or **spline handles**, are selected, a significant number of tool icons—at the top of the spline editor—remain dim, which means they are not active.

Navigate the Spline Editor: Independent Zooming

The spline editor allows you to zoom independently on an axis, as you can see in Figure 10-2, where I have zoomed in to the y-axis (vertical). The y-axis represents the magnitude of a change over time, whereas the x-axis (horizontal) represents the time duration itself. Compare Figure 10-2 with Figure 10-1.

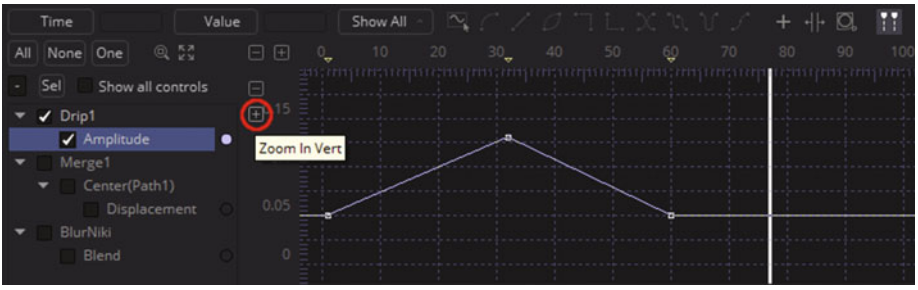

Figure 10-2. Click y-axis plus to zoom in on the rate of change

Interestingly, zooming in vertically is the same as zooming out horizontally, which is shown in Figure 10-3, where the representation of the spline curve becomes even more peaked, even though I have not edited the spline itself. Zooming out on the time serves to zoom in on the rate of change (keyframe values), as you can see.

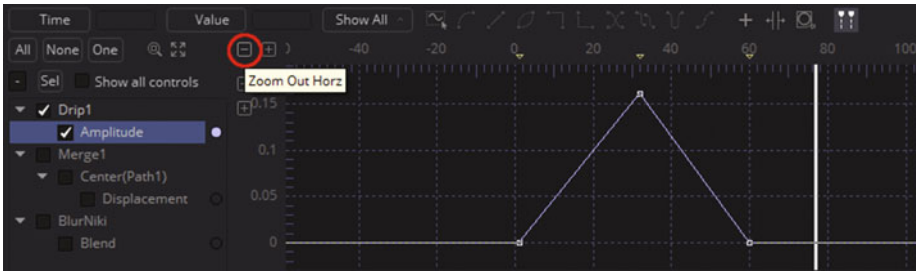

Figure 10-3. Click x-axis minus to zoom out on the rate of time

You can either click on these zoom icons, shown in Figure 10-4, or click and drag your mouse over the numbers in the X and Y margins, which label the values used in your graph.

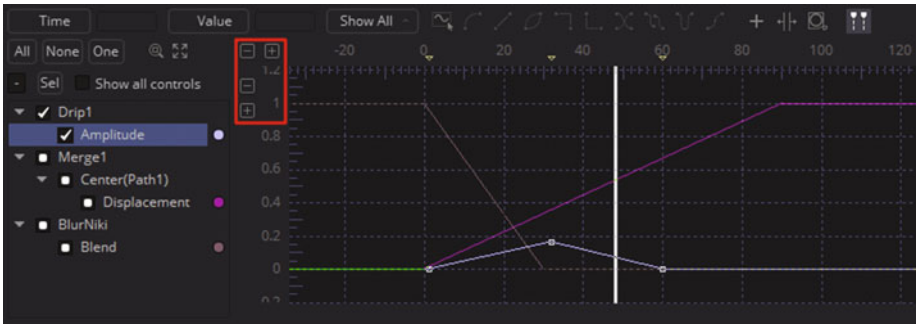

Figure 10-4. Zoom to fit splines, turn on all curve visibility

As you can see in Figure 10-4, there are colored dots in the spline header that color the splines (bright color when the spline is selected, darker color when not active). Click on one now; you will get the **Spline Color Picker** dialog.

Coloring the Spline Editor: Customize Spline Color

The Spline Color Picker dialog allows you to set any of 16,777,216 color values for your spline curve by selecting **basic color** swatches on the left or setting **HSV** or **RGB** numeric values on the right. You can also click an empty **custom colors** swatch, seen on the left in Figure 10-5, to save any non-standard color you create.

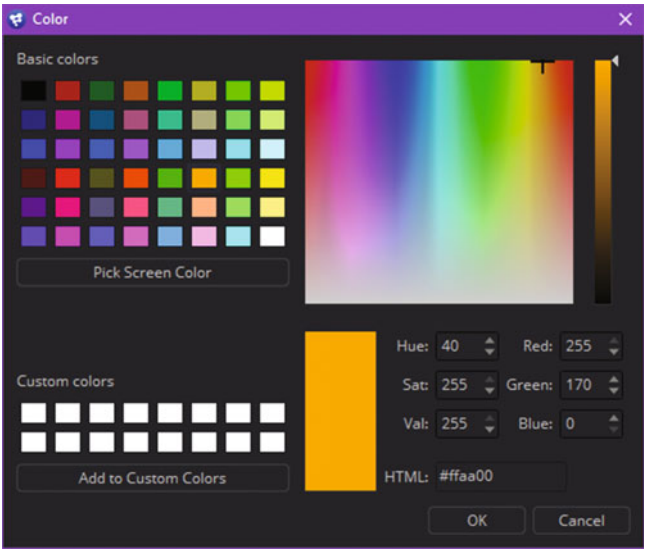

Figure 10-5. A Spline Color Picker dialog

Viewing Splines: Pan, Show, and Hide Splines

You can drag splines around in the viewport in the same way that you drag things in other areas of Fusion, by using the middle mouse button to click and drag the view. You can also use the All, None, and One buttons for views that have lots of splines showing to make the workflow easier to comprehend. The **All** button turns on all splines for editing, as is shown in Figure 10-6.

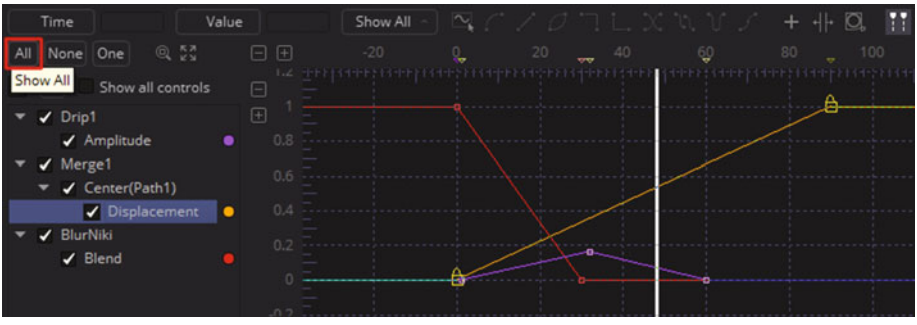

Figure 10-6. The All button turns on all splines for editing

The **One** button turns on one spline at a time for editing. If you select another spline for editing, a previously selected spline will be turned off. This allows you to work with splines in isolation without referencing other splines in the project, as shown in Figure 10-7.

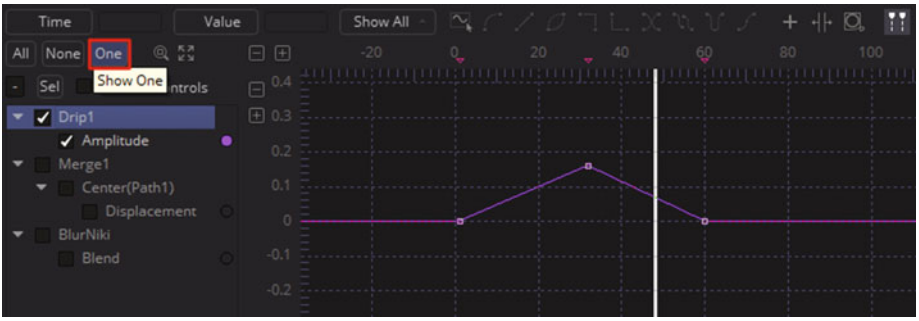

Figure 10-7. *The One button toggles individual spline display*

The **None** button turns off all splines for editing. Why? Think of this as a **reset** to use before you select the One button. The **Sel** button turns off all splines that are not selected by the blue highlighting. This button does not affect how checkbox settings have been set by yourself, whereas the top three icons do affect those checkbox settings—All checks them all, None will uncheck them all, and One allows only one (selected) to be checked at any given time. The optimal way to get familiar with these is to use them in a VFX project. The **Show all controls** button, when clicked, will show all options in all nodes whether they're keyframed or not. This is not frequently used for VFX work, as you generally only want to see what you are actually working on in any given editor (the "less is more" theory, as it were).

If you do click the Show all controls button, use it with your Sel button, as shown in Figure 10-8, so that you're only displaying all the controls for one of the nodes you are working on.

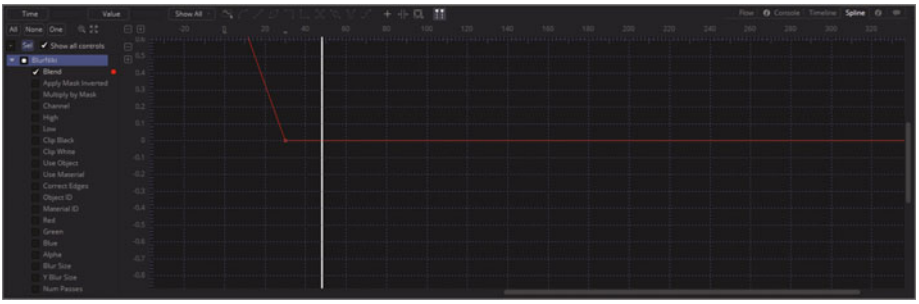

Figure 10-8. *Use the Show all controls button with the Sel button*

Next, let's take a look at how to change the spline data.

Controlling Spline Curvature: Tensioning Handles

Select the middle keyframe (top point) in your drip node, which brings the ripple effect in and out of the VFX composition. As you can see in Figure 10-9, you will get several new UI elements, including two spline tensioning handles on either side of the selected point.

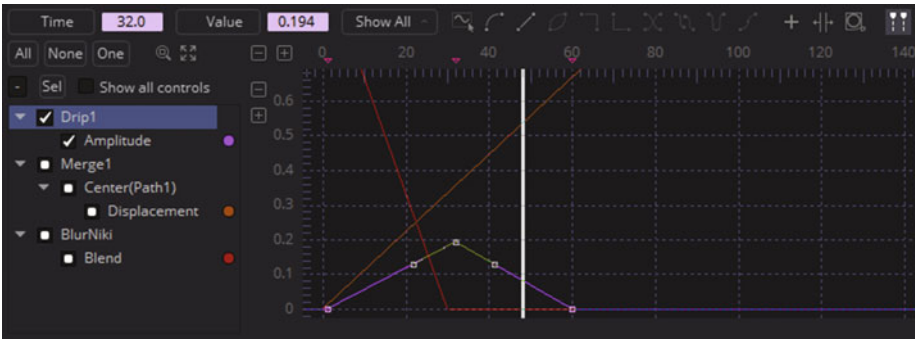

Figure 10-9. Select the middle (top point) keyframe in the spline, revealing the spline editing handles

As you can see, once you select a spline node or control point, you get the keyframe (Time) value and its corresponding data value, shown in light purple. You also get two icons that allow you to turn that segment of the spline into a polygon, straight line, or curved line. These are shown highlighted in red in Figure 10-10. I have also pulled one of the **tensioning handles** away from the spline to create a **control curve** on that half of the spline. A spline is controlled by the **handle length** as well as the **angle** of the handle, so play around with this to get a feel for how it will control your spline curvature.

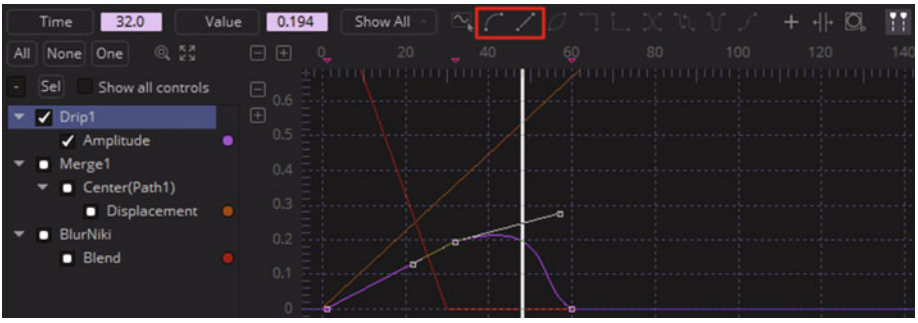

Figure 10-10. Pull right spline handle and create curved motion

If you now pull the other (left) tensioning handle away from the spline, you will notice that this affects the movement of the right tensioning handle to some extent. To move handles independently of each another, hold down the **CTRL** key (Command on Mac) on your keyboard, as shown in Figure 10-11. This will tell Fusion 8 that you want to "break" this connection between the spline handles.

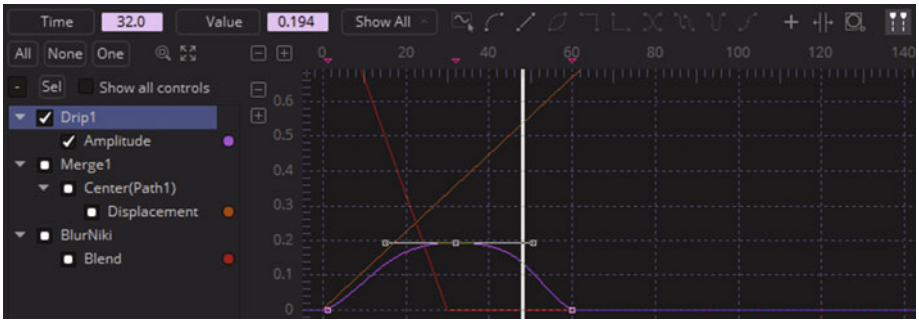

Figure 10-11. *Hold down CTRL key and move spline tensioning handles independently of each other*

To refine a spline curve further, you can insert **control points**. When you mouse-over any segment of a spline, it will turn blue. Click anywhere on that segment, and insert a new control point, as is shown in Figure 10-12. Pull the tensioning handles and create a complex motion control curve for your VFX parameter.

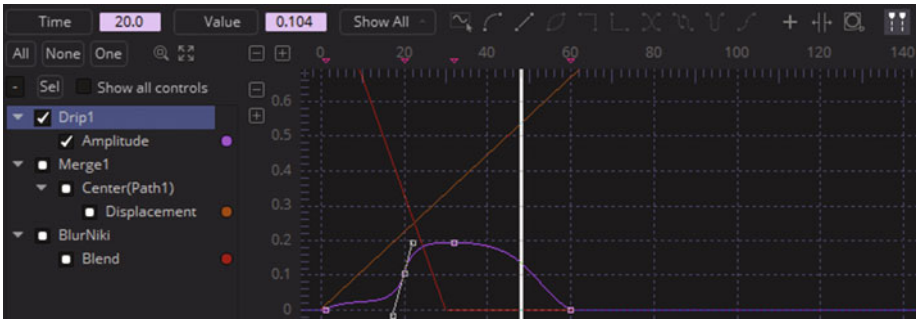

Figure 10-12. *Mouse over left half of curve and click to insert a control point*

There are other ways to affect spline curvature, such as using the context-sensitive menu approach that we used in Chapters 8 and 9. Select the merge node and right-click your first keyframe spline node, as seen in Figure 10-13, and select the **Smooth** option, which will turn the curve from the polygonal spline representation to a smooth spline curve representation.

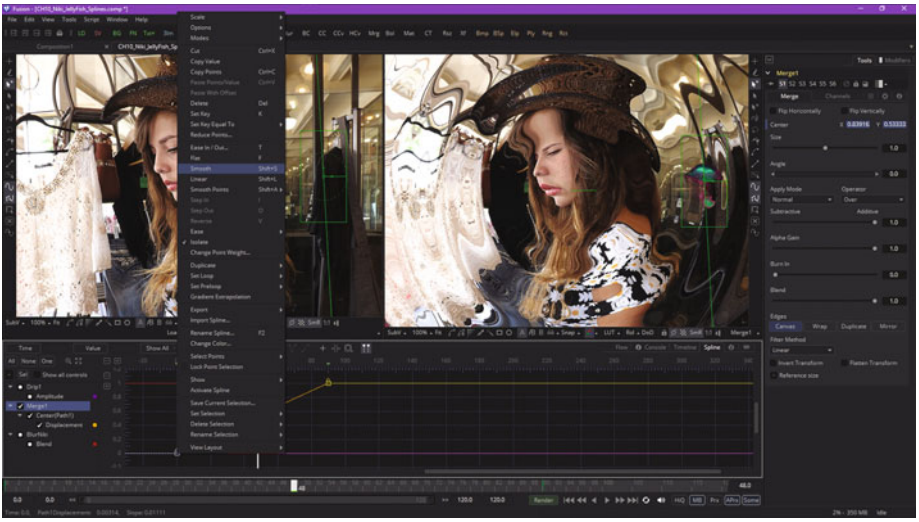

Figure 10-13. Right-click first merge control point and then choose Smooth

You can see your smoothed Merge1 motion control curve in Figure 10-14, which now ramps the merge much more smoothly over time. This gives a more subtle but also a more believable result.

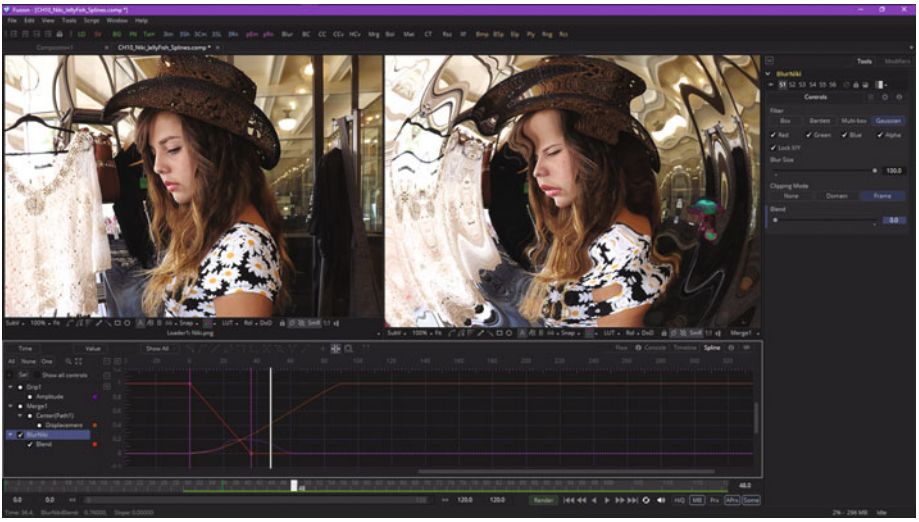

Figure 10-14. Marquee select the blur curve control points and use the time stretch tool to insert purple time-stretch handles

You can also transform several spline control points at the same time using a **marquee selection**, found with the other spline tools (dimmed unless multiple control points are selected) at the top of your Fusion 8 spline editor pane.

Spline Scaling Tools: Time Stretch and Shape Box

Select the **BlurNiki** node in the spline header area and marquee select two control points in this curve. Click the **time stretch** tool, shown circled in red in Figure 10-14, at the top-right of the spline editor. This will add two vertical purple bars that function similarly to the timeline editor playhead. You can drag either one and smoothly scale the time values for that segment. This essentially "stretches" time, hence the name for the tool.

There is another tool called the **shape box** tool that can do the same thing as the time stretch tool, but in both dimensions at the same time! Let's look at your **zoom marquee** tool and use it to zoom in on this project. The tool is seen in Figure 10-15 circled in red; click to activate it, and then draw a marquee around the **Drip1** node spline curve, which will put a white dotted line around the area that you want to fill your spline editor pane.

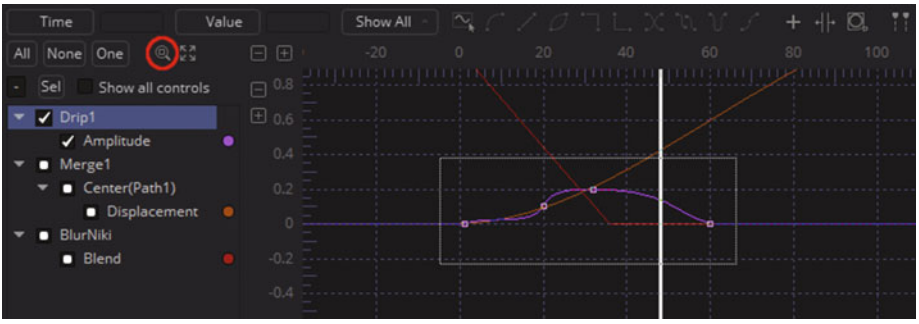

Figure 10-15. Use a zoom marquee tool to zoom into a drip curve

Once the marquee zoom is done, as shown in the top panel of Figure 10-16, this tool will turn off and you can again use your marquee select to select the last three keyframes (control points in the spline editor) on your Drip1 node. Once these are selected, click on the **shape box** tool, shown in the middle panel of Figure 10-16 (above the yellow popup tooltip). A **control cage**, complete with **resizing handles**, should appear around your selected Drip1 node's (keyframes) spline control points.

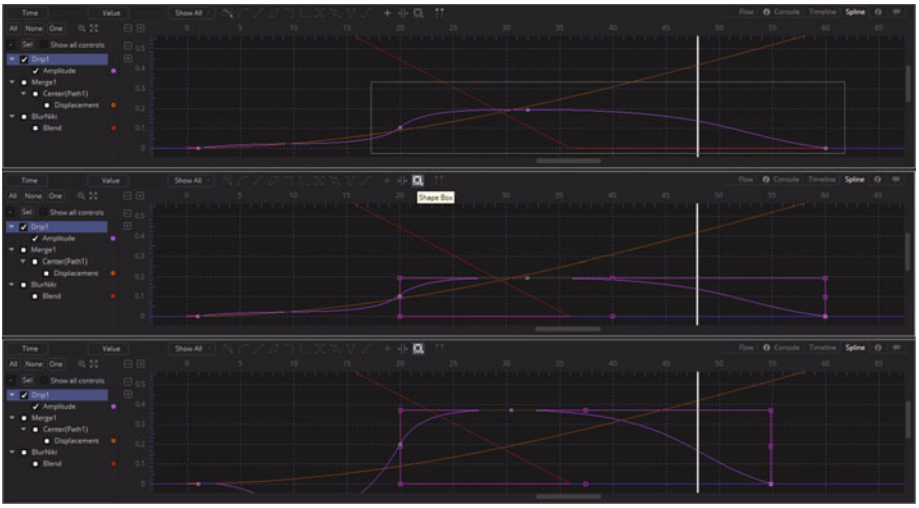

Figure 10-16. Select the shape box tool to scale time in both the X and the Y dimensions

I used the right-side cage handle to scale the curve in toward the left, and the top cage handle to scale the curve up.

Control Point Macros: Spline Curve Algorithms

If you select **more than one** control point on your spline curve, you will enable (unghost) the rest of the control point tools, which are essentially algorithms or "macros" that can automate some common spline editor tasks. I selected the middle two drip control points using a selection marquee, as shown in Figure 10-17, and suddenly all of the tool icons that were ghosted at the top of the spline editor became **live**, or activated for use.

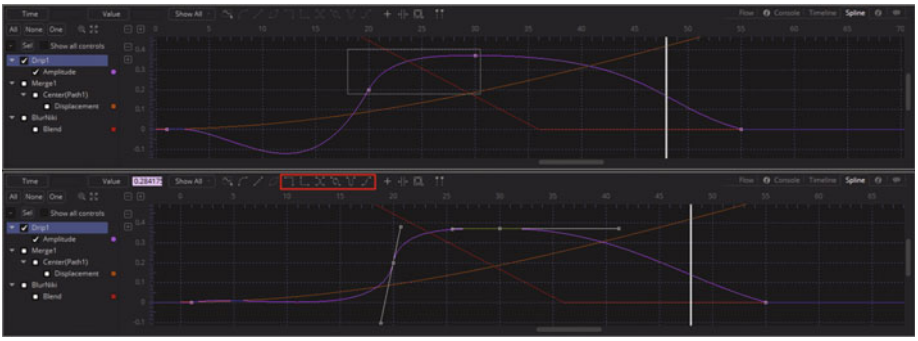

Figure 10-17. Selecting two or more control points will enable more spline algorithm tools at the top of the spline editor

The first two, **step in** and **step out**, create basic square polygon "steps" between the two selected points, as is shown in Figure 10-18. These create jerky, rather than smooth, movement.

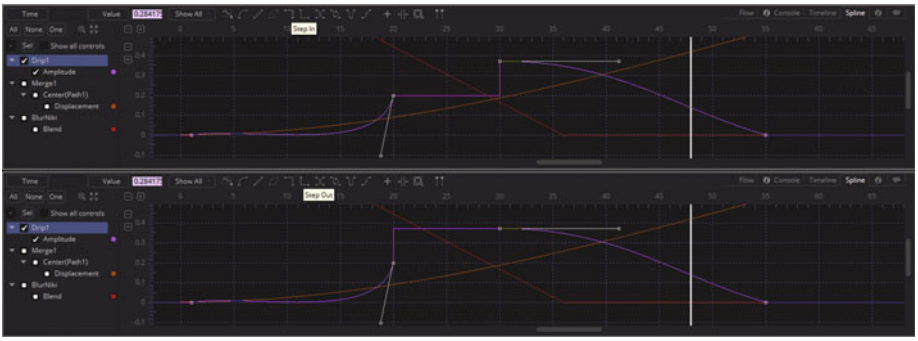

Figure 10-18. *Use step in or step out to create polygonal lines*

The next one, **reverse**, simply reverses your mathematical spline data. As you can see in Figure 10-19, this is not invert spline (mirror), but rather a complete reversal of a spline's direction.

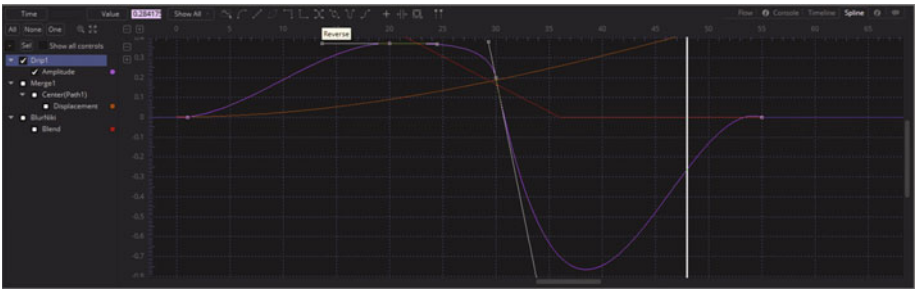

Figure 10-19. *Use reverse tool to reverse mathematical (spline) data*

The last three tools create **copies** of the selected curve region and paste them onto the rest of the spline. Let's take a look at these before I finish up the chapter. As you can see in Figure 10-20, I have selected the last two control points on the Drip1 node, as I want them to repeat *ad infinitum* to create my looping effect, just as if I had selected the **Loop** option in a control panel. Click the **Set Loop** icon to loop the last section of the spline.

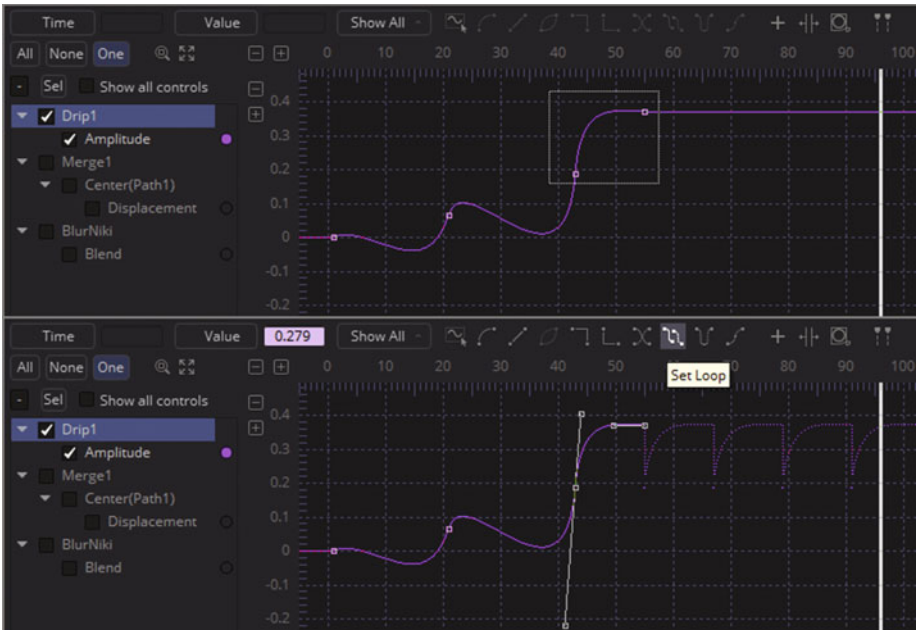

Figure 10-20. Select your last two control points for looping

Click the **Set PingPong** icon to loop the last section and mirror it at the same time, as can be seen in Figure 10-21. I did not see a PingPong option in the Control Panel settings I looked at in Chapter 9, so this would be a way to achieve this effect.

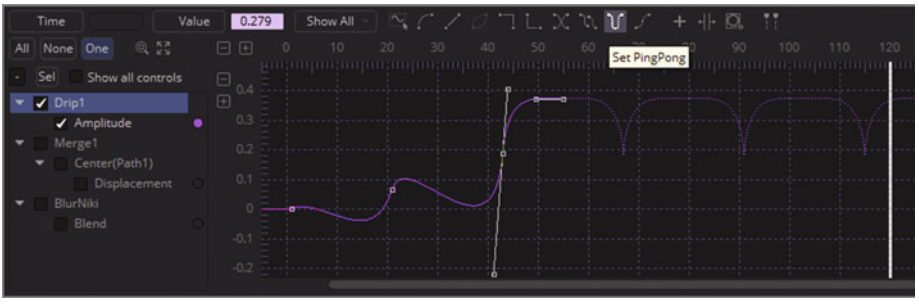

Figure 10-21. Click the Set PingPong icon to loop and mirror curve segment

The **Set Relative** icon loops the spline segment relative to the last point, creating a staircase-like looping result, as shown in Figure 10-22.

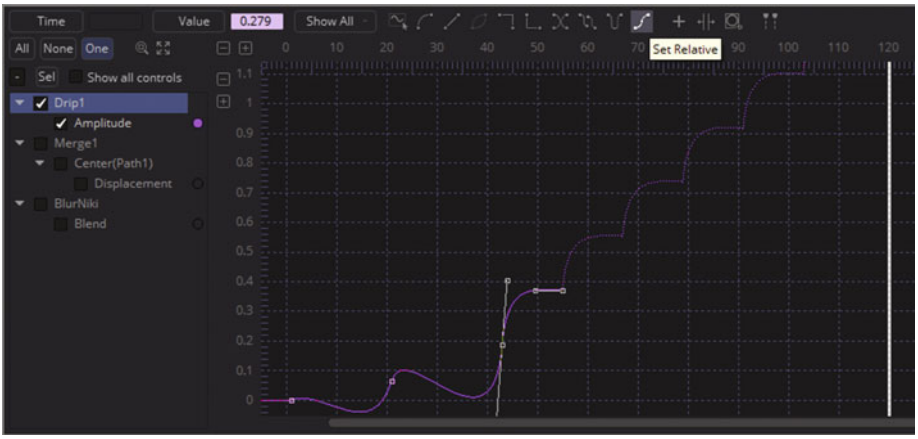

Figure 10-22. Click the Set Relative icon to loop and offset curve segment

Summary

In the tenth chapter, we took a look at Fusion's spline editor and spline curve concepts, tools, and algorithms (or macros). We looked at panning, zooming, selecting, and editing splines.

In Chapter 11, you will learn all about **masking, polylines**, and advanced concepts and techniques that are related to **animated polyline masking** in advanced VFX project pipelines.

Chapter 11

VFX Pipeline Pixel Isolation: Animated Polyline Masking

Now that we have covered the three major VFX editing areas in Fusion 8—Flow, Timeline, and Spline—it's time to get into some more advanced concepts. The concept of **masking** objects and areas was covered in Chapter 2 when we discussed using alpha channels to define transparency, which isolates the pixels that you wish to composite, blend, or process algorithmically—or all of the above, in some instances.

In still image compositing software packages, like GIMP, PaintShop Pro, or Photoshop, you can draw **splines** on your images to **cut** (or mask) **out** the object or element you wish to **isolate**. Fusion also uses this approach with what it calls *polylines*, which are the different types (natural B-spline, cubic Bézier, etc.) of mathematically-derived **curve topologies** that are supported for use in the VFX software package. We looked at these in Chapters 9 and 10, but we'll be getting into more detail regarding **polylines** in this chapter, as we see how to use them as inputs for masking out detailed areas of pixels that we want processed in our VFX composition pipelines. If that isn't advanced enough, we will also look at how you can animate the masks like you can with morphing and warping software.

In this chapter, we'll look at polyline types and tools, using polylines for masking and **rotoscoping**, animating polyline masks, using polylines for motion paths, and **double polylines**.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_11

Polylines: Masks, Rotoscope, and Motion

You have already seen polylines used in Chapter 9 for a motion path, and in Chapter 10 in your spline editor. Let's look into the differences between the different polyline (curve) types, and see how they are implemented in Fusion using the floating polyline toolbar that you first saw in Chapter 9 (Figure 9-11). After that, we'll take a look at how to utilize polyline tools.

Polyline Types: Cubic Bézier and Natural B-Spline

There are two types of control curves used in Fusion currently. One is the standard **cubic Bézier** used in software such as GIMP and Inkscape, and most other packages that use splines, and the other is the **natural B-spline**, which is more similar to the quadratic Bézier used in SVG (Scalable Vector Graphics). If you are a 3D modeler you may also be familiar with the non-uniform rational B-spline (NURBS) topology found in 3D modelers such as Moment of Inspiration 3.0. It is possible that other spline topologies could be added to Fusion in the future, especially due to its increasing support (fusion) of 3D-into-2D workflows.

Bézier Polylines: Two Tensioning Handles for Each Control Point

The Bézier polyline is used the most widely in Fusion 8, in the flow node editor (view panes), timeline editor, and spline editor, as we saw in Chapter 10. When Bézier handles are retracted (or are tangent to their control curve), they create a polygonal (straight) line control curve (open) or closed shape, as seen in the spline editor on the left side of Figure 11-1.

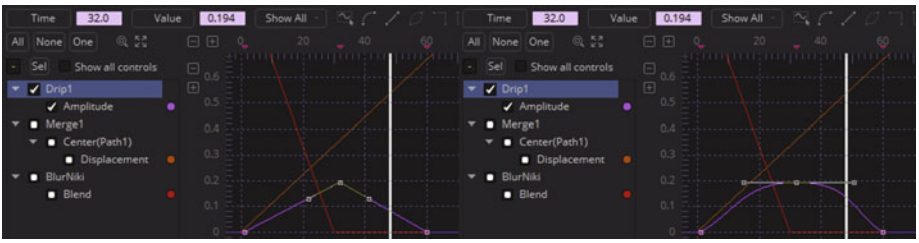

Figure 11-1. *Bézier polylines can be used in the spline editor*

When you pull these control curve tensioning handles out from the spline node or away from your polygon, you create your curvature (curve). Since I already covered a significant number of tools related to Bézier curves in Chapter 10, in this chapter I will cover new aspects of curve modeling so that I can cover as many topics as possible.

B-Spline Polylines: Bounding Cage for Organic (Smooth) Shapes

In most instances, one can achieve the same control curve results using the cubic Bézier topology as they could using the natural B-spline topology. B-splines (especially NURBS) will often be used for **organic modeling**, where things need to be curved rather than sharp. This is especially true for 3D character modelling.

B-splines cannot do "sharp" or polygonal (straight line) curve transitions, which is why they're not used in Fusion's spline (time) editor. Figure 11-2 shows the motion path from your Chapter 10 project. I inserted a control point halfway up and pulled the tensioning handles out to create a more natural, curved path for the jellyfish to follow on its journey to the surface. I then converted this to a B-spline so that you can see the difference. Figure 11-2 shows the Bézier on the left and the B-spline on the right. I pulled a couple of the control cage points away from the curve so you can see how this works more clearly, and to show how a curvature is maintained at all times with this spline topology, eliminating all jerky movements.

Figure 11-2. Bézier curve converted to B-spline

There are no abrupt transitions or changes in a B-spline like there are at the control point for the Bézier curve, which at the end (bottom) of the top half of the curve instantly goes in a different direction to start the bottom half of the curve.

Your **open B-spline** can have control cage handles on both sides of the curve, whereas a **closed B-spline** (shape) will only have control cage handles on the outside perimeter, which is why this topology may be desirable for use with organic masking shapes used to isolate areas in VFX composites to be processed.

The way you make this topology conversion in Fusion is to right-click on your Bézier spline, as shown in Figure 11-3, and then select the polyline motion path, which in this case Fusion named **Path1:PolyLine** (Component Name : Component Type). Then, select a **Bézier to B-Spline** topology conversion algorithm, using the resulting submenu.

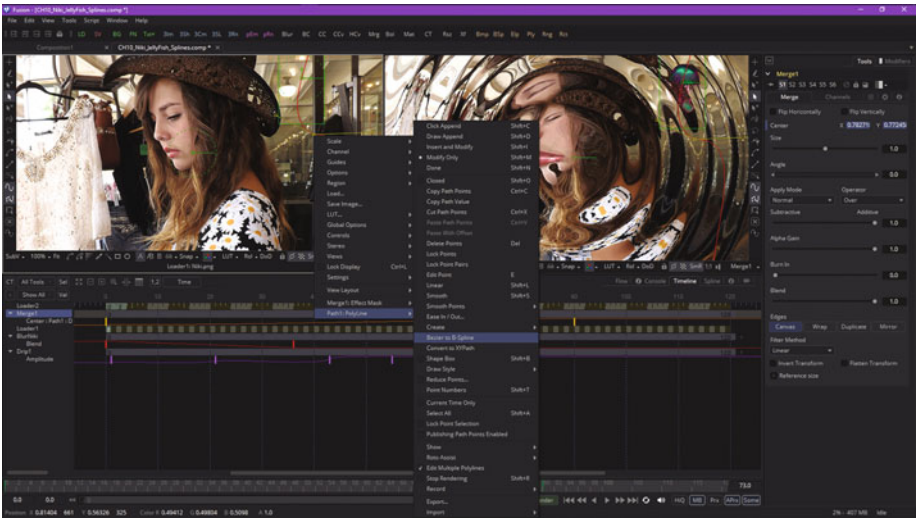

Figure 11-3. *To convert topology, use Bézier to B-Spline option*

Next, let's take a look at Fusion 8's polyline toolbar. We will go back to the Chapter 9 jellyfish composition project and edit the linear motion path from scratch. This project also has linear motion paths in the timeline and spline editors, so I can focus on polyline features starting with polygons, as all curves start out as straight polygon lines until tension handles turn them into curves.

Fusion's Polyline Toolbar: Working with Polylines

Open the Chapter 9 jellyfish project, and select the right view dot in the merge node, and in the timeline view select that merge node. This will give you the motion path and the polyline toolbar, as shown in Figure 11-4. Select the **insert and modify** polyline tool, seen circled in red, and click in the center of the path.

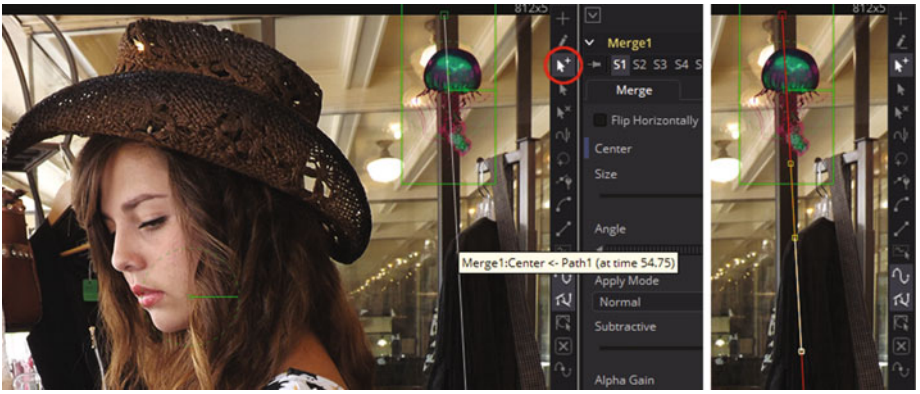

Figure 11-4. *To add complexity to a curve, use the insert and modify tool*

To maintain the straight polygonal line, your tensioning handles will be tangent to, or parallel with, the original line segment, so you have essentially converted a vector (ray) into a cubic Bézier spline, mathematically speaking, once you adjust those tensioning handles.

Next, mouse-over the bottom handle (it will turn white) and make the spline curved, so that the jellyfish travels more naturally through the composition. Drag the tensioning handles so that the jellyfish motion path has a nice S shape. Use your CRTL modifier key to move the handles independently of each other, and you can position your spline path end points as well using the arrow icon, which edits (but does not add or delete) point locations. This editing can be seen on the left in Figure 11-5.

Figure 11-5. Add a point to the motion curve, edit tensioning handles, and select Done

I circled the **insert and modify** tool, as well as the **show key points** and **show all handles** tools, which are all turned on, as designated with a lighter gray color. If you want to lock in your motion curve, use the **Done** option, as seen on the right side of Figure 11-5, which will turn the curve white, and prevent any further editing. If you want to delete points, turn on the icon second from the bottom (it is an X), and if you want to optimize your spline point count, use the **reduce points** tool, found at the very bottom of the polyline toolbar. Next, let's take a look at mask generation (creation) with Fusion, which uses a different set of polyline toolbar icons.

Visual Effects Masking: Selective Pixel Processing

There are several different types of masking that you can do in Fusion 8, including effects masking (also called post-masking), pre-masking, garbage mattes (also called garbage masking), and animated polyline masking. The one you'll use most often is post-masking, where the mask is applied after an effect.

Effect Mask: Applied Post-Effect to Isolate Areas of a Composited Element

Select the **Drip1** node in the flow editor, with Niki.png (Input) set to View1, and Merge1 (Composite) set to View2. Right-click **View1** and select the **Drip1: EffectMask** option, then select **Polygon** mask, as is shown in Figure 11-6.

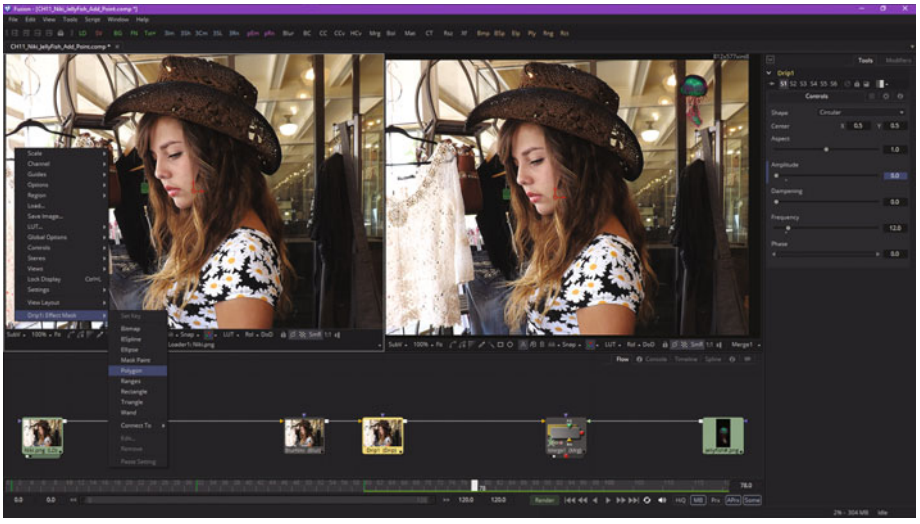

Figure 11-6. Select the Drip1 node and the Polygon effect mask

This will add a Polygon1 mask node tile above Drip1 into the mask input. Select this node, as seen in Figure 11-7, and a polygon mask polyline toolbar will appear in View1 on the left. Click at the top of the hat using the **click append** tool used in mask creation (both shown circled in red) and click out a rough outline of the model, as is seen on the right side of the model.

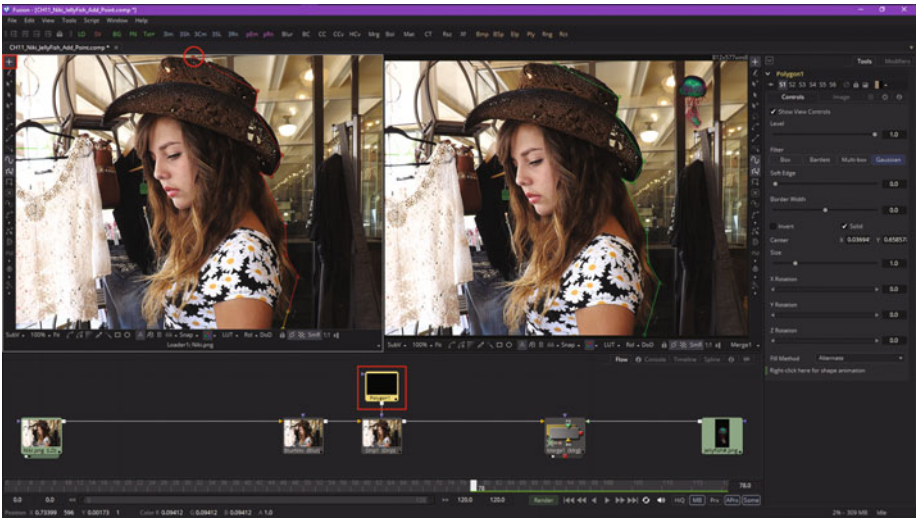

Figure 11-7. Use click append tool to create right half of mask

You do not have to spend a lot of time on this polygonal mask outline; just place vertices where your directional shifts in the mask perimeter occur, as later on you'll use tensioning handles to make these curve segments conform with an outline of the subject matter you are masking—in this case it is a model.

As you can see in Figure 11-8, once the polygonal cage is all of the way around your model, and you put your last spline point on top of the first spline point, the cursor will change to the close shape symbol, and the word **Polygon** will pop up in a tool tip, as shown at the top of Figure 11-8. Click to close the polygon, and a black and white representation of your mask will appear inside of your Polygon1 node tile's preview area, which you can see in Figure 11-9.

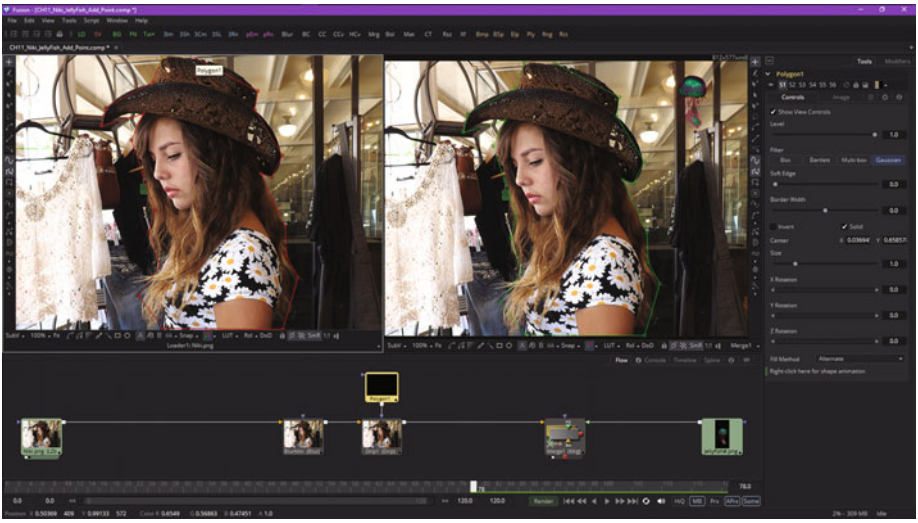

Figure 11-8. Use click append to finish mask, and then close polygon

Now you're ready to refine your polygonal mask, making it into a polyline mask—or a Bézier mask, more precisely—by selecting the arrow (point select) tool. Or, you can use the add point tool, as long as you only click on your points to select each one for editing. Use the handles to adjust the curvature of the polygon, point by point, until it fits snugly.

As you can see in Figure 11-9, I have turned all of the straight lines into custom curves, and this polyline mask edge-fits the model fairly nicely at this point. You can continue to refine your point placement and tensioning handles if you like.

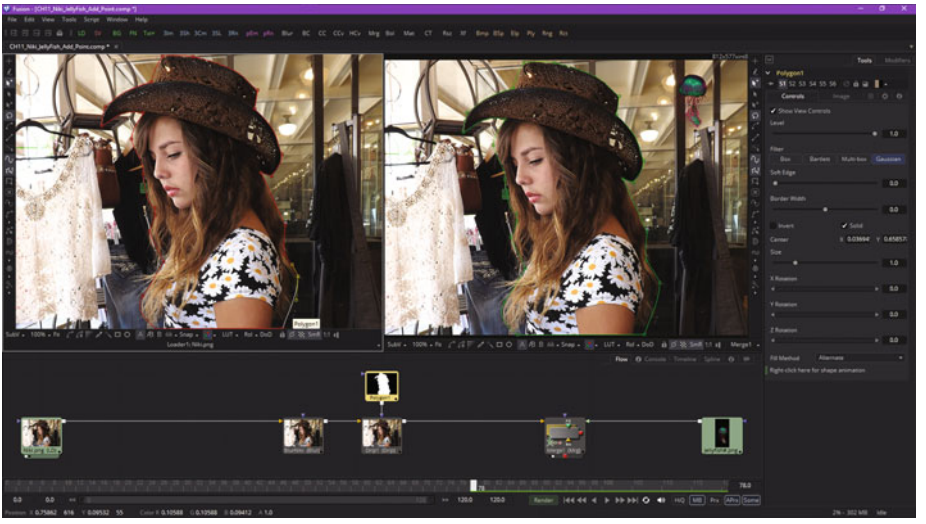

Figure 11-9. A closed polyline mask now shows in the flow node tile for your mask node

Next, drag a wire (pipe) from the Polygon1 mask node into the BlurNiki node mask input. As you can see in Figure 11-10 in View2, your mask now defines where your effects will be applied.

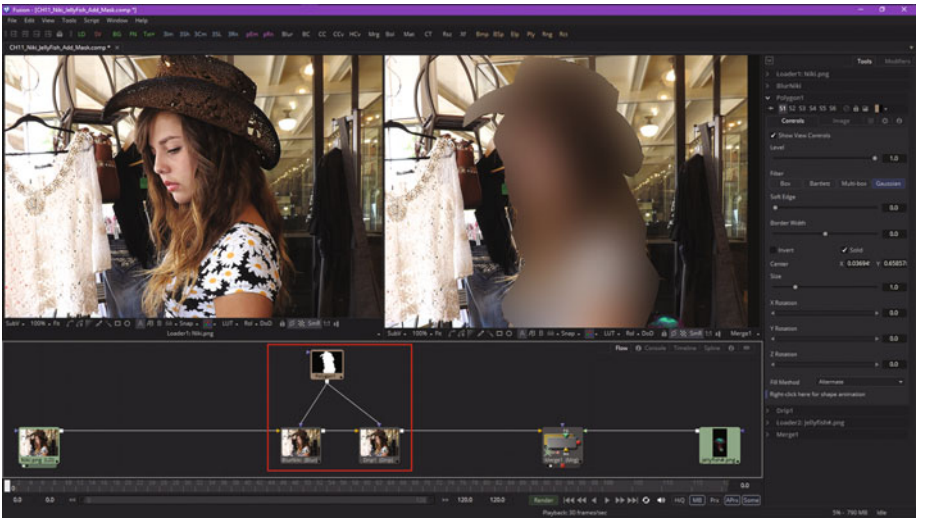

Figure 11-10. Wire the polygon mask node into the BlurNiki node

Play the effect in the timeline, as shown in Figure 11-11.

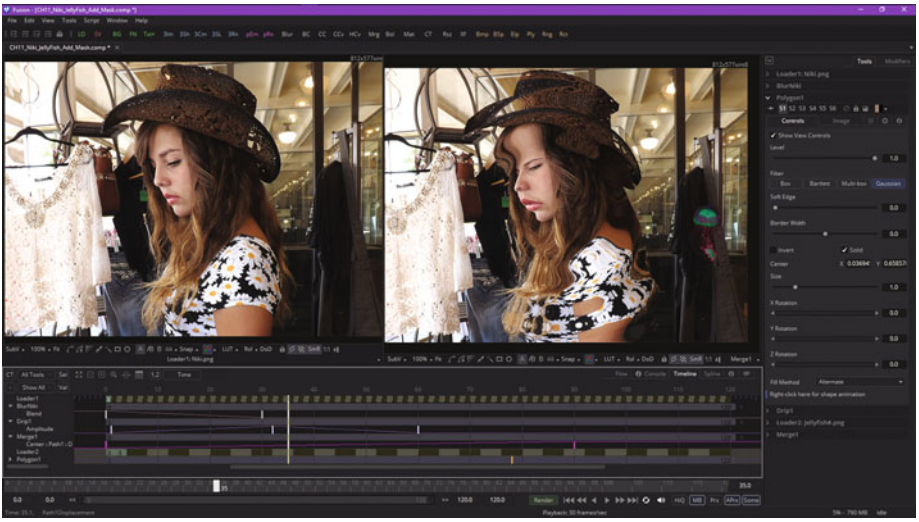

Figure 11-11. *Drag the playhead and animate the masked effect*

As you can see, a polyline mask now constrains the effect processing pipeline to the model herself, and leaves the store around her intact and unprocessed. You could combine this with a fade, for instance, and create a sci-fi transporter special effect.

To see the mask in one of the viewports, mouse-over your Polygon1 node and select one of the two viewer dots. I chose to display the mask in View1 and the merge in View2, as you'll see in Figure 11-12, and then I selected the merge node after that, so I could see the motion path controls more clearly. Your project pipeline is getting more complex, with multiple masked effects processing and composited animated imagery following a motion path, all combined seamlessly within the flow node editor.

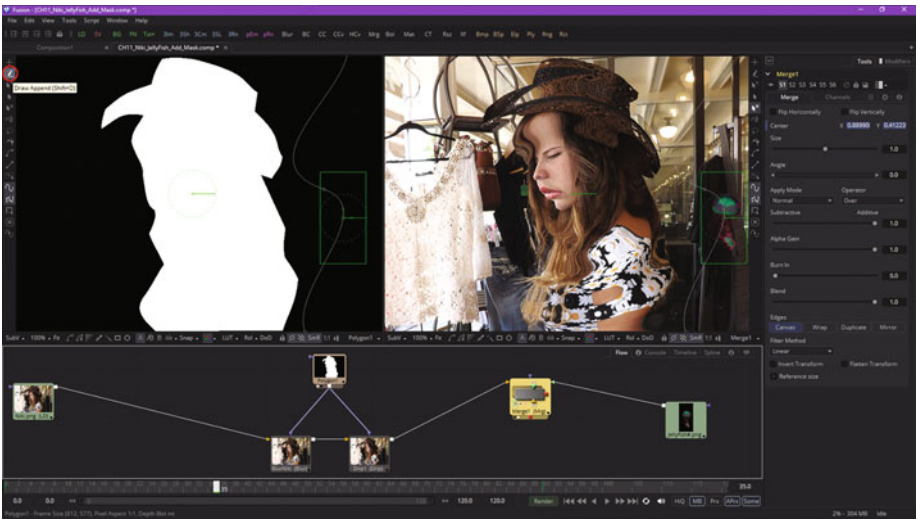

Figure 11-12. Show the mask of model in View1; and showing the draw append tool

Garbage Matte: Quick and Dirty Pixel Removal for Keying Tools

Directly underneath the click append effect mask creation tool in the polyline toolbar is the **draw append** garbage matte (mask) creation tool, which uses a **pencil** icon, seen circled in red in Figure 11-12. Unlike a click append tool, this draw append tool will give you a continuous, smooth curve result, rather than a polygon result, and will generate more data (control) points. I recommend using this tool for creating "garbage mattes" for use with your chroma-keying or luma-keying (greenscreen) workflows.

Since we will be covering this in the next chapter, I'll continue on to cover the concept of pre-masking. Then, we will go over how to animate polyline masks, and how to use double polylines.

Pre-Mask: Applied Pre-Effect to Isolate Application of Effect Itself

Pre-masking uses the same type of effects mask that was generated in the work process seen earlier in this chapter; the only difference is in when the mask is applied during the processing pipeline—hence its name. Unlike the effect mask we covered already, which I referred to as a post-mask, the pixels in the pre-mask will be processed by the node (tool) **before** any special effects (algorithms) are applied. Thus, in pre-masking, **only masked pixels are processed by the algorithm**, versus all image pixels being processed prior to the application of the designated mask (for post-masking).

One advantage of pre-masking is that each algorithm tool node will render more quickly, and in some cases will produce a more believable visual effect result. As an example, when using Fusion highlight or glow tools, a pre-mask approach can be used to restrict your lighting effects to certain areas of the image, while allowing the result of that effect to extend beyond the limit of the mask, especially if double polylines, which we cover later in this chapter, are being implemented. An advantage to pre-masking is implementing realistic behavior for glows and highlights, so that real-world results can be closely simulated in post-production using the Fusion VFX pipeline.

Let's change around your current project and implement a pre-mask. We'll use a transparent checkerboard pattern to show you visually what's going on for the rest of the chapter, as it will be easier to visualize. Right-click in the flow editor and select **Add Tool ➤ Composite ➤ Merge**, as shown in Figure 11-13.

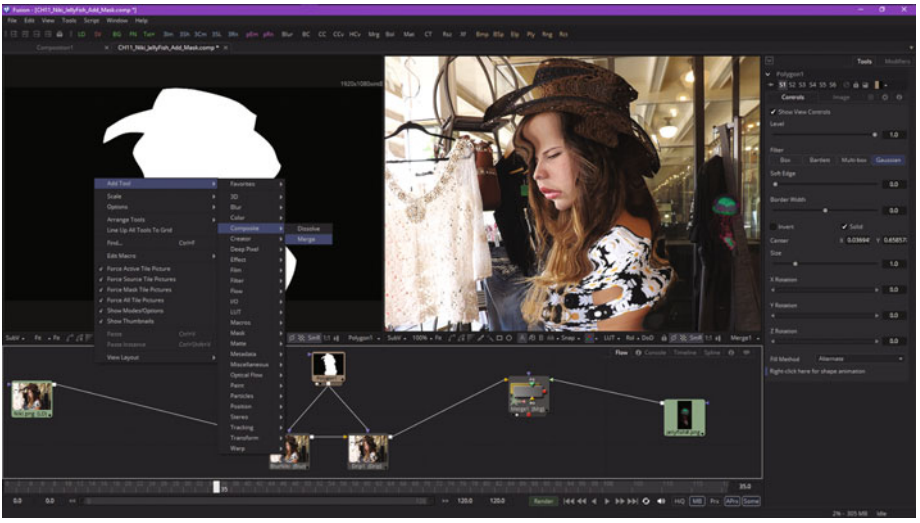

Figure 11-13. Adding a Merge2 compositing tool node to the project

Delete the polygon mask pipes connecting to the blur and drip nodes and connect your **Polygon1** mask output to the Merge2 node's **foreground** input. Your **Niki.png** image loader connects to the **background** input, and the output goes to your blur and drip nodes, as is shown in Figure 11-14. To show only the model over transparency, set the Merge2 **compositing operator** to **In** and the **apply mode** (also called a blending mode) to **Screen**, as shown on the right side of Figure 11-14, highlighted in red. Be sure to experiment with these five operator mode options, in conjunction with the seventeen apply mode options, as that should give you a feel for what these blending algorithms can do, at least visually.

To make sure that your polygon mask and Niki.png subject matter are synchronizing pixel-for-pixel, select your **Polygon1** mask node and select the **Image** tab and the **Output Size Custom** tab, then enter the resolution for the Niki.png image.

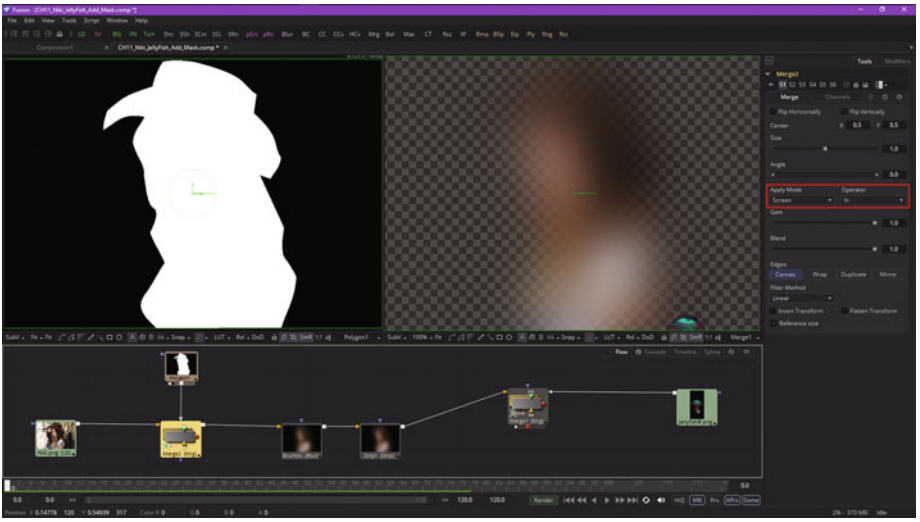

Figure 11-14. *Set merge operator to In, and the apply mode to Screen*

Enter **812** in the **Width** data field and **577** in the **Height** data field, as is seen in Figure 11-15. As you can see, you now have the Niki model only, running through an effects processing pipeline, with the polygon-masked subject matter on the transparent checkerboard pattern showing in the same viewport (View2) as the spline control points. The way that this pipeline is set up now will allow you to refine your mask while looking at all of your VFX pipeline processing in real-time, while also seeing the mask result in the left view in the same real-time! Fusion is really cool in how many different ways you can set up a VFX workflow, and all because it uses more advanced nodes, instead of layers.

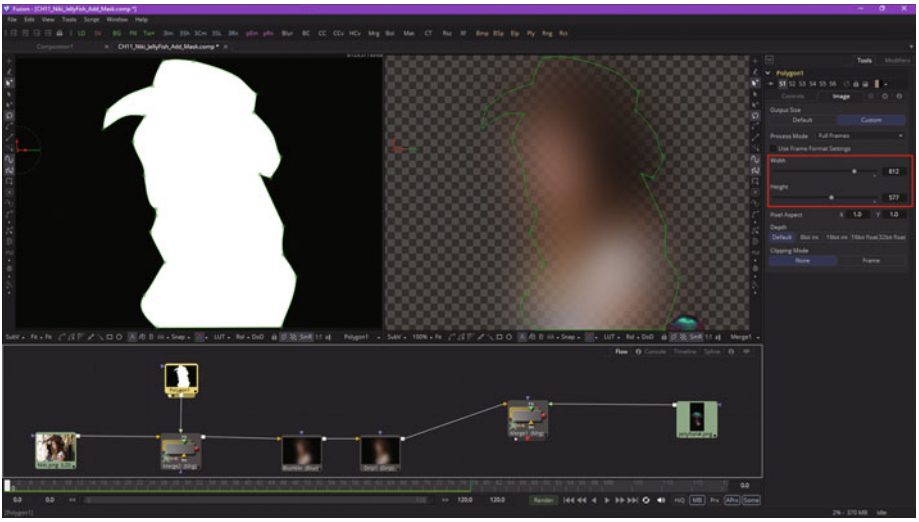

Figure 11-15. Set Polygon1 mask width and height to match the Niki.png image attributes

Figure 11-16 shows the use of the **modify only** tool, seen in red, along with the **close the polyline**, **show key points**, and **show all handles** icons, all toggled on, and therefore in use. Adjust your polygon, and watch your VFX pipeline process in real-time!

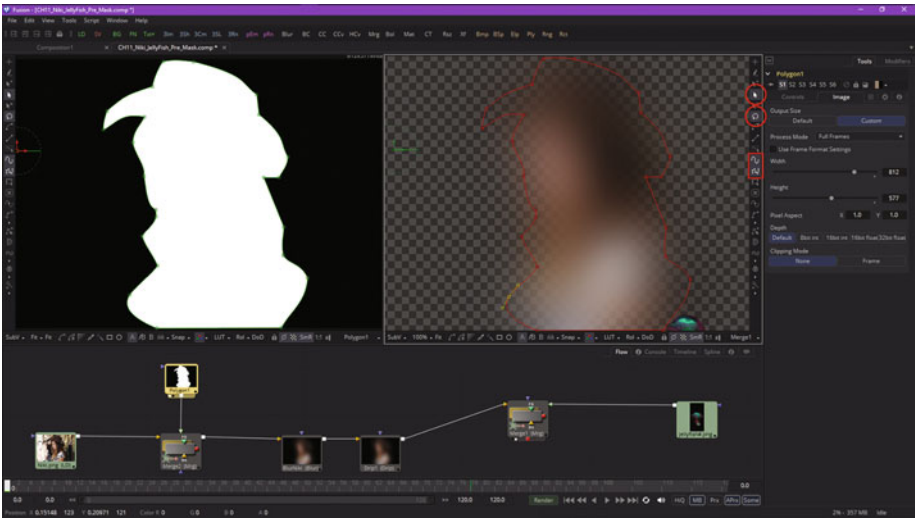

Figure 11-16. Use the modify only tool to modify your pre-mask in real-time

Next, click the **Timeline** tab, and drag the playhead to **65** to choose a frame that doesn't have any blur or drip visual effect processing going on. This can be seen in Figure 11-17. You can now further edit your mask polyline. I show a point selected that's not really needed, just so I can show how to use your **delete points** tool, shown circled in red along with the control point I'm deleting (and the time ruler keyframe 65 and the Timeline tab).

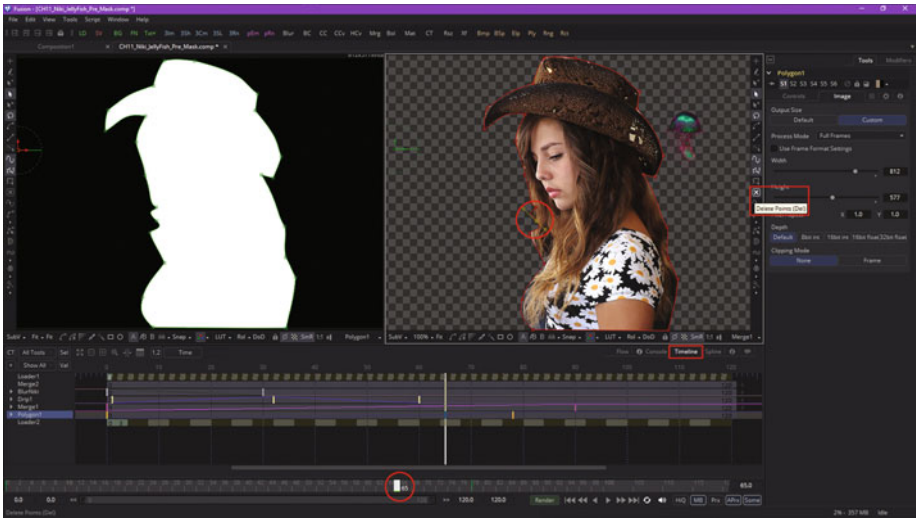

Figure 11-17. *Move the playhead to frame 65, and delete unneeded points*

Next, let's take a look at how to animate polyline masks.

Animated Polyline Masking: Keyframing the Mask

Even more advanced VFX pipelines can be created by using **animated polylines** to create masking for your visual effect. This method can be used with static imagery, image sequences, vector assets, imported digital video, or any other effect or objective that needs to be masked. Animating polyline masks is done much like an effect node animation is accomplished—by enabling animation and then changing your playhead position, and then adjusting control point position and handles on different frames. Fusion will observe what you are doing and will create your keyframe data.

For the current project, this should amount to selecting the **Polygon1** mask node and, in the Control Panel area, clicking on the **Controls** tab and, at the bottom, right-clicking on **shape animation** (control), as is shown in Figure 11-18.

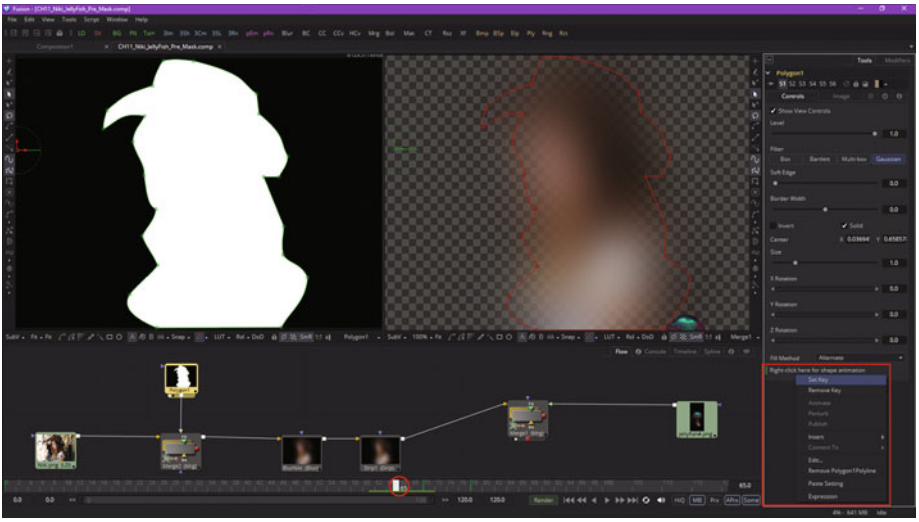

Figure 11-18. Select Polygon1 mask node, and set shape animation control key in the Controls Tab

Select **set key** and set your initial keyframe, then enter the **timeline editor**, as seen in Figure 11-19, and go to frame 70 using the playhead. Pull some polyline control points out at the bottom of the mask. This will show both the result and the mask!

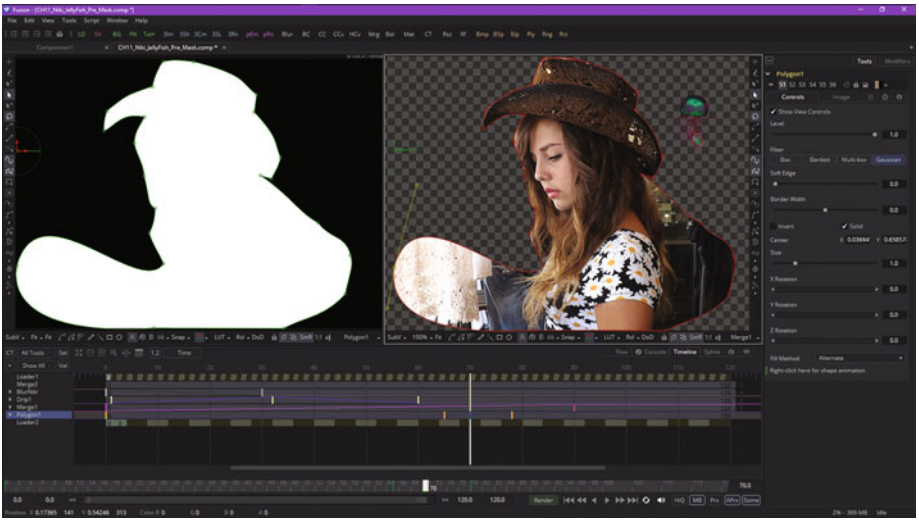

Figure 11-19. Move playhead to 70 and edit some control points

To see the smooth polyline mask animation, simply drag the playhead back and forth between keyframes 65 and 70. It will show your polyline mask "morph" result about halfway between the keyframe range, and, as you can see in Figure 11-20, the result is quite professional! The primary reason Fusion uses polylines for masking rather than using a pixel-based approach is to enable polyline animation, as we are learning here, to be done quickly and professionally.

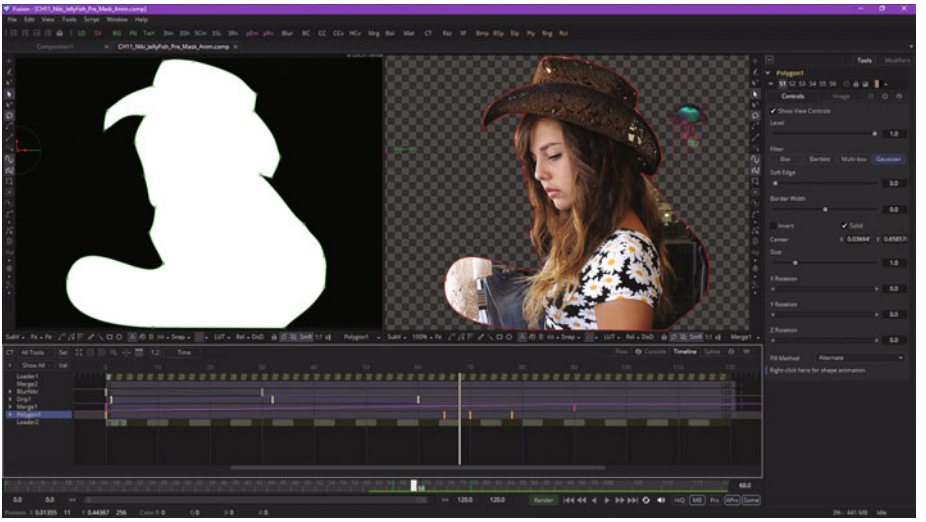

Figure 11-20. *Drag the playhead and watch your polyline animate*

Next, let's take a look at **mask edge softening** controls.

Double Polylines: Blurring the Edges of Your Mask

As you may know from other popular new media software such as GIMP, masking tools can apply a blur function to your mask to soften the edge for the entire mask. Since the blur is applied across the mask, it applies a softening effect equally across all of the edges. However, there will be times when softening isolated parts of the masking polyline while keeping other parts razor sharp will be desirable for the project. Fusion calls the controlled softness *non-uniform softness*. In Fusion, this is created by converting the polyline shape or segment from single polyline to double polyline. Double polylines are comprised of an inner and outer shape. The inner shape is the original polyline shape. An outer shape can also be added in order to determine the spread area for the softness, so that you can control how any effect ramps in 2D space.

The greater the distance between the outer shape and the inner shape, the softer that edge portion of the shape becomes. To convert the current mask into a double polyline, click the **Double Polyline** button, seen highlighted in red in Figure 11-21, in both viewports, as we have set up the flow node editor to show the polygon mask in View1 and the merge composite in View2 so that we can see the mask and its result at the same time.

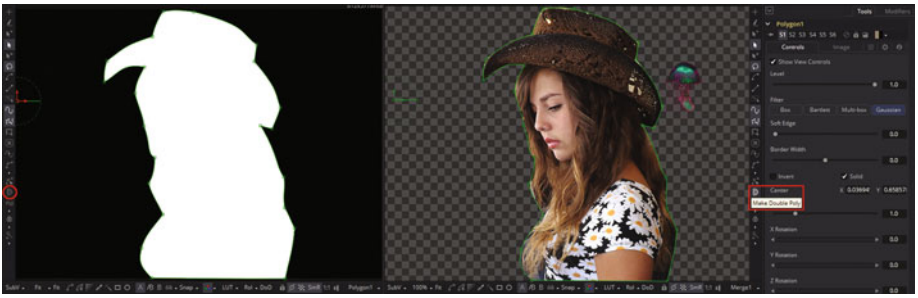

Figure 11-21. *Click the make double tool and select a polyline*

You can also right-click in each viewer and select **make outer polyline** in the polyline mask's context menu. The shape will be converted into a synchronized inner and outer polyline, represented by a red and green dashed spline, as seen in Figure 11-22. As you would expect, both polylines start with the same shape as the original source polyline. This keeps the mask sharp (as its default setting) and, more important, will allow animation that has been applied to a polyline to remain intact. Use the modify only tool to move some outer polyline points, as seen in Figure 11-22, so as to create your double polyline softening.

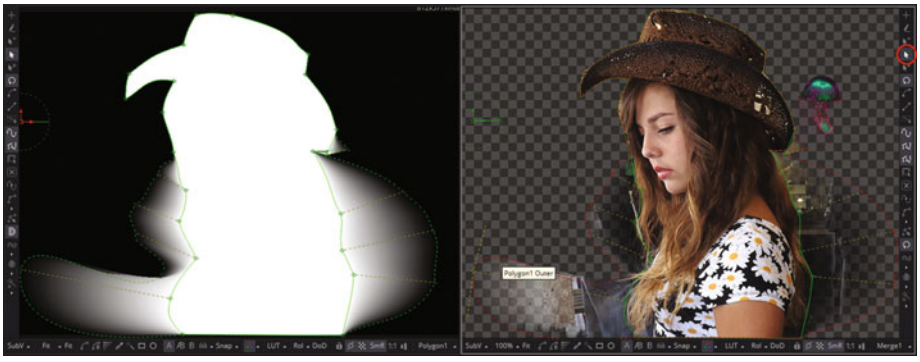

Figure 11-22. *Showing Controls and Effect of Double Polyline Mask*

If the spline is de-selected, you'll need to click on it, with the double polyline tool active, in order to show Fusion 8 which polyline you want to double. What is really cool is that in the mask-only View1, you can see the exact amount of blur or softening that is being applied as you work with this tool. Next, drag your playback head and watch your doubled polylines animate! Since you have not keyframed your outer polyline, only your inner polyline animates, thus animating your softening effect, as can be seen in Figure 11-23.

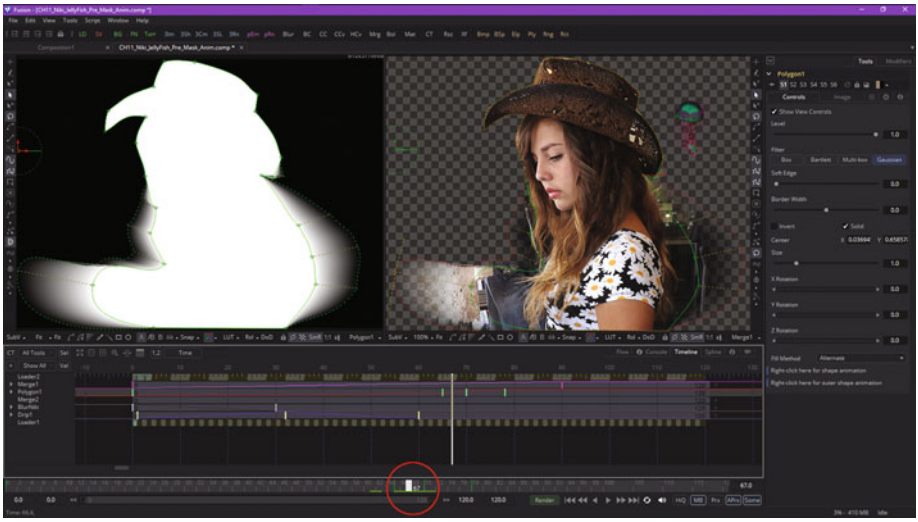

Figure 11-23. *Drag your playhead and watch the double polyline animate the soft blur area around the model*

As you are starting to realize, there is a considerable amount of work to be done after you set up your VFX pipelines, especially on a keyframe by keyframe (effect and mask tweaking) basis. We can look at how to set up these tools and how they work in this fundamentals title, but true expertise with Fusion comes from time spent fine-tuning VFX projects until the result is visually perfect, as well as seamless with actual **perception of reality** standards present in film, television, and 3D gaming.

Summary

In this eleventh chapter, we took a look at how advanced masking is accomplished in Fusion. We looked at how to create masks, how to animate masks, and how to control edge-softness blending using double polylines. In Chapter 12, you will learn about keying, or **chroma keying** and **luma keying**, concepts and techniques.

Chapter 12

VFX Pipeline Automated Masking: Matte Generators

Now that we have covered the advanced concept of animated mask creation, let's cover another advanced VFX concept and technique called either chroma keying or luma keying in the VFX industry. Since VFX are frequently shot in front of a bluescreen or greenscreen, this is a common feature in VFX software packages, and one that you really need to understand. The technology behind keying algorithms is complex and provides powerful tools for you to integrate into your VFX project pipeline.

We will look at how **chroma keying** is done with **Primatte V** and Fusion's internal legacy keying algorithms. We will look at the **chroma key** tool, the **ultra key** tool, and the **Primatte** tool, which has partial support in Fusion and full support in Fusion **Studio.**

Fusion 8 Keying: Concepts and Tools

Bluescreen keying, and more recently greenscreen keying, has become a mainstream compositing approach for working with both VFX and 3D software packages. This is because it automates the often tedious masking process, although it requires that you shoot in front of special blue or green cloth, paint, or sets. The **keyer** tool has algorithms that then extract these colors, leaving only the subject matter on transparency, creating the alpha channel algorithmically for the VFX artisan. I will be using an image

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_12

sequence provided to me by my friend **Scott Gross**, who has worked at Photron (the creator of Primatte) since day one. This professional **chroma-keying** software package is available for Autodesk (Smoke, Flint, Flame, Burn, Inferno), Adobe, Quantel, and Apple, and is integrated into both Fusion Studio 8 and its primary competitor, Nuke. There is a basic functionality for **Primatte** in Fusion (free) as well. Therefore, I will start with this Fusion 8 keyer tool. There are a significant number of different keyer tools available in Fusion 8; we will cover the ones used the most frequently during this chapter.

Professional Keyer: Using Basic Primatte V Keyer

Open Fusion and right-click **Add Tool ➤ I/O ➤ Loader** import Niki.png to use as your backplate (background), and then use **Add Tool ➤ Matte ➤ Primatte** to add the Primatte keyer, as shown in Figure 12-1.

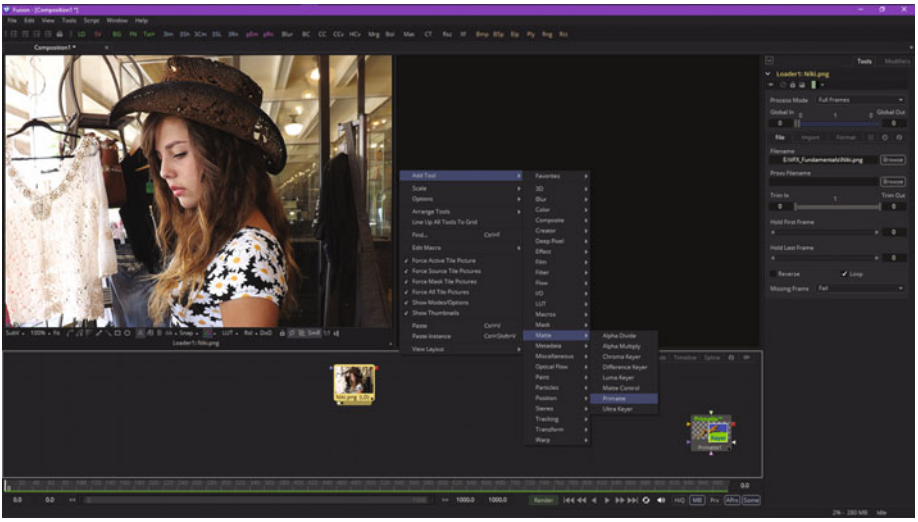

Figure 12-1. Add loader and Primatte nodes to a new VFX project

Right-click again in your flow node editor, then use the **Add Tool ➤ I/O ➤ Loader** context-menu sequence to get to the loader tool's **Open File** dialog. Add a sequence of bluescreen images, numbered Alex1.tif through Alex20.tif, as shown in Figure 12-2.

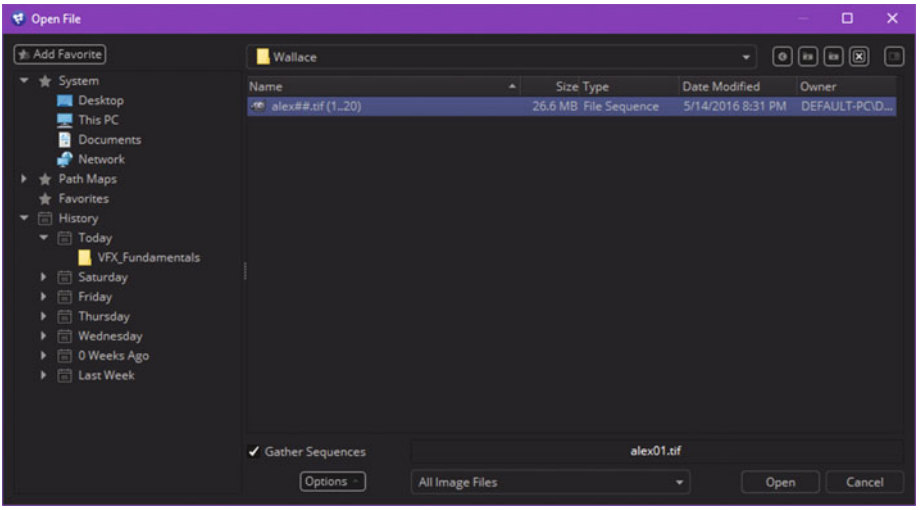

Figure 12-2. Import the series of twenty Alex.tif images

Wire Niki.png into a **Primatte1.Background** input triangle, and wire alex##.tif into a **Primatte1.Foreground** input triangle. Select the Primatte1 node (tile), and click the View2 dot at the bottom, to show your Primatte keyer in the view on the right, with Niki in the left view. Notice Fusion will label the views if it has room. This is shown in Figure 12-3, along with the **Auto Compute** button, which you are going to click next, to invoke the proprietary keying algorithm in Primatte. This will automatically remove green or blue background color from the bluescreen or greenscreen asset.

Figure 12-3. Connect the background image and foreground series

When you click the **Auto Compute** button and trigger those auto compute algorithms the Primatte keyer is world famous for, Primatte can “pull a perfect key” for you, without your having to set any parameters at all.

The results of this can be seen in Figure 12-4 in View2, along with some advanced Primatte algorithms and settings that are available in the Fusion Studio version of the software, but which have been disabled in Fusion’s free version, at least for now. If you do a lot of bluescreen and/or greenscreen work, you may find it worthwhile to pay the $995 for Fusion Studio 8. You would have to pay between two and four thousand for Primatte on other platforms, so this tool alone is worth the Fusion Studio $995 upgrade price.

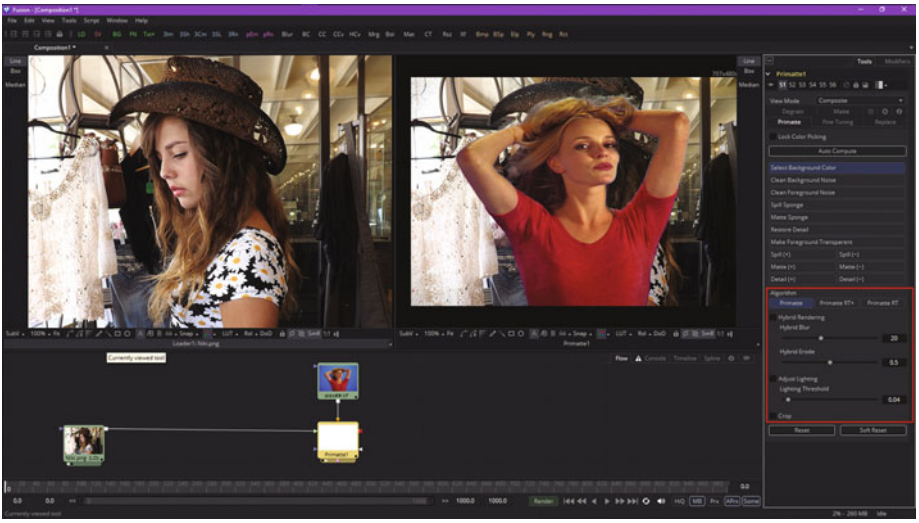

Figure 12-4. *There are some Primatte V Keyer settings that require Fusion Studio*

If you click on any of the controls or options shown in the area circled in red on the right-hand side of Figure 12-4, you will get the dialog shown in Figure 12-5, which advises you that you can upgrade to Fusion Studio to unlock a full set of advanced options for professional Primatte V (5.1) use.

The **Primatte Algorithm Feature Unavailable** dialog can be seen in Figure 12-5 and has a link to Blackmagic Design's website.

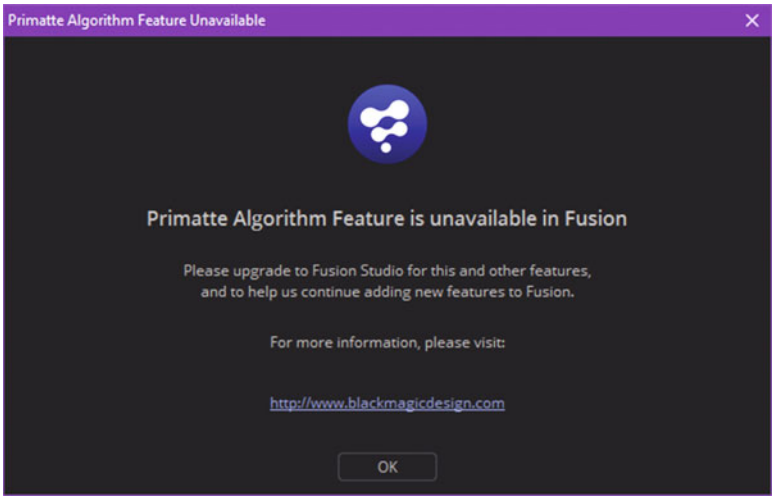

Figure 12-5. *Upgrade to Fusion Studio dialog*

Chroma Keying: Using the Fusion 8 Chroma Keyer

Since we can't get into Primatte's advanced options, let's take a look at the other keyer algorithms in Fusion, which are nothing to sneeze at, as these can produce professional results as well. The most basic is the chroma (color) keyer, which keys based on color. There is also a luma keyer, which keys based on light. If you wanted to shoot against a red screen background, you would use the Chrome Keyer.

We will look at other Fusion keyer types later on in this chapter. Start a new Fusion project, and again add loaders for Niki.png and alex##.tif image resources. This time, right-click and **Add Tool ➤ Matte ➤ Chroma Keyer** to add a chrome-keying tool (algorithm), as can be seen in the middle of Figure 12-6.

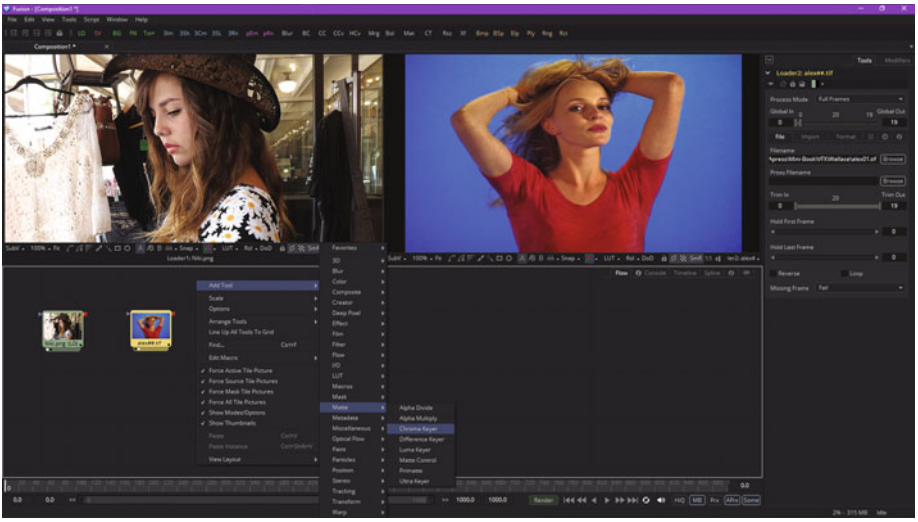

Figure 12-6. Start a new project, and add a chroma-keyer tool node

Next, wire the alex##.tif image series loader node into the chroma keyer node so as to provide the chroma-keyer algorithm the data (pixel arrays) that it will process. Using Fusion shorthand, the way this is written is `Loader2:alex##.tif.Output -> ChromaKeyer1.Input`, and it is shown in Figure 12-7.

Figure 12-7. Wire an image series Image Loader node into the chrome-keyer tool node

Select your chroma-keyer node and set its viewer dot to View2, which is also shown in Figure 12-7, along with the main control panel, highlighted in red on the right side.

Make sure the (default) Chroma button is still selected, then drag a selection marquee around as much of the bluescreen as you can, without including any other color, as can be seen on the right side of the View2 (right) viewport.

This will add these bluescreen color values to the keyer node's algorithm for processing later. You can do the selection operation more than one time, if you see that the algorithm has not removed all of the blue values from your result, which will be shown to you in the ChromaKeyer1 viewport (View2) as long as you have selected the right-hand dot at the bottom of the node. As you will see in Figure 12-8, there are some blue values left at the top-middle and lower-left part of the image; be sure to marquee select these as well, as seen in Figure 12-8, and these will be added to the algorithm.

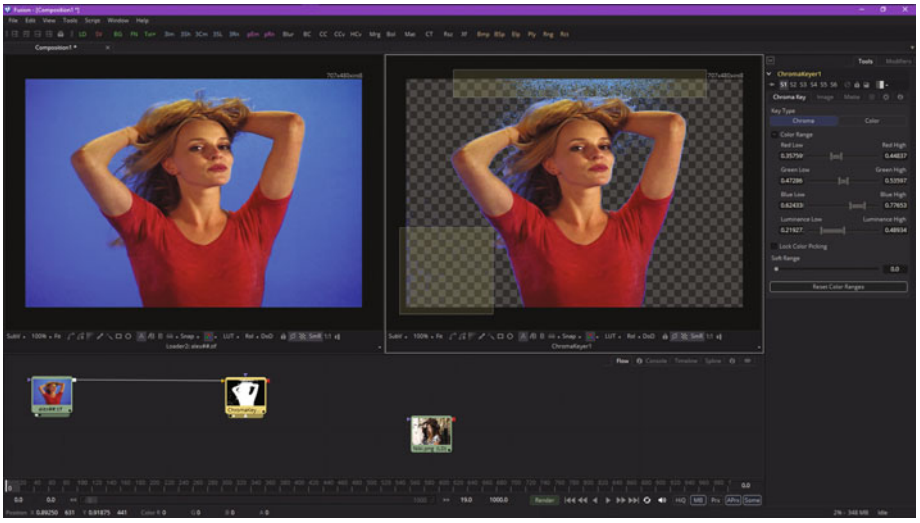

Figure 12-8. *Continue marquee selecting blue values to add to your key mask*

As you can see in Figure 12-8 and Figure 12-9, whenever you select an area of blue color values, they are removed from the image mask result. This allows you to see what an algorithm is doing in real-time, so that you can achieve a progressively better result with this tool by using this marquee selection work process.

Figure 12-9. Zoom in 400% by using zoom widget at bottom of View2

As you can see in Figure 12-9, there are some light blue pixels still remaining near the model's hair. To make selection of these pixels easier and more precise, use the **zoom** widget at the bottom of your View2 viewport, which should be set to 100%, also known as View Actual Pixels, and use your drop-down arrow—or, in this case, it's a pop-up arrow—to set the zoom to four-times magnification, or **400%**, as seen in Figure 12-9. Select an area of light blue pixels that does not overlap with the model's hair, as seen at the top of the right-hand pane in Figure 12-9.

Once you have selected all of the blue values available for selection in your transparent (masked) areas, which will be done on the pixel level, you can click the **Image** tab, seen on the right in Figure 12-10, and use the **spill method** algorithm setting to remove the bluish halo around the image subject.

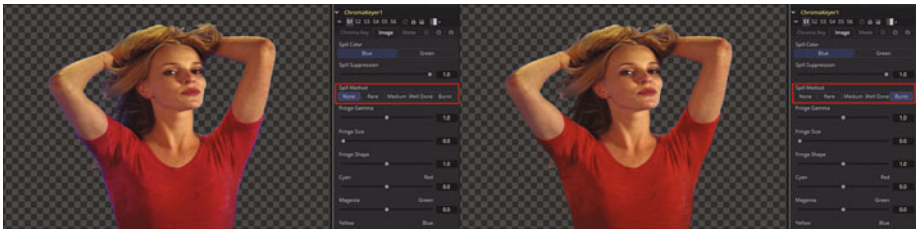

Figure 12-10. Adjust the spill method algorithm settings under the Image tab

To composite this result over another image, use a **merge** (composite) tool using the **Add Tool ➤ Composite ➤ Merge** context-menu sequence to add a merge node, as is shown in Figure 12-11.

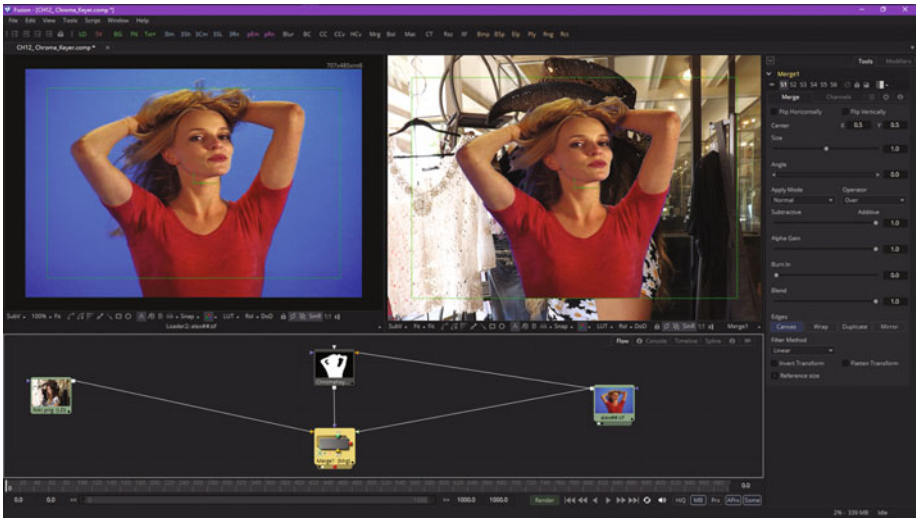

Figure 12-11. Add a Merge node to composite the matte over the Niki model image

In the flow node editor, wire Niki.png as the background plate and the image series as the foreground image, and then wire the chroma-keyer output into the mask input, as shown at the bottom of Figure 12-11. Notice that you're giving the data (pixels) in the image series to the chroma-keyer node algorithm, as well as giving it the masked foreground image for your composite merge operation. Since Fusion is chroma-keying imagery algorithmically, the algorithm will work for every image in the series in the exactly same way.

Bluescreen and Greenscreen Keying: Ultra Keyer

Whereas the chroma keyer will work with any color you wish, in case you want to shoot on a redscreen, for instance, there is a keyer called the **ultra keyer** that is even more optimized for use with **green or blue** in Fusion. If you do not upgrade to Studio and unlock Primatte for your greenscreen work, this is what you would want to use instead. Launch Fusion again (or start a new project) and use **Add Tool ➤ I/O ➤ Loader** to add the series of 20 TIFF images so that we can look at this algorithm next. I will forgo the backplate image and merge composite steps, as the work process will be the same as in the previous section. Figure 12-12 shows the image series loader node and the context-menu sequence (**Add Tool ➤ Matte ➤ Ultra Keyer**) to add the ultra-keyer node tile, which I show selected on the right

side of this figure. Notice that your control panel options are much simpler, as this algorithm is already tuned for blue and green.

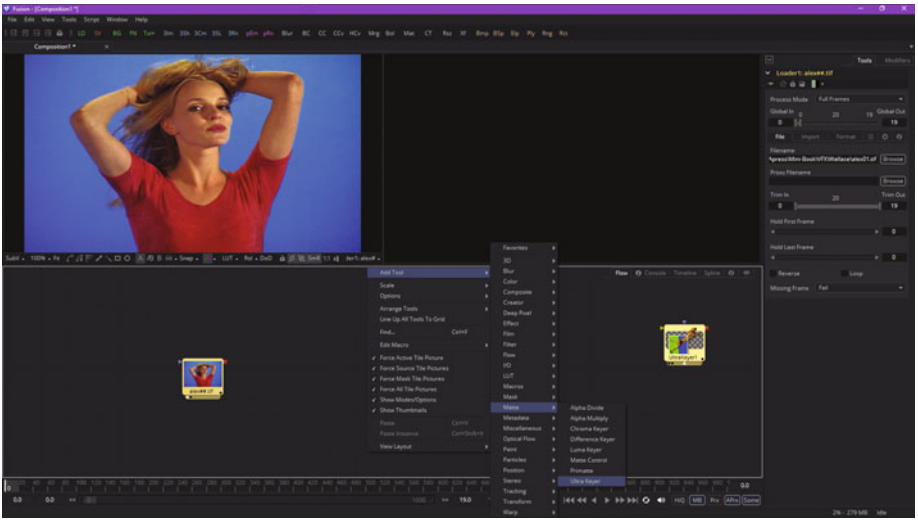

Figure 12-12. Add image loader and ultra-keyer nodes in Fusion

As you can see in Figure 12-13, once you wire your nodes together the algorithm is already removing the blue values. It does this all at once, and some of the transparency checkerboard pattern is visible. To control the result, use the two sliders highlighted in red in the ultra keyer control panel.

Figure 12-13. Use the ultra keyer control panel to adjust the bluescreen removal effect

Drag your **background correction** slider to the right as much as is needed to remove the rest of these blue values. This was a value of 0.9750 on my system, which allowed me to achieve complete transparency, as can be seen in Figure 12-14 in View2.

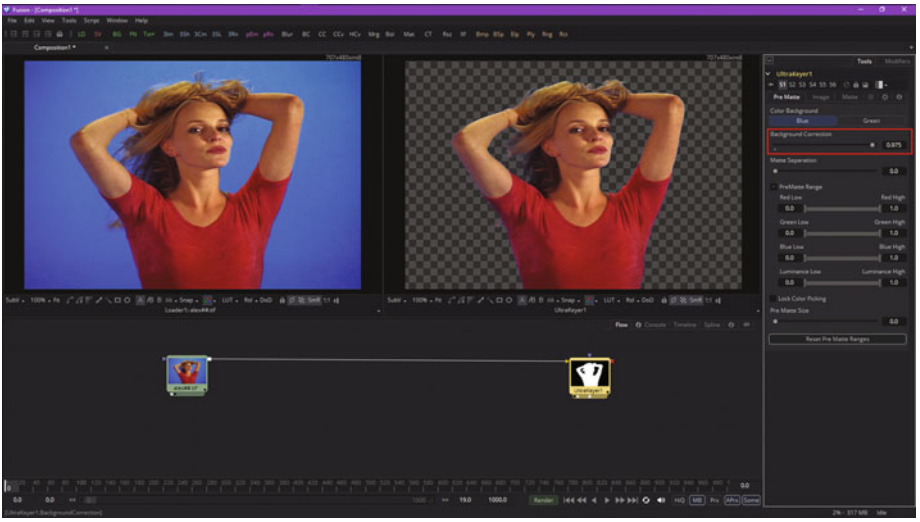

Figure 12-14. *Remove all blue values with a 0.975 setting*

Note that the result is superior to the previous chroma key algorithm, as there is no blue fringe that will need to be removed! This is because the ultra keyer's algorithm is specifically optimized for bluescreen and greenscreen, which both use a very carefully pre-defined color value, which the ultra keyer was looking for. Also notice in Figure 12-14 that I have expanded a section called **PreMatte Range** underneath the two primary sliders for ultra keyer. This contains RGB color sliders that allow you to select colors that you want to also be transparent in addition to the blue (or green) values used in the bluescreen (or greenscreen) assets.

Figure 12-15 shows how this works using red color value slider settings. You can use this to remove other colors in the mask that you are creating. Here, we use the **red high** value, between 0.65 and 0.67, to attempt to remove everything except for the red shirt.

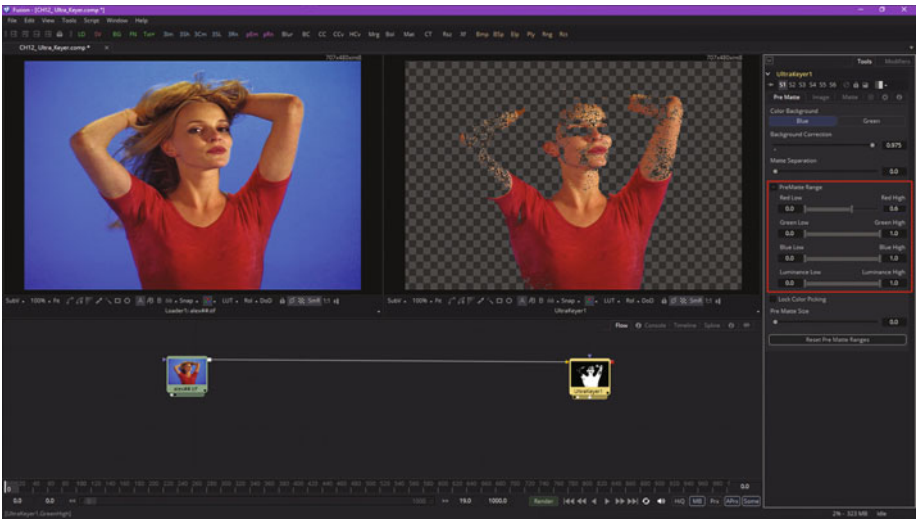

Figure 12-15. Use a red high slider value to isolate red pixels

The **PreMatte** tab is the most important ultra-keyer tab, with settings for **blue** (bluescreen) or **green** (greenscreen). Based upon that background color selection, the ultra keyer will iteratively merge the pre-keyed image over a blue or green background before further processing, guided by your **background correction** slider setting. In certain use cases this could help to improve edge quality results. Matte separation helps separate the foreground from the background before color selection.

You can increase the **matte separation** while viewing your alpha, as shown in Figure 12-16, to eliminate a majority of the background. Stop increasing this slider if you see holes in the subject or loss of detail on the edge of the matte. You can set the view to show the **alpha** data using the viewport RGBA widget, as can be seen in the pop-up menu at the bottom of View1 in Figure 12-16.

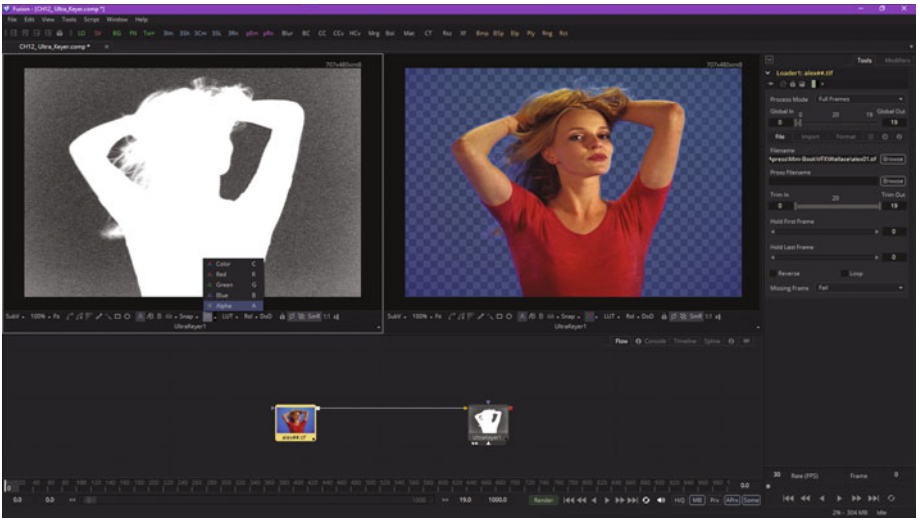

Figure 12-16. Set both views to ultra keyer and set one to display only alpha channel data

I went back and used the new alpha channel view to reset my primary two sliders, finding that I only needed to use **0.55** for the background correction and **0.3** for the matte separation, as I could now better see the result using the Alpha black and white view, as shown on the left side in Figure 12-17. The optimal mask has only black or white pixels, with gray anti-aliasing pixels along the edges. Next, let's take a look at your Image tab controls.

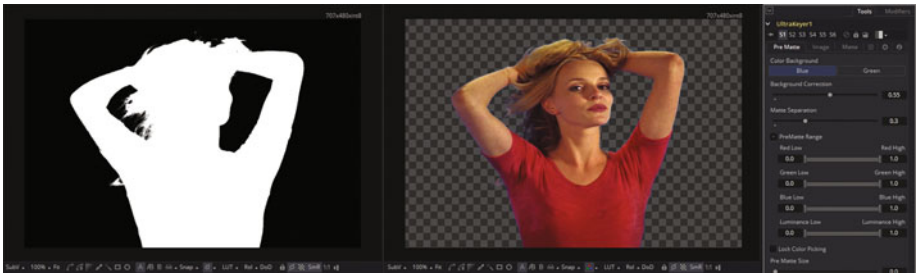

Figure 12-17. Use the alpha view to set a more precise setting value for matte separation and background correction

As you can see in Figure 12-18, I left **spill suppression** at the maximum 100% (1.0) setting. Spill is the blue fringe you saw back in Figure 12-9, and is caused by the background screen color coming through anti-aliased or semi-transparent grayscale areas of the alpha channel. In blue- or greenscreen keying, this can cause the color of the background to become apparent on the "fringe" of that foreground object that you are pulling a

mask for. A spill suppression algorithm will attempt to remove color from this fringe area. The process used is optimized for either blue- or greenscreen based on the color you selected in the **PreMatte** control panel.

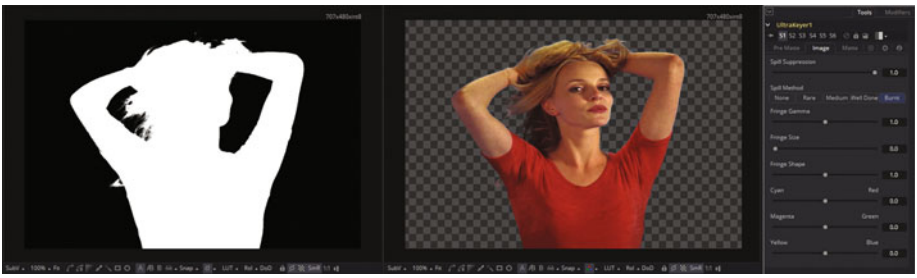

***Figure 12-18.** Select Burnt spill-suppression method, remove blue fringe pixels*

The **spill method** selects your spill suppression algorithm strength. The **None** button is selected when no spill suppression is needed. The **Rare** button removes a little of the spill color, and is the lightest spill suppression method.

The **Medium** button is recommended by Fusion for use with greenscreen, and the **Well Done** button is recommended for use by Fusion in bluescreen. Finally, the **Burnt** button works best with bluescreens, and is primarily used for troublesome shots.

A **fringe gamma** control will be used to adjust the brightness of any fringe (also sometimes referred to as a "halo") that may surround your masked (ultra-keyed) image subject matter. **Fringe size**, on the other hand, expands or contracts the width of the fringe that is around the subject matter. The **fringe shape** will force the fringe to be pressed toward the external edge of the image or pulled toward the inner edge of the fringe. Its effect is most noticeable when the fringe size slider value is large.

You can also use your Cyan to Red, Magenta to Green, and Yellow/Blue color sliders to color-correct the semi-transparent pixels around the fringe of the subject matter. This allows you to color-correct your fringe (instead of removing it) to match the new background color (if that color is uniform, that is). I used the Burnt setting, seen in Figure 12-18, to remove a blue spill color along the fringe of the masked subject.

The third tab is the **Matte** tab, which allows you to post-process your matte (mask or alpha channel) result once you have fine-tuned your bluescreen removal using the first two tabs.

The **matte blur** will essentially anti-alias a jagged edge on a matte (mask) by applying a small amount of blur along the edge of the matte using a Gaussian blur, just as you would do in GIMP or Photoshop using a 0.15 to 0.2 blur value. In Fusion, to apply a 0.2 Gaussian blur uses a 1.2 setting,

as seen on the right side of Figure 12-19. Notice in your left alpha view that the edges of the mask are no longer jagged and are instead nice and smooth, using the gray value to anti-alias the jagged edges away. You can drag a slider to see the amount of blur applied.

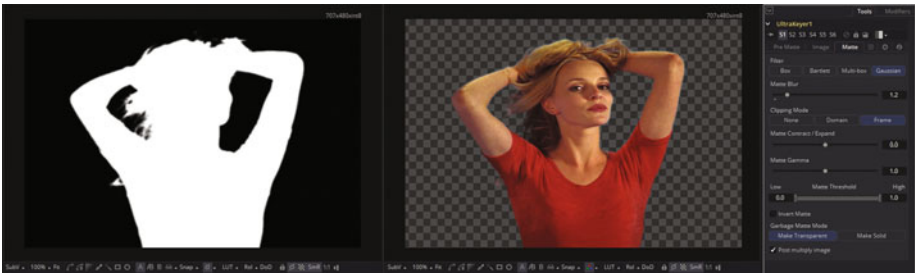

Figure 12-19. Apply a Gaussian blur to anti-alias the mask edges

Other matte (mask) fine-tuning controls include a **matte contract/expand** slider, which will shrink (contract) or grow (expand) the semi-transparent (anti-aliasing) areas of the matte. Values above 0.0 will expand the anti-aliasing area, and values below 0.0 will contract the grayscale areas in the mask. Your contract/expand control should be utilized with your matte blur to take away any hard edges on a matte (mask) and to reduce any color fringe. Since this control only affects the grayscale (semi-transparent) areas for the mask, it has no effect on black and white areas (the matte or mask of the subject itself).

The **matte gamma** slider will raise or lower the values of the matte in semi-transparent areas. Higher values should cause the gray area to become more opaque, and lower values cause the gray area to become more translucent. Again, any black or white region of a subject matter matte will remain unaffected.

The **matte threshold** control has two slider settings, one for each end of a spectrum, allowing you to set a range rather than define single data values. Any data value below the low threshold becomes black (transparent) in the matte. Any values above the upper threshold will become white (opaque) in the matte, whereas all data values contained within this range should keep their defined transparency values, relative to the magnitude of the defined range. This control can be used to mitigate a “salt and pepper” noise effect in the matte. To show this and the “Invert Matte” checkbox seen in Figure 12-20, I went back to the original zero settings for the first two control panel tabs so that we can use the matte threshold setting to remove the background noise in the original matte. I used a 0.45 Low setting and 0.6 High setting to remove 95 percent of the noise, and I also selected the “Invert Matte” option to show you what that can do for a mask.

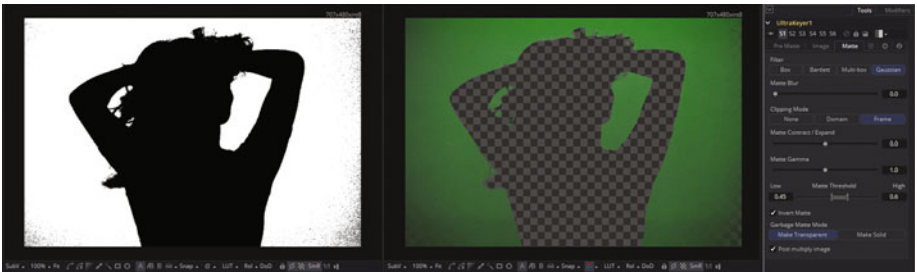

Figure 12-20. Use matte threshold to remove noise; invert mask

When your "Invert Matte" checkbox has been selected, alpha channel data that has been created by a keyer will be inverted, causing the transparent areas to be opaque and the opaque areas to be transparent. This is useful to "flip" or reverse a mask. Sometime it is easier to mask what is not the subject and then invert the mask, so use of mask inversion is very common in the VFX industry.

There is also a **garbage matte mode** pair of buttons that allow you to define how garbage mattes (selection of areas in a mask that are "garbage" and can be quickly removed in bulk) are applied to (summed with or subtracted from) the detail sections of the matte.

Garbage mattes are mask tools or custom images connected to your garbage matte input on your keyer tool (node) tile. The garbage matte input data will then be applied directly to alpha channel data of the matte. Garbage mattes can be used to remove unwanted elements on your greenscreen set that you do not want to be keyed, such as reference props for actors, microphones, or booms used for camera or audio equipment.

The **Make Transparent** button should be used to make your garbage matte transparent (black), whereas the **Make Solid** button should be used to make the garbage matte solid (white).

The **"Post Multiply Image"** checkbox will instruct the keyer to multiply the color channels for your image against the alpha channel created for your image. This option is usually enabled, and therefore will be on by default. If you deselect this, your image can no longer be considered "pre-multiplied" for purposes of merging it with other images.

As you have seen in this chapter, keying, along with the masking covered in the previous chapter, is an advanced VFX topic that really deserves its own book in order to cover it saliently. For this reason, make sure to check out the other algorithms on the **Add Tool ➤ Matte** submenu, as they allow you to do even more with your alpha-channel creation and matte pulling than what we have had the time to cover within this chapter.

Summary

In this twelfth chapter we took a look at how to work with bluescreen and greenscreen still image and motion image assets in Fusion. This is called *keying* in the VFX industry. We looked at the professional keyer in Fusion, called Primatte V, and then we looked at the Fusion chroma keyer, which allows keying using any background color value. The more advanced ultra keyer is optimized for bluescreen and greenscreen usage specifically. If you are not upgrading to Fusion Studio, these are the primary matte keyer algorithms that you will be using in the Fusion 8 free software version, which is why I covered these specifically during this chapter. Thanks again to Scott Gross of Primatte for providing me with the bluescreen Alex model image sequence that I used to show off these keyer tools. Make sure to get your own greenscreen dropcloth or paint soon!

In Chapter 13, you'll learn about Fusion **motion tracking** concepts and techniques.

Chapter 13

VFX Pipeline Pixel Tracking: Using Motion Tracking

Now that you have covered the advanced concepts of animated mask creation and bluescreen and greenscreen keying, let's cover another advanced concept and technique called **motion tracking**. Since visual effects are often applied to video footage, this is a common feature in VFX software packages, and one that you will need to comprehend. The technology behind motion tracking algorithms is complex, but it can provide powerful tools with which you can integrate **video** into the VFX project pipeline.

We will look at how **motion tracking** is done with **pattern recognition** and tracking, including what a **pattern** is, what the **search area** is, how you track **multiple patterns**, **stabilization**, **pattern channels** selection, **adaptive pattern tracking**, **offsets**, **motion stabilization**, **movement matching**, and **positioning**.

Motion Tracking: Concepts and Tools

Motion tracking is a major part of working with both VFX and 3D software packages, so this will be a good chapter for use with Fusion, which is both VFX and 3D software in one combined software package. I'll use some videos from the `sample-videos.com` website and `hollywoodcamerawork.com` website to demonstrate motion tracking concepts using Fusion over the course of this chapter.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_13

Track a Pattern: Algorithmic Pixel Region Analysis

Your **pattern definition** is a region of pixels that are selected for tracking in a video frame using the **tracker** tool in Fusion. I downloaded `SampleVideo_1280x720_2mp.mp4` from `www.sample-videos.com` for use in showing Fusion's basic motion tracking functionality in this section of the chapter. Open a new Fusion composition, right-click in the flow node editor, and **Add Tool ➤ I/O ➤ Loader** to import this sample video, shown as number 1, in Figure 13-1.

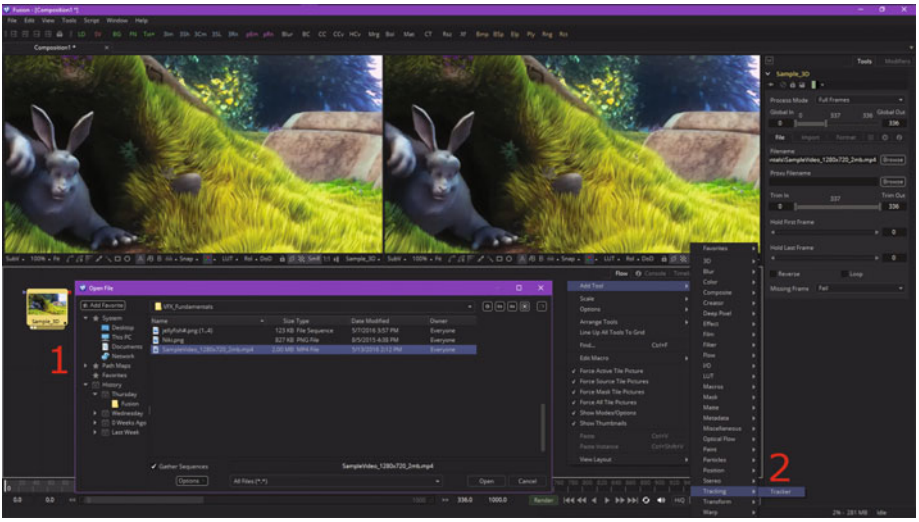

***Figure 13-1.** Add image loader and motion tracker nodes to a new VFX project*

Right-click again in your flow node editor and use the **Add Tool ➤ Tracking ➤ Tracker** context-menu sequence to add your motion tracker tool node, shown as number 2 in Figure 13-1.

The pattern definition widget will be represented in the viewer using a solid green rectangle when your tracker tool has been added, as you can see in Figure 13-2, circled in red. I've also found the cut in the sample video clip at frame 209, and so I set this frame as the end of this working segment, also seen circled in red at the bottom in the time ruler section.

Figure 13-2. Set the working segment to end at 209 (time ruler)

A motion tracking tool can define more than one pattern, and each of your pattern definitions will produce its own path. When you first add a "virgin" tracker node to your flow editor, Fusion starts you out with one pattern displayed in the viewer using a small green rectangle.

This rectangle will turn red when it is selected, as you will soon see in Figure 13-3. When your cursor is placed over this pattern definition rectangle, the control expands, and two rectangles will appear. The outer rectangle will have a dashed line, while the inner will use a solid line. The outer rectangle is your **pattern search definition** area, and the inner rectangle is the **pattern definition** area.

Figure 13-3. Various stages of use of pattern definition widget

If you need to select a new pattern definition area, you will move this widget by clicking on the small white box at the top left of the pattern rectangle and dragging the widget into position in the first frame (in this case) where you are going to define the pattern you want to give to this motion tracker.

Whenever you move the pattern rectangle, an overlay will appear that shows a **zoomed-in version** of the pixels contained within the rectangle, as is shown in the middle view in Figure 13-3.

The **zoom widget** is designed to help you to position the pattern definition with pixel-accurate precision. Notice that I have placed the pattern center at the tip of the bunny ear where the **contrast** between the light and dark pixels is the highest.

The pattern definition rectangle can also be resized by dragging on its edges. When the cursor is over the on-screen control widget for a pattern definition, a second rectangle with a dotted border should appear around the pattern definition. The outer rectangle defines the **pattern search area** from which a motion tracker algorithm will sample tracking data.

When processing from one frame to the next during motion tracking, the motion tracking pattern recognition algorithm will analyze the search region surrounding the last tracker position in an attempt to recognize the pattern definition in the next frame. The larger the search region definition is in pixels, the better chance the motion tracking algorithm has of successfully tracking rapidly moving patterns (subjects, objects, etc.).

On the other hand, the more pixels this algorithm has to process, the longer it will take to calculate the motion track. Therefore, there is a trade-off that you need to consider when setting the size of your search region definition (outer box).

There are some obvious ways to optimize your pixel array, which contains your search area tracking data. If you happen to be tracking an object that is moving rapidly across the screen from left to right, the algorithm will require a search area pixel array that is wide but not tall, since all the movement is horizontal. If your search area is smaller than the movement of the pattern from one frame to the next, your tracking algorithm will likely fail to find your subject (pattern) and could start tracking the wrong pixels.

It is therefore important to take the speed and direction of object motion into consideration when setting your search area. It is also interesting to note that the size of the search area can be animated over time, if that is needed for your project.

Animating the search region definition can be useful when the motion tracker pattern definition accelerates or decelerates rapidly part of the way through your source digital video clip. Now, let's generate some motion tracking keyframes!

Create a Motion Path: Generate Tracker Keyframes

Let's take a look at how to generate a motion path that will follow the face of the subject we are tracking. Once the pattern is selected, you can instruct a tracker to create a motion path that represents the position of your tracked scene element—in our case, it's the tip of an ear—in each frame. Each pattern enabled in the tracker list, shown on the right in Figure 13-2, will produce its own path data; how you use this data is based on your VFX project objectives. Before instructing a tracker to create a path, you need to set your render range, during which a pattern is moving through a frame—in our case, this range is frames 0 through 209. You learned how to set the render range in Chapter 9. Once you have set the render range, you can click on any of the tracking transport buttons, seen in Figure 13-2 at the top of the Tracker1 control panel, and initiate tracking algorithm processing. Once motion tracking algorithmic processing has begun, you cannot work in the flow node editor until the motion path generation process has been completed.

I used the far right motion tracking generation button, **Track Forward**, with my keyframe location at zero. There is also a **Track Forward from Current Time** button to the left of that.

To stop the motion tracking algorithm, use the red middle button (**Stop Tracking**). You can even run the tracking algorithm in reverse using the **Track Reverse** and **Track Reverse from Current Time** buttons seen on the left. There are visual representations of each of these button labels on these motion tracking icons.

When I clicked the Track Forward button, using the pattern settings seen on the right-hand pane in Figure 13-3, there weren't enough search region pixels for an algorithm to develop a motion path. Also, since the ear quickly moved onto a colored background for most of the scene, rather than staying on the dark black color, the pattern itself disappeared, at least from the perspective of the algorithm. Therefore, I switched from the ear to the chin, as seen in Figure 13-4, and adjusted my frame range from 8 to 192, as shown circled in red in the time ruler. I also enlarged the search region significantly and selected the **"Best Match"** option in the algorithm's **adaptive mode** setting, shown circled in red.

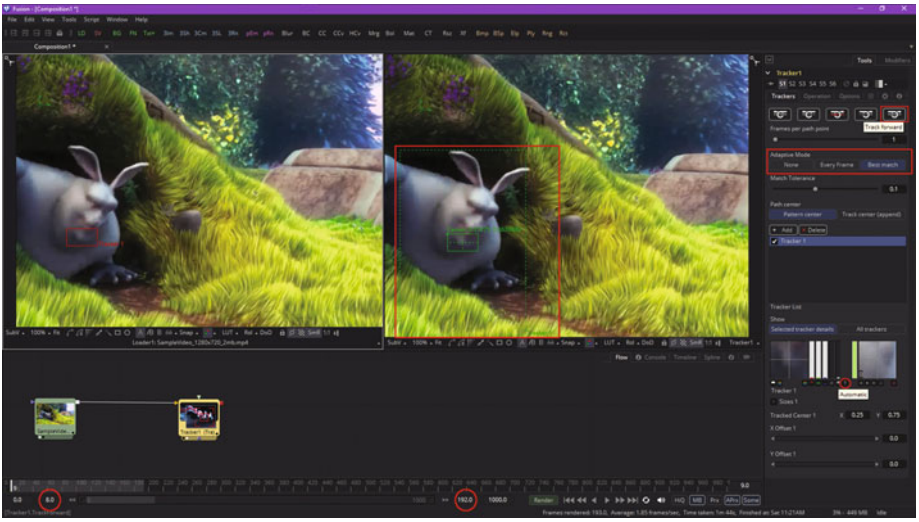

Figure 13-4. Set a new range, search area, and adaptive mode

The pattern recognition algorithm offers three different approaches (each is a different algorithm), as the "ideal" pattern, if there actually is such a thing, would almost always undergo shifts in pixel orientation, color, lighting conditions, and other "chaotic" variables present in digital video footage.

These variables can adversely affect pattern recognition to the point that a pattern will actually become unusable from the algorithm's (mathematical) perspective. This is what happened to me with the ear pattern definition, so I changed the pattern as well as the **adaptive mode** that I was using for this tracker.

An adaptive mode is applied to all active patterns while the motion tracking algorithm is processing. If you only want some pattern definitions to use the adaptive mode setting, then you will need to disable all other patterns in the tracker list before tracking.

The tracker list is under the Add and Delete buttons in the area I highlighted in red on the right side of Figure 13-4.

The tracker tool offers three distinctive modes for its pattern detection algorithm, which is used during motion tracking and can help to correct for your unforeseen pattern conditions.

These modes can be set using your adaptive mode buttons, seen in the tracker control panel in Figure 13-4. When adaptive mode is set to **None**, a pattern within the rectangle is acquired when the pattern is selected, and that becomes the only pattern that will be used during the motion tracking. If **Every Frame** is chosen, the pattern in the rectangle will initially be acquired

when the pattern is selected and will be acquired again on each subsequent frame. A pattern found for frame 1 will be used in a search on frame 2, a pattern found in frame 2 will be used in a search on frame 3, and so forth. This method helps the tracker's pattern recognition algorithm adapt to changes in motion pixels over time due to lighting, shadows, color, and subject movement.

The downside to the Every Frame adaptive mode is that it is processing-intensive, as the algorithm does a lot more searching and defining of patterns. It can also has a tendency to "drift" due to sub-pixel pattern shifting from one frame to the next. Only use Every Frame adaptive mode if your other methods or settings have failed to produce the intended result.

Finally, the **Best Match** adaptive mode should function in the same fashion as Every Frame tracking. The difference is that Best Match, which is what I chose to use, will not redefine your pattern if the difference between the original pattern and the new pattern is too significant. Rather, it will default to your old pattern so as not to diverge onto a different (incorrect) path.

This helps to prevent the case where transient change in your video asset causes the tracker tool to become disoriented. For example, if a shadow passes over a pattern definition, your Every Frame tracking mode can start tracking the shadow instead of the defined pattern. Best Match mode should detect a change, and thus won't grab the pattern from that frame since the difference from the previous pattern is too extreme. I changed my adaptive mode and made my search area larger, and the tracking worked!

Figure 13-5 shows the tracker output in View2 (the image loader tile was set for View1, as is seen in Figure 13-4), which shows the algorithm setting the pattern definition rectangle on each frame and rendering that portion of the video inside of it to show you what's being done. I showed two different frames in the sequence, so you can see what your motion tracker algorithm is doing.

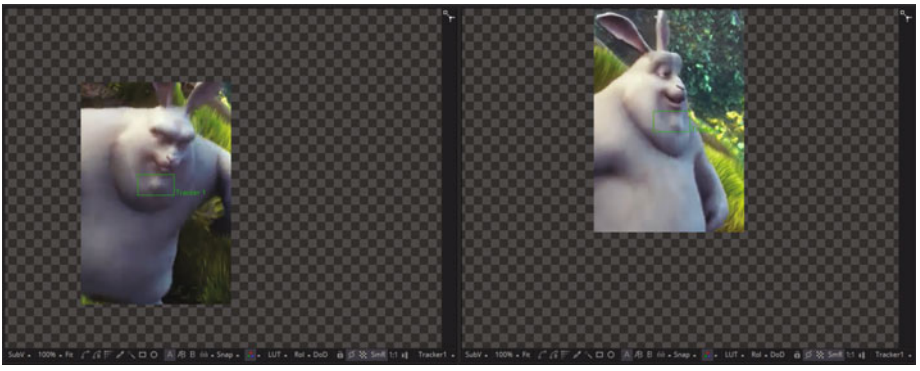

Figure 13-5. *Tracker will render the search area for each frame*

Once the rendering of the motion path has been completed, you will get a Render Completed dialog, as seen in Figure 13-6, along with the motion path, shown in green in both viewports.

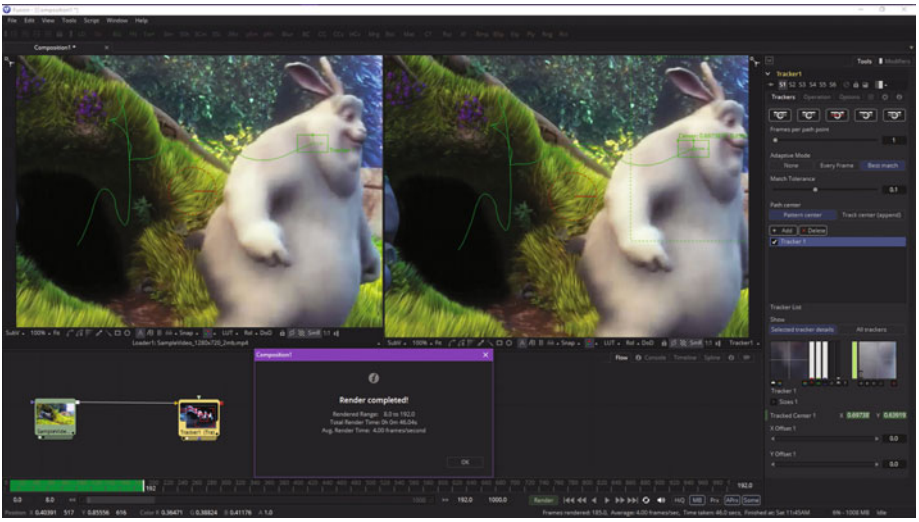

Figure 13-6. Once render is complete, you'll have a motion path

Let's add a couple more trackers to your bunny's face!

Track Multiple Points: Use More Than One Tracker

Click on the **Add** button, shown circled in red in Figure 13-7, to add your second tracker algorithm to the tracker tool node.

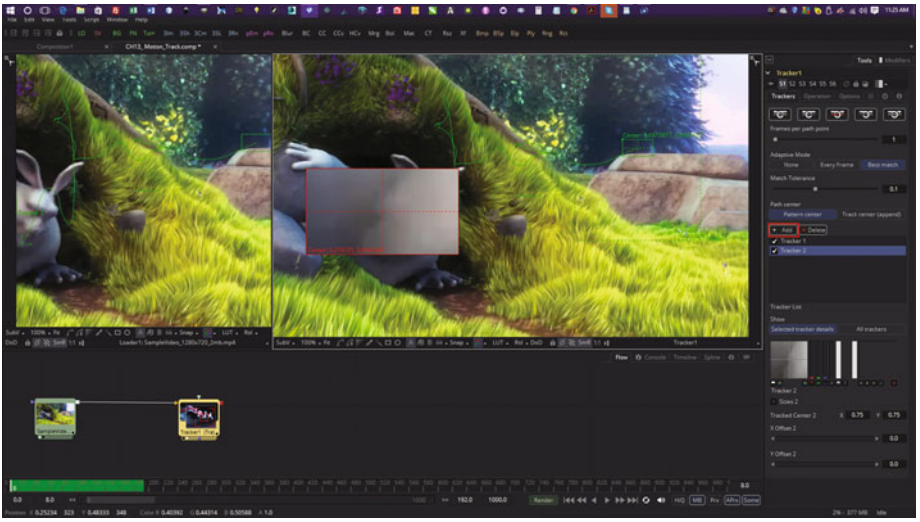

Figure 13-7. Use Add button to add additional motion tracker

Place your second tracker on the left cheek. Add a third tracker, as seen in Figure 13-8, and place it on the right eye.

Figure 13-8. Place three motion trackers on your bunny's face

You should add as many trackers as you need for a single tracker tool node instance. Usually, you will be tracking more than one subject in a complex VFX shot for film, television, games, or an interactive multimedia production. Figure 13-9 shown the rendering of these multiple trackers, each at the same time, in two of your search regions during the rendering sequence.

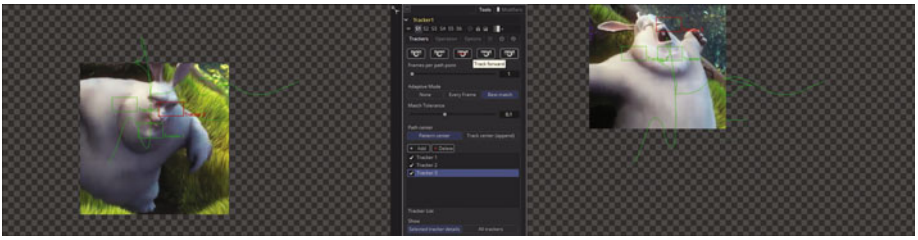

Figure 13-9. Multi-tracker rendering showing two search regions

Next, let's take a look at tracker corner positioning.

Corner Positioning: Add Content to a Motion Track

Let's use a four (corner) tracker scenario next, which might be used to replace a license plate on a car or picture in a frame, which is what we are going to use the tracker for next. Start a new project in Fusion and use **Add Tool ➤ I/O ➤ Loader** to find the **hcw_picture_frame.png** image series, as shown in Figure 13-10.

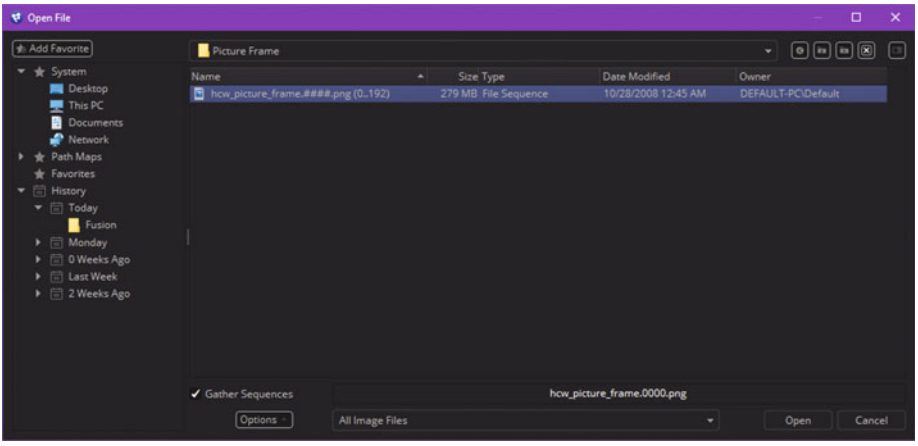

Figure 13-10. Use Add Tool ➤ I/O ➤ Loader and add the hcw_picture_frame.png image series

You can locate these files at the Hollywood Camera Work website at http://www.hollywoodcamerawork.com/trackingplates.html in the Picture Frame section. I cannot host them with the book project, due to legal terms of educational usage, but you can get the PNG series from their site, just make sure to read their legal terms!

Figure 13-11 shows the loader with the 193-frame series, as well as a tracker, which you add using **Add Tool ➤ Tracking ➤ Tracker**. Wire the image series into the tracker node as you did before, and then locate the tracker widget in the upper-left corner, as shown in the right view in Figure 13-11. I am showing a zoomed-in view of the widget in each of these figures so you can see just how precisely I'm positioning the tracker widgets in the corner areas of the picture frame, which we will be motion tracking.

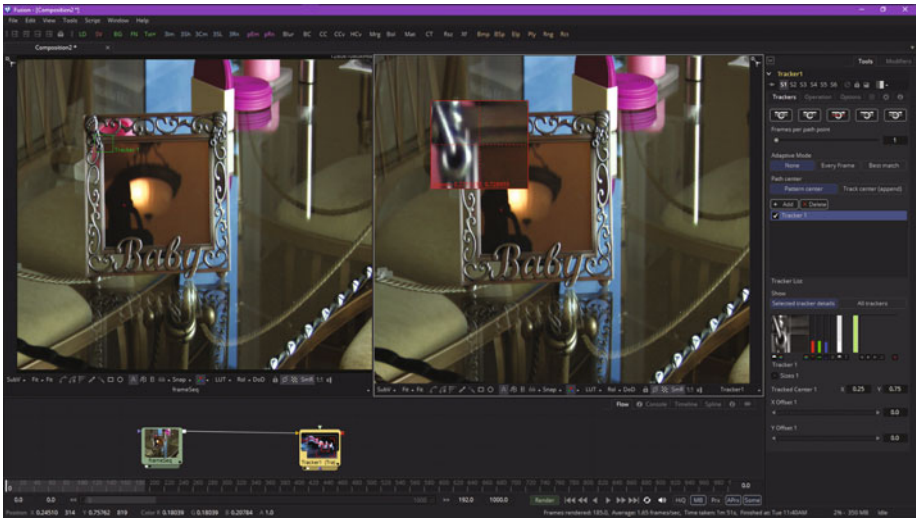

Figure 13-11. *Wire up your nodes and position your first corner*

The next step is to select a tracker node and configure it so that it's using the **Corner Position Operation** (algorithm) and four corner tracking data trackers, both shown highlighted in red on the right side in Figure 13-12.

Figure 13-12. Select the Corner Positioning button

Once you select the Corner Positioning button in the Operation section, Fusion will add three more trackers, and will even connect them with a parallelogram construct, as seen in Figure 13-12, using red wires for the connections and green lines to show the tracker node's data definition widgets.

As you can see, the next step is to position these three trackers in their respective corners so that you show the tracker the inside of the picture frame that you want to motion track. Go ahead and do this now, to get some practice in using the tracker.

I've shown how I positioned center-targeting crosshairs, located in the center of each zoom widget, in the corner areas of the picture frame. Once you do this correctly, all you have to do is select the correct tracking algorithm option, which is not always so easy to do, to get a good motion tracking result. As you can see in Figure 13-13, I'm also using these crosshairs to line up the inside of the picture frame, where picture meets frame, to define where I want the Niki.png model photo placed.

Figure 13-13. Position the other tracker widgets in each corner

The next step is to add in another image loader node for the Niki.png image that you want to track and overlay the frame with, and to add an image saver node to save out an MPEG-4 video file. Use the **Add Tool ➤ I/O ➤ Saver** menu sequence and save the **NikiFrame.mp4** digital video asset, as is shown in Figure 13-14.

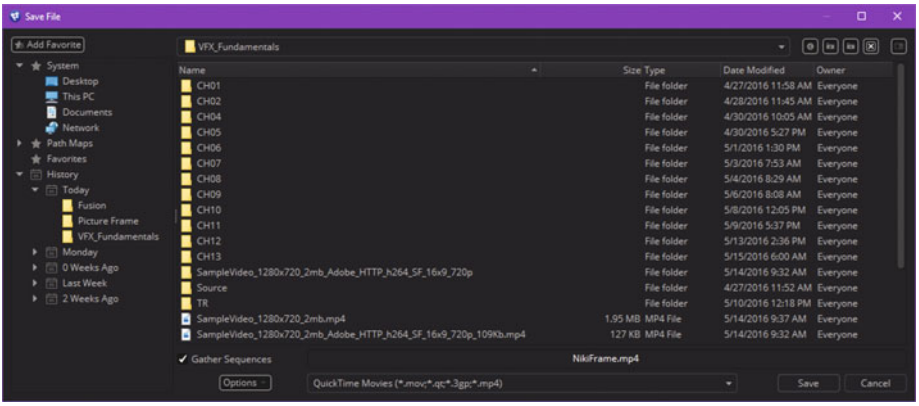

Figure 13-14. *Specify QuickTime movie saver named NikiFrame.MP4*

If you use the Niki.png image for your **foreground input** image (try it in a loader node), you'll get a distorted result. The way I solved this was to open frame 0 of the frame image series and select the inside of the frame, and then I used the pixel dimensions to crop a matching resolution image (NikiFrame.png).

Use **Add Tool ➤ I/O ➤ Loader** to add the **NikiFrame.png** image, as seen in Figure 13-15. You can also see the image saver node and the correct tracker setting (**Best Match**) highlighted in red on the right side of the figure. I originally tried **None**, as is the recommended practice (try this first), but the corners broke away from the frame, so I tried Best Match next, which worked!

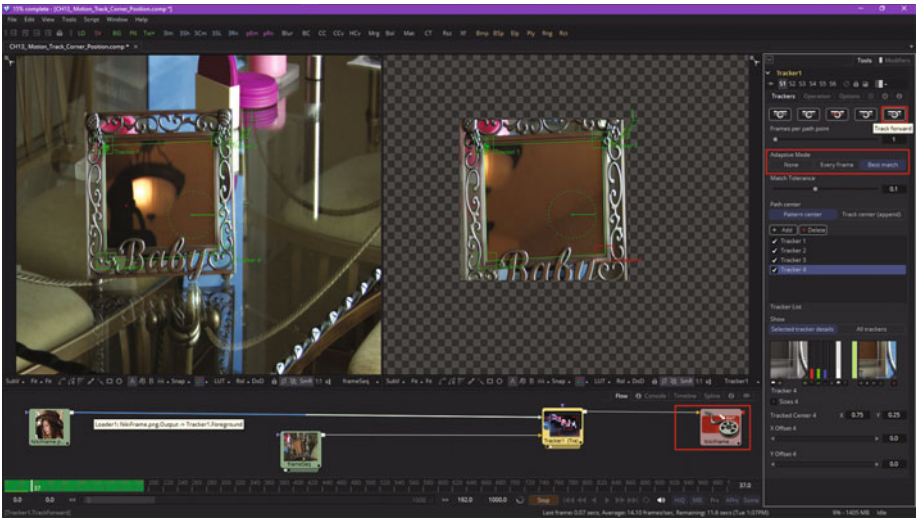

Figure 13-15. Wire the NikiFrame assets and click the Track Forward button

Use the **Track Forward** button to calculate your tracking path, then select the **saver** node tile, as seen in Figure 13-16.

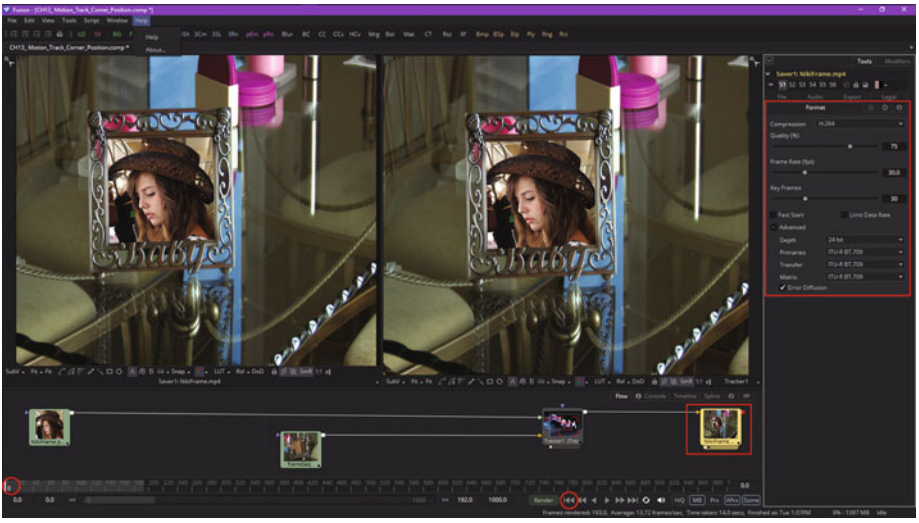

Figure 13-16. Set the MPEG-4 codec settings and reset to frame 0

As you can see in Figure 13-16, I specified MPEG-4 H.264 AVC compression with the default **75% Quality** and **30 FPS** with **30 keyframes**. Since I am a multimedia producer, I specified 24-bit true color. I left the other settings at their defaults, which, as you will see, gave me a very clear result in only 10 MB of data.

As I am ready to render this out to the QuickTime MPEG-4 image saver node, I used the "reset to frame zero" widget, shown circled in red in Figure 13-16, to reset the project views. Then I used the **File ➤ Render Manager** menu sequence to open the **Render Manager** dialog, shown at the top of Figure 13-17.

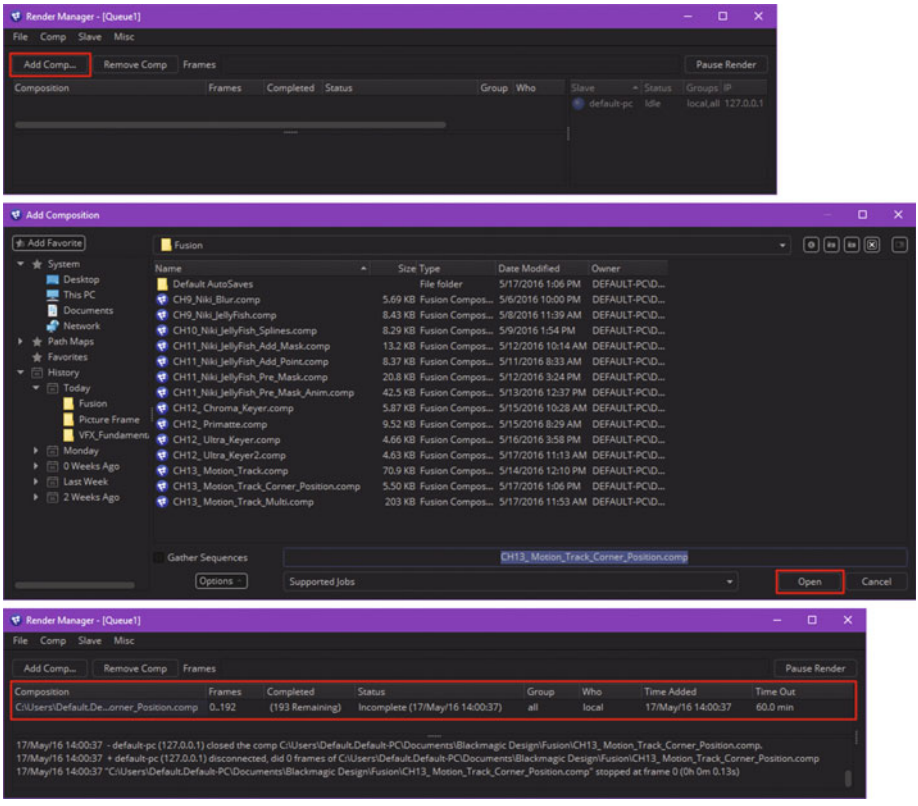

***Figure 13-17.** Using Render Manager and Add Composition dialogs*

Click on the **Add Comp** button, shown circled in red. This will bring up the **Add Composition** dialog, seen in the middle of Figure 13-17. I added the `CH13_Motion_Track_Corner_Position.comp` file to this rendering queue by selecting it and clicking the **Open** button, which adds this current corner-position motion tracking project to the Fusion VFX project(s) rendering queue.

Once I did this, the Render Manager populated with basic information regarding the rendering operation, shown circled in red, along with any technical information such as paths, errors or issues, systems using the rendering queue, and the like. I'm showing you this in case you want to render lots of projects at the same time, or render projects you know can take a long time to render. Fusion Render Manager will render these while you're peacefully sleeping, allowing a VFX workstation that you built during Chapter 1 to make you lots of money.

Next, let's take a look at how to get an image saver node to save the image sequence out to an MPEG-4 H.264 AVC video file by using a **File ➤ Start Render** menu sequence or the green Render button found at the bottom of the time ruler user interface, located at the bottom of Fusion 8.

It is important to note that you can just use the Render button or Start Render menu option for real-time rendering; you do not necessarily need to use your render queue. I was showing you the work process as it is used for advanced (and large) VFX projects.

Once you select your Start Render menu option, the **Render Settings** dialog will open, shown in Figure 13-18 on the right, and you can click the **Start Render** button, shown circled in red. Also shown circled in red in the time ruler is a **Render** button that can be used as a shortcut. Once you start a render, you will see your viewports animate from frame 0 to frame 192.

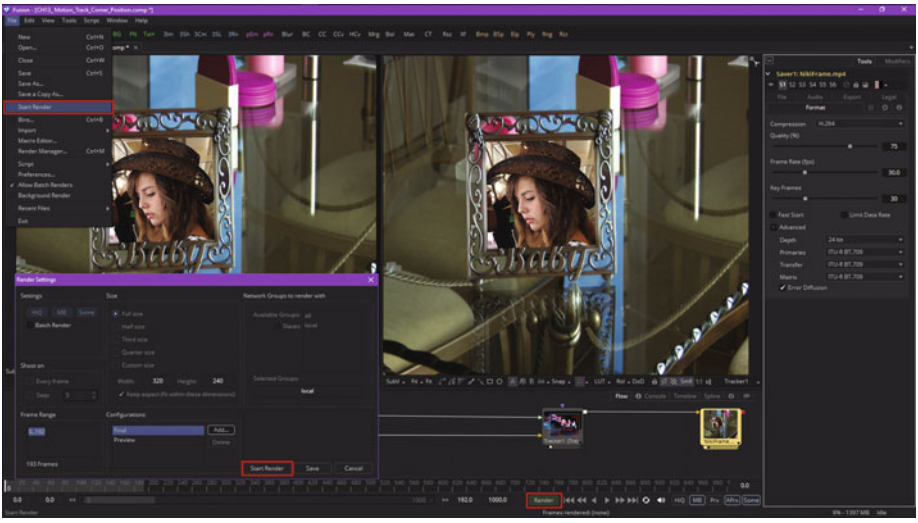

Figure 13-18. Use the Render or Start Render buttons to render

Once you click Start Render, your viewports should start showing you what Fusion is rendering out to the QuickTime codec, as you can see in Figure 13-19. If you want higher quality than the QuickTime MPEG-4 codec can provide, simply export to ProRes, DNxHD, or Sony XDcam, then use Sorenson Squeeze Desktop Pro 11 to optimize your data footprint, by applying codec compression to the project so that you have a manageable digital video asset file size.

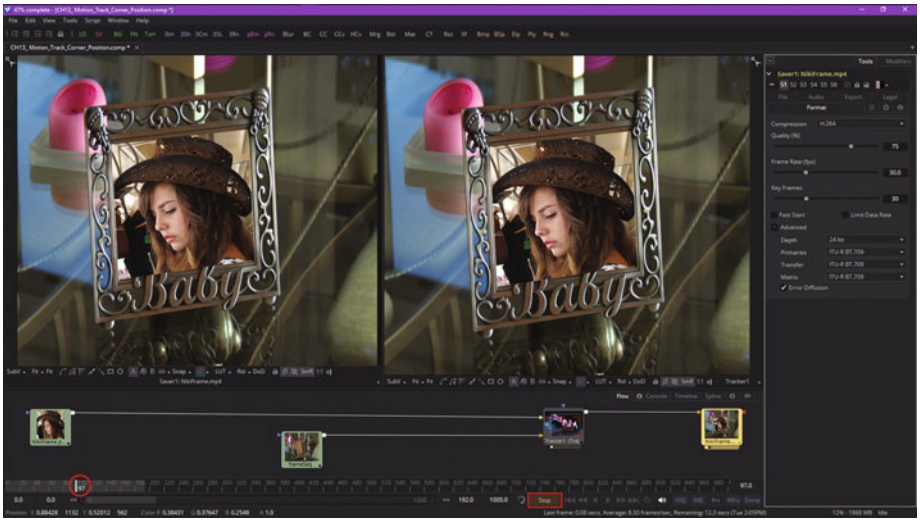

***Figure 13-19.** Fusion 8 viewports show each frame being rendered*

As you can see in Figure 13-20, there is now a NikiFrame MPEG-4 file that is only 10 MB in my `VFX_Fundamentals` folder, and when I play it back I get a smooth, high-quality motion tracking VFX result. As you have seen during this chapter, motion tracking is a complex area in VFX pipeline design. You'll have to work with it for a while in order to become familiar with it due to the number of options available, as well as your having to set the data-gathering widgets to collect the correct pixel data to process.

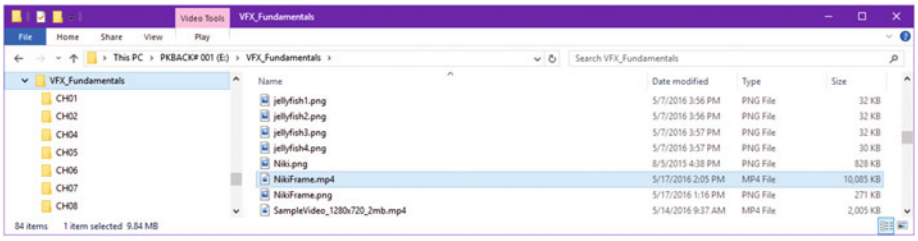

***Figure 13-20.** Your NikiFrame.mp4 file is generated in only 10 MB*

I am trying to get into progressively more complex areas in Fusion as the book proceeds, at least during these final seven chapters, and so now we're going to go into the third dimension and show how Fusion can bridge the realms of 2D and 3D inside the same VFX compositing engine.

With each passing chapter, you will become more and more amazed at what Fusion 8 can do for your VFX project pipeline.

Summary

In this thirteenth chapter, we took a look at motion tracking concepts, principles, and formats that can track objects through your digital image (sequence) assets and digital video assets. We looked at the Fusion tracker node (algorithm) and how it defines what data is examined on a frame-by-frame basis so as to apply pattern recognition algorithms and settings to it. The resulting motion path is used to attach objects to or to guide other algorithms in Fusion as to how to process their objectives. One of these examples used corner positioning to replace an image in an empty frame in a moving shot of that frame. We also looked at how to use an image saver, the Render Queue, and the QuickTime MPEG-4 H.264 AVC export codec to export your project using a digital video file format.

In Chapter 14, you will learn about **3D** concepts and techniques as we take a look at Fusion's 3D capabilities. The rest of this book will incorporate these 3D capabilities as well, so each chapter in the book builds upon the previous one.

Chapter 14

VFX Pipeline 3D Production: Compositing 3D Assets

Now that you have an understanding of some of the advanced 2D features in Fusion 8, let's spend the rest of the book using Fusion in its entirety, which includes using the **3D compositor** and its features. In this chapter, we will look at how Fusion differentiates 2D and 3D, how it bridges them together seamlessly, and how to add new 3D elements into your VFX project pipeline. With this knowledge we will be able to incorporate 3D elements and assets for the rest of the book (the advanced topics chapters).

Advanced Compositing: Fuse 2D with 3D

Fusion continues to add new features that make it worthy of its powerful name, fusing traditional 2D image-based compositing and 2D mask vectors with rapidly emerging 3D vector geometry-based modeling and animation. As you know from the first section of this book, 2D image layers only have infinitely flat X and Y dimensions, which does not match up with the reality of film, TV, or 3D game production, where content is either captured with a live-action camera in a 3D environment or has been created in 3D modeling, surfacing, animation, effects, and rendering applications.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_14

Fusion 6 added an OpenGL GPU hardware–accelerated, 3D compositing environment to the flow node editor, adding support for geometry, point clouds, and particle systems. This opened up the software to new features such as taking 2D images into 3D space and importing cameras, lights, and textures from 3D applications such as Autodesk 3D Studio Max by using the FBX file format.

Fusion continues to add i3D features, enabling realistic surfaces by using illumination algorithms and shader compositor tools and workflows. You can render 3D using realistic depth-of-field settings, and you can use motion blur and super sampling. You can extrude 2D text into 3D text and bevel it, and you can cast shadows across 3D geometry using VFX pipeline elements.

Fusion 3D Scenes: 3D Tools for 3D Environments

Fusion has **3D tools** that allow you to add 3D compositing, enabling you to create an environment supporting your 3D assets while also maintaining your 2D compositing pipeline containing digital imagery, digital video, digital illustration, or digital painting assets. In Fusion, if you want to "fuse" 3D with your previously 2D VFX compositing pipeline, you would use the **Add Tool ➤ 3D** menu and its related **Material**, **Texture**, and **Light** sub-menu structures, shown in Figure 14-1. Also shown are the **FBX mesh 3D** node tool and a **Gargoyle.FBX** file imported from Autodesk 3D Studio Max, just to show you what is possible.

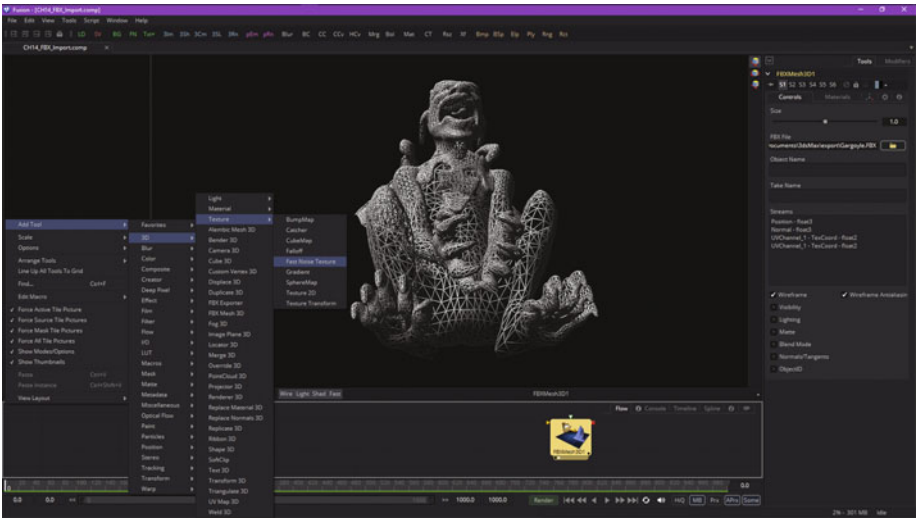

Figure 14-1. *Fusion adds 3D compositing features via the 3D menu and its sub-menus*

These 3D tools will allow you to do things like create “primitive” geometry, such as a flat plane, cube, sphere, torus, cylinder, or cone, or import complex geometry (mesh) objects, as I did to create Figure 14-1. You can create lights and cameras, point clouds, projectors, text, and fog, and warp, deform, bend, duplicate, displace, weld, transform, and UVW map 3D geometry.

When you create a Fusion composition, you do not need to define whether it will be 2D or 3D, because you will use nodes or tools to seamlessly combine your 2D and 3D scenes. As you will see in the final two chapters, particle systems do need to be defined as 2D or 3D, however.

As you will see in Chapter 15, 3D geometry will be added to your VFX pipeline using primitives (3D shapes), image planes, and FBX mesh 3D assets imported using an FBX mesh 3D node tool.

3D geometry is combined with lights and cameras by using the **merge 3D** tool, and is rendered to 2D images using the **renderer 3D** tool.

As you will see in Chapter 15, the surface appearance of your geometry can be defined using material tools, such as your Blinn and Phong tools. Materials can use textures that apply 2D images or 3D environment maps using UVW mapping coordinates.

Whenever these tool nodes have their viewports selected, a **3D viewer** will replace the 2D viewer we have been using thus far in the book. This allows you to navigate and manipulate your VFX pipeline using 3D. The interactive 3D viewer is highly dependent on the computer graphics hardware, relying on support for OpenGL, which is why we got into GPU hardware in Chapter 1. The amount of DDR5 memory, as well as the number of processors, processor speed, cache, and features of the graphics processing unit (GPU) will define the smoothness of operation and quality of the rendered visuals that will appear in your i3D viewer.

Bridge 3D and 2D VFX Projects: Merge and Render

As you probably have surmised, the tool nodes that output i3D scene data can’t be simply connected to 2D node inputs that expect 2D image data. As you have learned in this book already, **vector** (math) data needs to be **rendered** to **raster** (pixel) data.

For example, data that is output by the **FBX mesh 3D** node tool cannot be connected directly to the input of a (2D pixel-based) blur algorithm that is not expecting geometry data, UVW mapping data, and surface, material, and texture data. To hand the 3D object shown in Figure 14-1 to a 2D node tool, you would need to use a camera 3D node to photograph a 3D scene, a merge 3D node to connect the camera and the mesh together in a scene, and a renderer 3D Node to render your 3D scene into 2D pixels, which would then be input into the blur node tool’s input for processing by its algorithm.

We will be looking at the 3D content production pipeline during this chapter, as well as during the next several chapters covering 3D.

Basic 3D Scene: Shapes, Camera, Light, and Merge

There are a minimum of **three nodes** that are needed in order to create a 3D scene compositing pipeline. One is the merge 3D node, which serves as a **SceneGraph** of sorts by collecting all of your 3D scene components together. Into this merge 3D node plugs the other 3D nodes, one of which must be the camera 3D node, which is used to photograph, video, or "render" the other required node, which would be the 3D asset itself. This would include the FBX mesh 3D node, one of the Fusion 8.0 primitives (accessed using the shape 3D node), the Alembic mesh 3D node, the point cloud 3D node, the fog 3D node, the ribbon 3D node, the text 3D node, or the image plane 3D node. I am also covering the light nodes here, since they're needed to provide a realistic 3D depth appearance.

Shape 3D: Using Your Seven Fusion Algorithmic Primitives

Start a new Fusion composition, right-click and **Add Tool ➤ 3D ➤ Shape 3D** to add the node shown on the left in Figure 14-2. Also seen is the next **Add Node ➤ 3D ➤ Camera 3D** needed for the other required node, which you will be adding next. As you can see on the right in the Control Panel, there are seven algorithmic 3D "primitives" that you can select. I chose the **icosahedron** for this example, as it looks the coolest as a wireframe mesh. To see a wireframe representation of a primitive, simply select the wireframe option! The primitive is a 3D object defined using mathematics—an equation, more precisely—and not using vertices, edges, and faces, which is why this Shape 3D node is an algorithmic 3D asset creation tool.

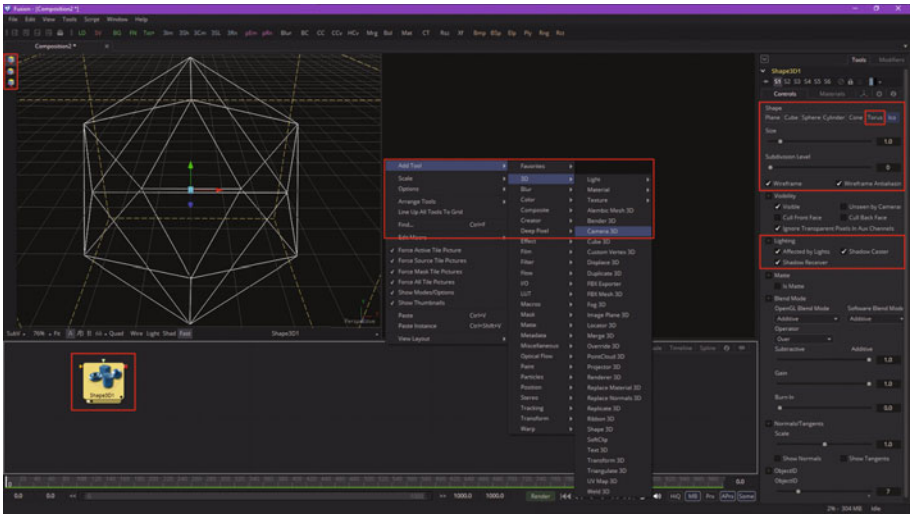

Figure 14-2. Add shape 3D and camera 3D nodes to a new project

Once you have the shape 3D node in the flow node editor, either drag it into View1 to set that view to that node (a cool trick) or click the left dot in the bottom of the tile as shown in Figure 14-2 circled in red. You can **move**, **rotate**, and **scale** this object using the three icons on the top left of the view, or use the **Q-W-E** keyboard shortcuts. Figure 14-2 shows your move widget in the viewport; click and drag on the colored arrowheads to move the object in that particular direction (X, Y, or Z) in 3D space.

I also circled the **torus** option in red, as we'll be using this later on when we cover lighting. A torus, which is like a doughnut, gives us a far more complex inside and outside surface to show off the lighting effects of something like a spot light.

Next, let's take a look at the camera 3D node and Fusion camera system, as nothing in a 3D scene will render without using a camera!

Camera 3D: Using a Camera to Photograph the 3D Scene

One of the challenges for 2D and 3D compositors is matching the cameras used in live-action shots to the cameras used in the 3D content that is being seamlessly composited into those scenes. To allow you to succeed at these challenges, Fusion provides a flexible virtual 3D camera functionality where you'll configure common camera controls such as the angle of view, focal length, plane of focus (depth of field), aperture, and clipping planes. These are shown in Figure 14-3 and allow you to set up your own virtual camera

that matches the camera you are using for your live-action shots. You can also import cameras from 3D software. Figure 14-3 shows a **camera 3D** node and control panel.

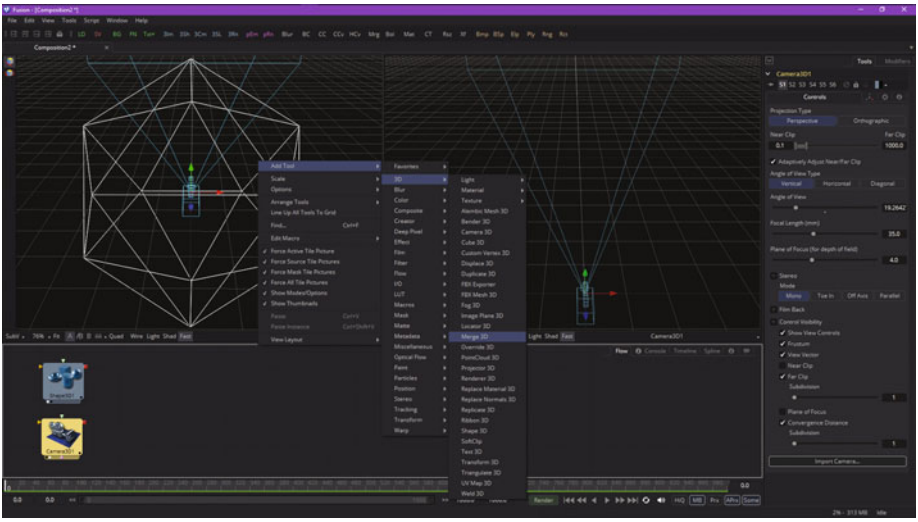

Figure 14-3. *Select camera 3D node to see control panel options*

When adding a camera 3D node, you must connect it to your merge 3D node in order to view those 3D elements also connected to that merge 3D node, which we will cover in the next section. This node is added using **Add Tool ➤ 3D ➤ Merge 3D**, as is seen in Figure 14-3 along with a selected camera node and its options.

To view a 3D scene through a camera, select the merge 3D node that the camera is connected to, or any tool downstream of the merge 3D node. You can also drag that selected merge 3D node, or any downstream node (tool), into a viewer. Alternatively, you could use the view-select dots at the bottom of the tool node tile.

If you have more than one camera, you can right-click on the axis label in the bottom corner of the view and choose the camera name that you want to view.

Next, let's take a look at a merge 3D node tool tile and how it allows us to build a scene.

Merge 3D: Assembling the 3D Scene Components

You'll notice that the **merge 3D** node has only one option, which we will select, to pass through the lighting information, which is shown highlighted in red on the right side in Figure 14-4. I am also showing the **Add Tool ➤ 3D ➤ Renderer 3D** menu sequence to save on the number of figures used. These are the four nodes that are a minimum requirement for creating a 3D Fusion scene.

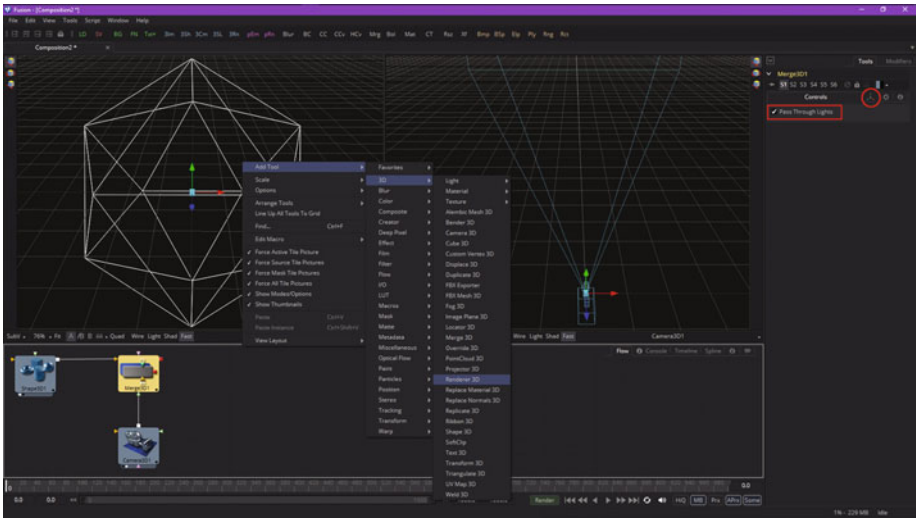

Figure 14-4. A merge 3D node aggregates all the other 3D nodes

Every 3D tool in Fusion will output a complete 3D scene, so that the 2D/3D VFX compositor has the maximum flexibility in creating any visual effect. This is why your merge 3D node tool has this name, as it is merging anything connected to it into the same 3D scene, kind of like a 3D consolidator.

Unlike "traditional" 3D modeling and animation software, where 3D objects occupy the same (global) 3D scene environment, Fusion 3D scenes are created using a camera 3D tool and Merge 3D tool in conjunction with each other.

A merge 3D tool makes this possible by taking attributes of the 3D tools plugged into it and combining them into a single 3D scene data–processing pipeline. Unlike a 2D merge tool, your Z-order of 3D objects in a scene is not restricted to just your background and foreground inputs.

Your merge 3D node allows an unlimited number of 3D data inputs, combining them according to their absolute positions in 3D space. A merge 3D node includes a **unified 3D transformation**, which is accessed via a tab seen circled in red in Figure 14-4.

Your unified 3D transformation parameters should be used to adjust the position, scale, and rotation of all the elements that you have combined using that particular merge 3D node.

Therefore, it could be considered to be a 3D scene transform utility panel! A global transformation algorithm will not only be applied to 3D geometry, but also to lighting, effects, and 3D particles. All transformations will take place around the scene (environment boundary) pivot point. This pivot point forms the basis for relative 3D object parenting in your 3D environments.

Next, let's take a look at how to convert your merge 3D scene data into 2D data that can be processed by Fusion's other VFX pipeline processing tools, many of which I covered earlier.

Renderer 3D: Converting the 3D Data for Use in 2D Compositing

Once you've added the renderer 3D node to your basic 3D Fusion scene and wire the output from the merge 3D node into it, you will get a white (flat or "blown out") representation of the 3D ISO object when you select View2 (white dot) at the bottom of the tile. This is shown circled in red in Figure 14-5, along with the next **Add Tool ➤ 3D ➤ Light ➤ Spot Light** context menu and the **"Enable Lighting"** checkbox to enable the lighting, which we will be adding next to get more realistic 3D render results. When you click the "Enable Lighting" option, the white in View2 will turn black, because there is not any light added to the 3D scene yet. Let's look at how to add light to the 3D scene next.

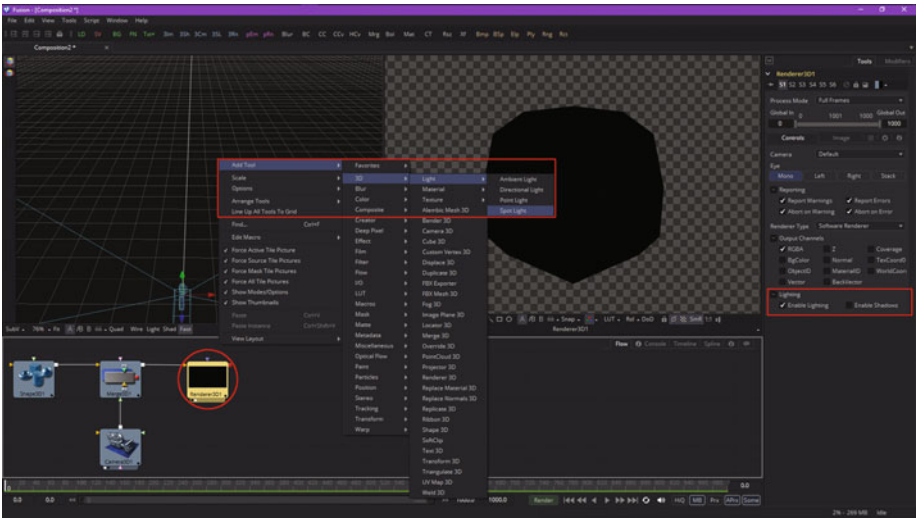

Figure 14-5. *The renderer 3D tool converts 3D data into 2D data*

Adding Lighting: Add Spot Light Node to Give 3D Primitives Depth

Select the **spot light** node, as shown in Figure 14-6 at the left bottom circled in red, and set the spot light color to a purple-pink hue, as shown at the top of the control panel. Play around with the **decay type**, **decay rate**, **cone angle**, **penumbra angle**, and **dropoff** setting to get a feel for how the spot light tool works. I changed the 3D primitive I was using to the **torus primitive** to get a better lighting result, as you can see in View2 in Figure 14-6.

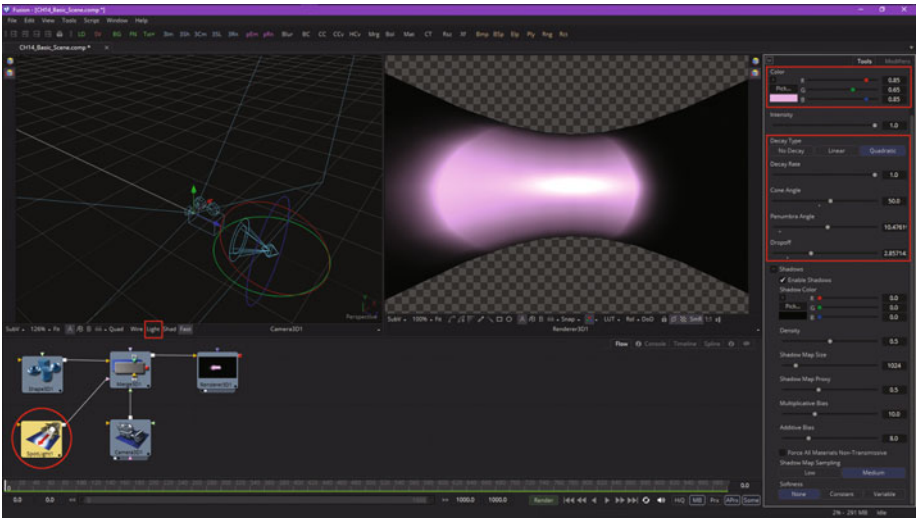

Figure 14-6. Add a Spot Light node and wire it into the Merge 3D node

You can add multiple light sources to a merge 3D (scene) to build highly detailed lighting environments. As you will see in Figure 14-7, I have added a **directional light node** and also turned on the **wire mode** and **light mode**, as seen circled in red.

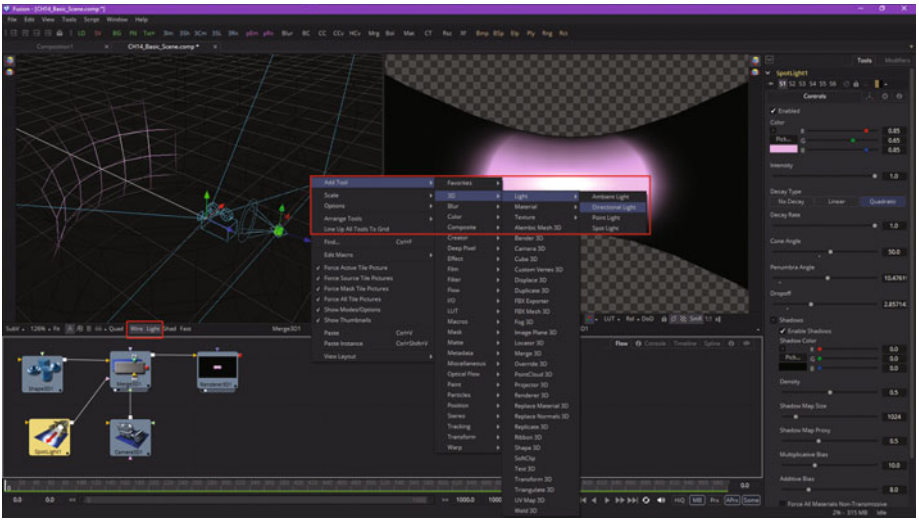

Figure 14-7. Add a directional light, turn wire and light modes on

Your context-menu sequence to add this node tool tile is **Add Tool ➤ 3D ➤ Light ➤ Directional Light**, so add that to your basic 3D scene project to see how this light works.

There are four different types of lights you may use for your 3D scenes, including **ambient light** (global illumination), **directional light** (along one axis), **point light** (like a lightbulb), and **spot light** (like a flashlight). The default light will be a directional light, unless you add your own custom lighting. The lighting won't be visible in a 3D viewer unless a **Light** button is enabled in the viewer toolbar. Light would not be visible in a renderer 3D node until an "Enable Lighting" checkbox is checked in the control panel, as seen highlighted in Figure 14-5. Tools that create or load geometry include similar lighting options.

A viewer toolbar Light button can be used to turn lights off or on. If lights are disabled in a viewer or final render, then your 3D imagery will be **evenly lit** using **ambient light**. As you have seen, this could make 3D geometry appear flat and less realistic, which is why we're covering custom lights in detail!

You can use ambient light to set minimum levels of light for your merge 3D scene, because ambient lights create general, or uniform, illumination for any given 3D scene. Ambient light simply exists everywhere, without appearing to emanate from any particular light source, and has no "real-world" equivalent; it is purely a 3D phenomenon. Ambient light can't cast shadows, and it will fill in the shadow areas within a scene the brighter it is set. I tend to either not use ambient light, or keep it low.

Directional lights, on the other hand, use parallel rays of light. These illuminate the entire scene from one direction, creating a plane of light travelling along one axis. The sun is one example of a directional light source, since it is far away (it is actually a point light source, which we will cover next). The moon is a better example, as it reflects the sun, which "parallelizes" the sun's rays, making the photons of light travel parallel to each other. Want proof of this? Look at your shadow during the day (blurry edges) and then look at it during a full Moon (crisp, razor-sharp edges), when the Moon is serving as a large reflector, and parallelizing the Sun's rays. Things like this are why we want to control shadow softness, which we will cover a bit later on. Figure 14-8 shows a torus in wireframe mode, with a directional light moved above it and rotated 90° so it shines down on the shape.

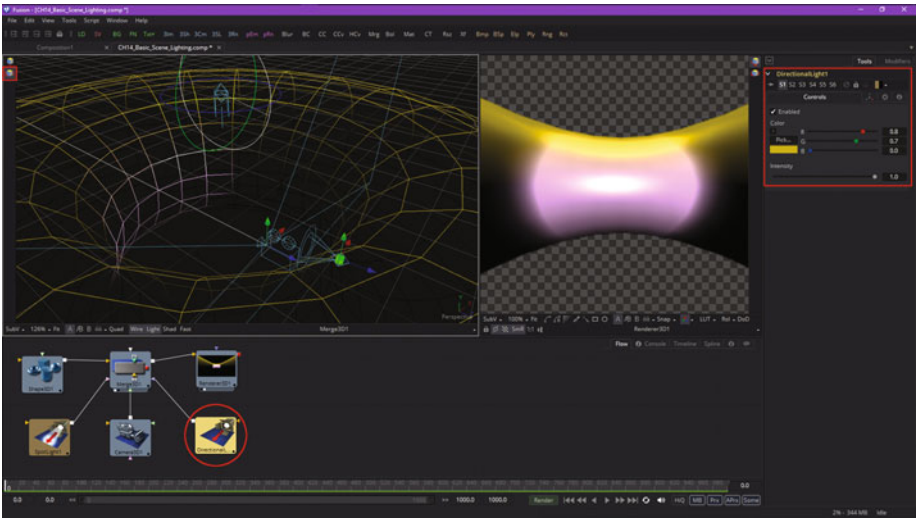

Figure 14-8. Move directional light above torus and rotate 90°

I colored this directional light **gold** so that you can see the difference between the spot light and the directional light. If you compare Figures 14-7 and 14-8 you can see that the directional light illuminates more of the torus wireframe mesh.

Continuing on, a **point light** is a light source that will emanate from an infinitely small point in 3D space. The example of a point light from your everyday life would be a light bulb.

A **spot light** is a more complicated point light that has settings that allow you to produce a well-defined **cone of light** featuring a **brightness falloff** from the middle of the light to the edges. An example of a spot light would be a flashlight. The spot light in Fusion is the only light that can produce

shadow results. In other 3D software, point lights will also create shadows, as will directional lights in many 3D applications.

A Fusion 3D tool that generates or imports geometry has additional lighting options based on unique attributes for that tool. These lighting options will be used to determine how that 3D object will be rendered using lights and shadows in a scene. For instance, if the **"Affected by Lights"** checkbox is active, the lights in the 3D scene can affect the 3D geometry in the scene. If the **"Shadow Caster"** checkbox is active, an object will cast shadows on other objects in the 3D scene. If **"Shadow Receiver"** is active, an object should be able to receive shadows cast by shadow caster enabled 3D objects. These options are highlighted in red in Figure 14-2 in the control panel for the shape 3D node tool in the **Lighting** section, which I opened and highlighted in anticipation of this topic. Similar lighting control options are available in other 3D tools.

Advanced Lighting: Shadows and Shadow Maps

Let's take a look at **shadows** in Fusion, and along the way I will show you some of the cool user interface options in the 3D compositor part of Fusion. The first thing we're going to do is to add an image plane 3D node using **Add Tool ➤ 3D ➤ Image Plane 3D**, as shown in Figure 14-9. We will select the spot light node and adjust the parameters to get rid of the white "hot spot." Also, we will make sure that the **"Enable Shadows"** option is selected, as shown in Figure 14-9 on the right inside of the red box highlighting.

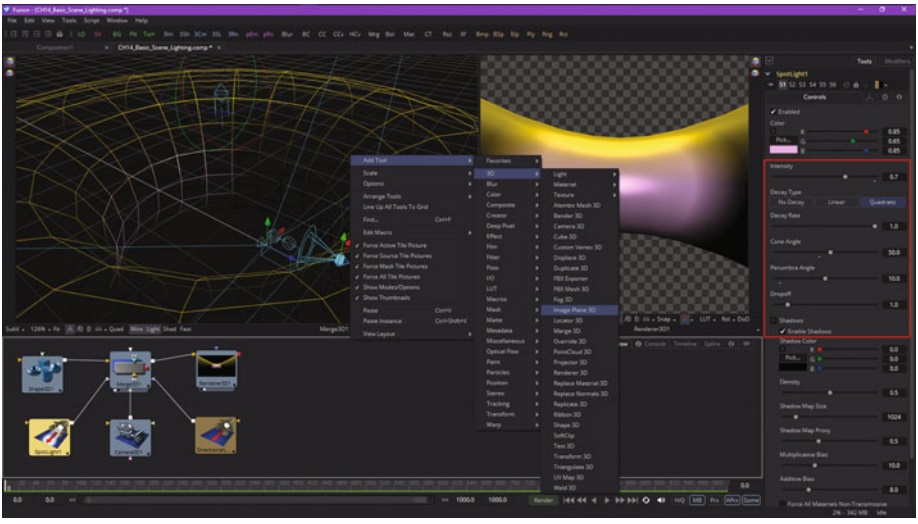

Figure 14-9. Add an image plane 3D Node and adjust a spot light

Now the torus mesh wireframe has a nice pink (spot light) and gold (directional light) coloring, showing you where those lights are hitting the object. This is a great technique (colored lights, at first) to use to show you where lights are affecting your 3D geometry, before your materials and textures are added, at which you would most likely switch light colors back to white.

Next, let's wire the image plane 3D into the merge 3D so it becomes part of this 3D scene composition, as can be seen in Figure 14-10. Once it is connected, you will see the image plane 3D mesh in the middle of the torus, which you will move later. The image plane 3D control panel settings that lock your aspect ratio (Lock Width/Height) tell how many squares will be in the mesh (10). You can also see a **"Wireframe"** option you do not need, as the View1 **"Wire"** option is enabled. Also, open up the **Lighting** section and make sure that all the lighting and shadowing options have been selected. If there's no geometry behind the image plane, you can turn the **"Shadow Caster"** option **off** to save on rendering processing overhead.

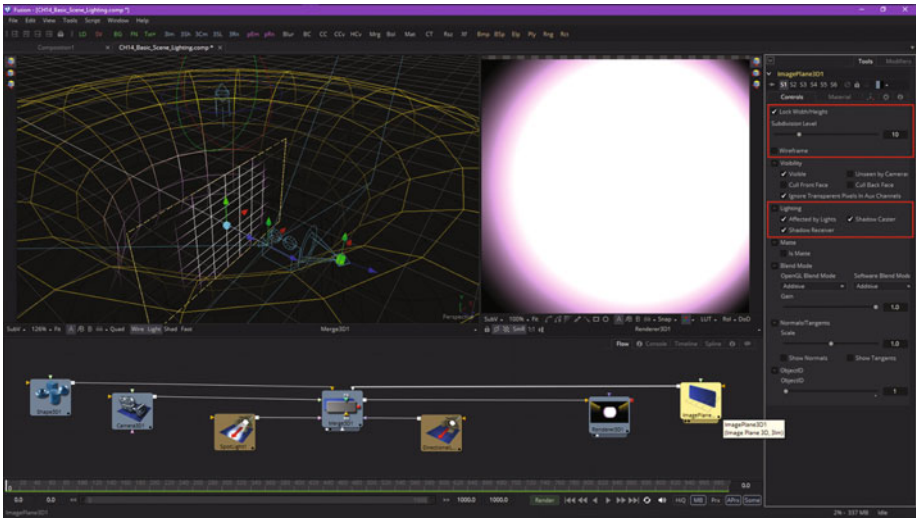

Figure 14-10. *Wire the image plane 3D node into a merge 3D node*

Next, select the **move** icon and use the blue z-axis arrow head to move the image plane outside of the center of the torus, as shown in Figure 14-11. The widget axis you are using will turn white to show that it is in use. The move icon is seen in red in the upper-left corner, and the image plane is now inside of the torus, as you can see in your renderer 3D node View2.

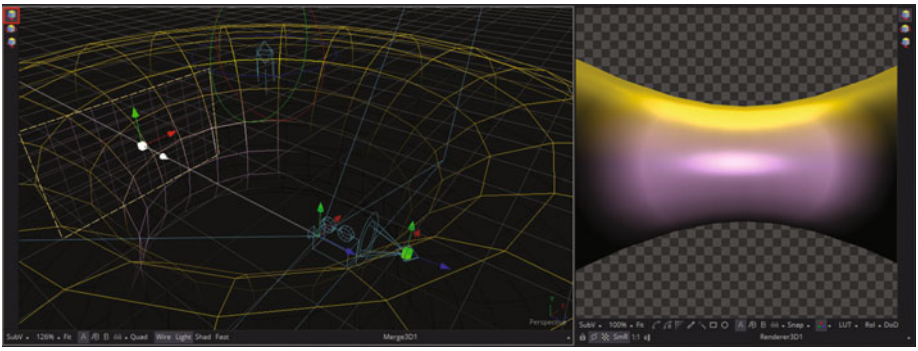

Figure 14-11. Use a move widget to move the plane on the z-axis

We need to move the image plane further, so let's switch into **Quad View** mode by clicking the **Quad** button shown in Figure 14-12. Each of the four views supports its own wire and light mode settings, which is why the active right view has **wire** selected.

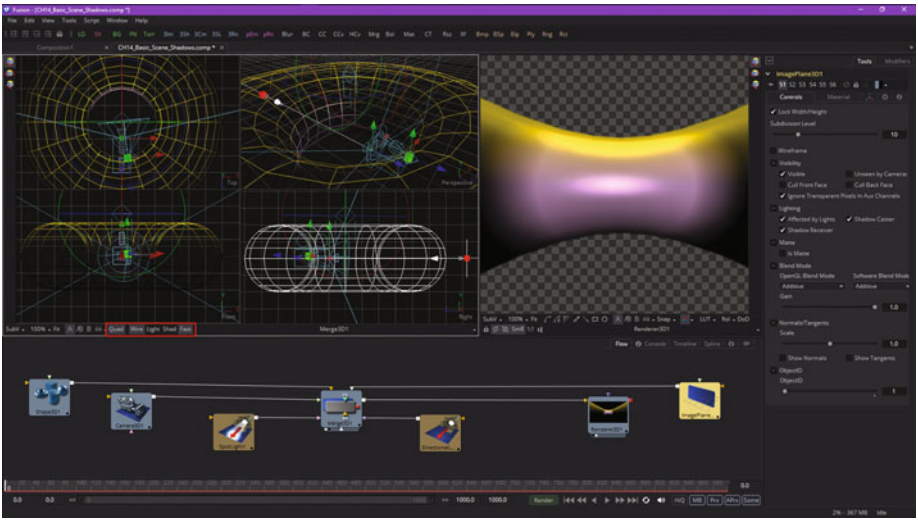

Figure 14-12. Turn on Quad View, and move the plane out of the inside of the torus

Now I can see the torus and plane in white. I use a **move** widget to put the plane outside of the torus, then switched to the **Front** view, as seen in Figure 14-13, so I could scale the plane up using the **scale** slider in the settings shown circled in red. **"Lock XYZ"** is another way of saying **"Lock Object 3D Aspect Ratio**."

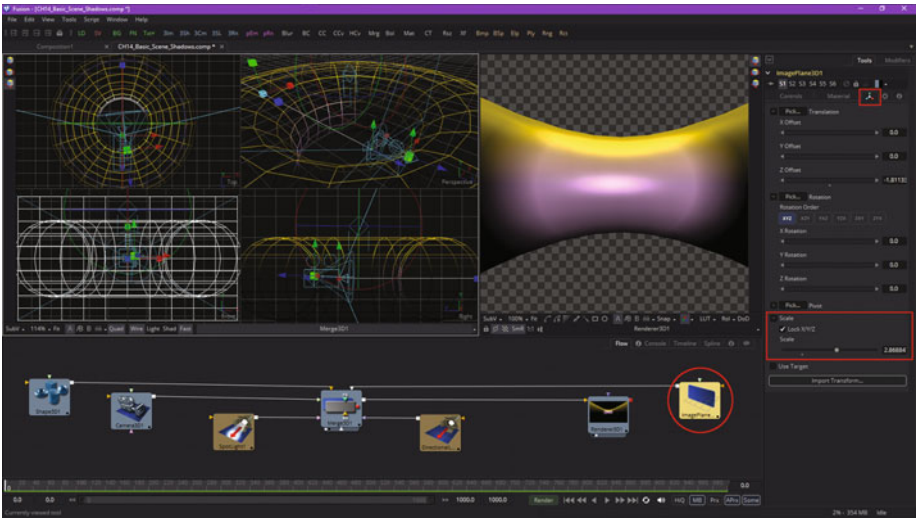

Figure 14-13. Move a scale slider in the image plane control panel

Now you can see the plane behind the torus in **View2**, as in Figure 14-14. I selected the **Material** tab, seen on the right in red, and selected the **"Receives Lighting"** and **"Receives Shadows"** checkboxes, as well as made sure the plane used a white color.

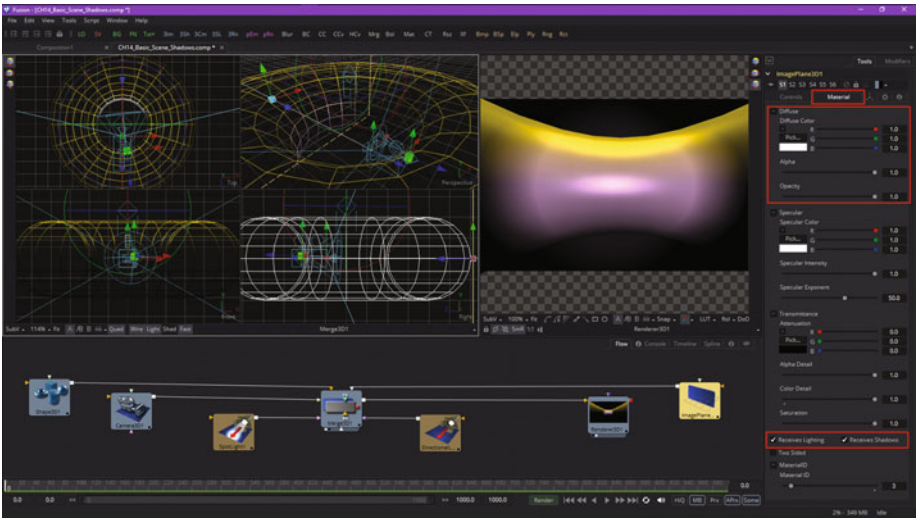

Figure 14-14. Use the Material tab to enable light and shadows

Next, turn off Quad View mode. Let's look at how you can selectively choose the different views in a single view mode by using the right-click context menu and then selecting **Camera ➤ Right** to set the right view. This way, we can start to position the camera to look down on the plane in order to see the shadowing effect.

Select the camera and the **move** icon, highlighted in red in Figure 14-15, and move your camera **above** the torus, as shown in the middle. The new renderer 3D View2 can be seen on the right; it shows the new positioning for the camera.

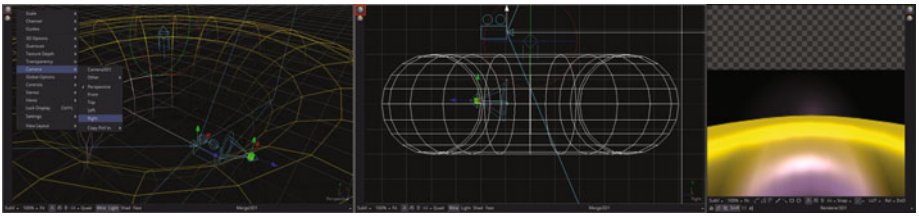

Figure 14-15. *Use move icon with y-axis arrow to move camera up*

Next, select a **rotate** icon, highlighted in red in Figure 14-16, and use an **X-rotation ring** to rotate the camera **forward**.

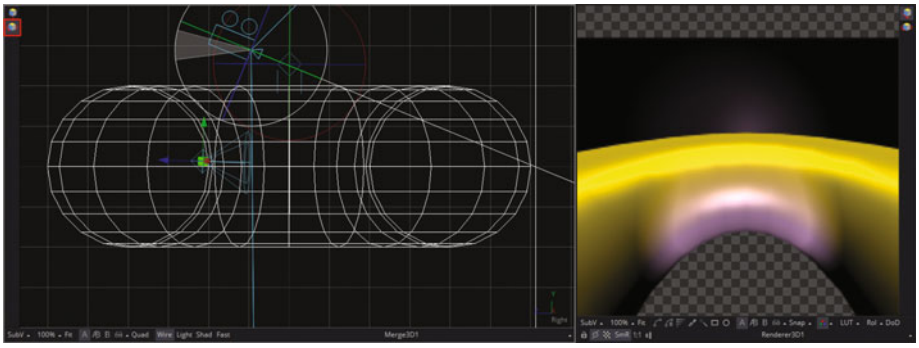

Figure 14-16. *Use rotate's X-rotation ring to point the camera down*

Although this fat torus and plane are fine for showing a lighting simulation, we really need a thinner torus that is **75°** from where it is now, or nearly parallel to the plane, so it'll cast a nice oval shadow, as seen in View2 in Figure 14-17. Move your spot light up and your camera down, as shown in Figure 14-17, and rotate them so that they will cast a nice shadow. Set your torus characteristics to **radius 0.6**, **section 0.1**, a **subdivision base**

of **32**, and a **subdivision height** of **24**. I also colored the torus **army green** to make it stand out a bit better. Finally, I selected the renderer 3D node tile and made sure the “Enable Lighting” and “Enable Shadows” options had been selected.

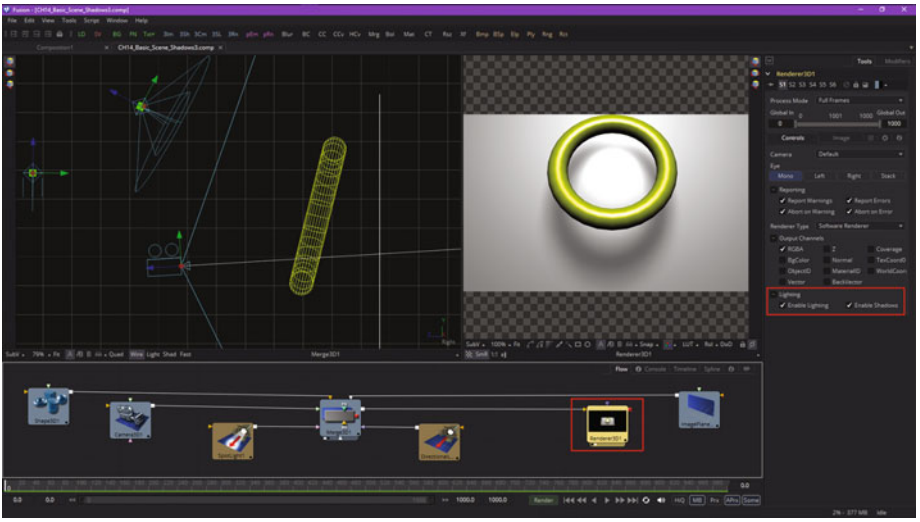

Figure 14-17. Rotate and scale the torus to better cast shadows

Next, I zoomed in **200 percent** on the renderer 3D **View2** so I could see the shadows better and then centered the view using middle mouse button panning. I also used **left + middle** button zooming to zoom into my merge 3D **View1** so you can see the camera and spot light positioning more clearly and mimic them in your own Fusion project.

As you’ll see at the top of the spot light control panel in Figure 14-18, I have increased the **intensity** of the spot light to 100 percent, selected the **No Decay** algorithm, set **Dropoff** to zero, and maxed out the **cone angle** and **penumbra angle** to get solid white lighting on the image plane 3D object so as to better show the shadow effect and to more clearly show the different shadow settings.

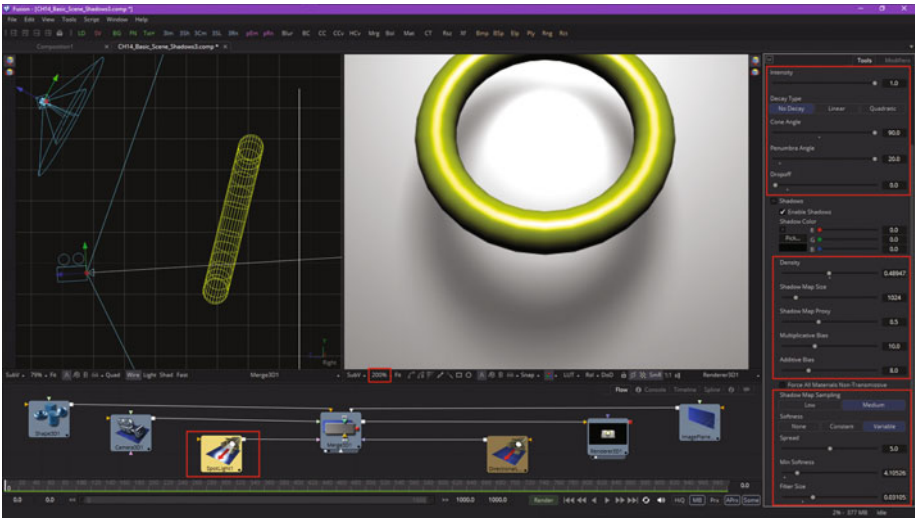

Figure 14-18. Pan and zoom renderer 3D view 200 percent to see shadows

As you can see, I used the default settings for the shadow effect shown in Figure 14-18, which gives a realistic result. I selected the "Variable Softness" option to expose the controls for spread, minimum softness, and filter size, as these are advanced settings that you will need to know about to create realistic VFX pipelines and believable 3D compositing results. Let's look at key shadow parameter options in detail to see what they do.

The shadow map parameters are seen in the middle section highlighted in red in Figure 14-18. **Shadow density** controls how dark the shadow is, as shown at 75 percent (0.75) in Figure 14-19.

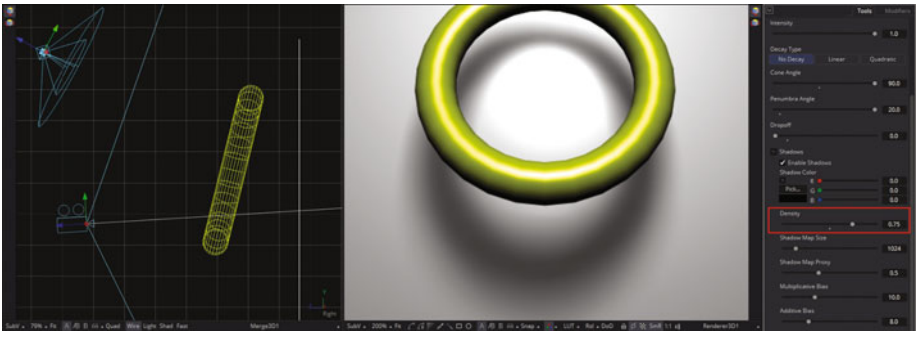

Figure 14-19. Set shadow density to 0.75 to get darker shadows

This control affects all of the other shadow settings—a **global** parameter, if you will. The other parameters affect only particular attributes (characteristics) for shadow properties, so these are more local, or **fine-tuning**, parameters, if you will. Let's take a look at them next, as their effect is less evident.

A **shadow map** is a type of depth map that specifies each pixel's depth in a scene. The depth information is used to create a shadow layer using the spot light settings. The shadow quality is inversely related to the size of the shadow map. Larger maps would generate higher-quality shadows, taking longer to render. Let's reduce the **shadow map size** 200 percent to **338** and take a look at the decrease in quality in the renderer 3D View1, as is seen in Figure 14-20. As you can see, the quality is literally terrible!

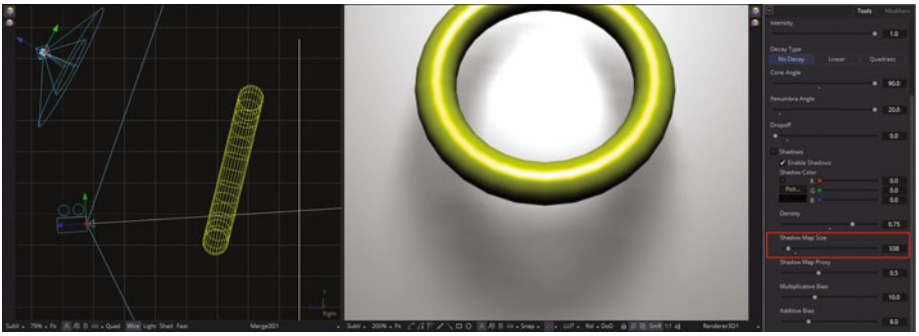

Figure 14-20. *Reduce shadow quality; set shadow map size to 338*

The wider the cone of the spot light, or the more falloff you set for the cone, the larger your shadow map needs to be to obtain high-quality results. As with anything, there is a point of diminishing return, where increasing shadow map size doesn't improve visual quality for a shadow. Do not set shadow map size to be any larger than it needs to be as a processing optimization. A **shadow map proxy** control can be used to set the percentage that a shadow map is scaled for fast interactive previews, but it does not affect quality, as you can see in Figure 14-21, where it is set at 100 percent.

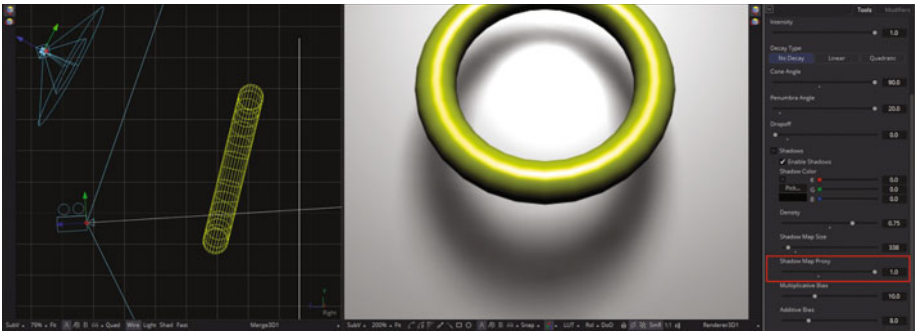

Figure 14-21. Shadow map proxy doesn't affect quality (it darkens shadow)

Multiplicative and **additive bias** work by adding a small **depth offset**. This moves your shadow away from the surface that it's shadowing, eliminating something called *Z-fighting*. When small bias values are used, as shown on the left side in Figure 14-22, 3D objects will "self-shadow" themselves. When high bias values are used, shadows become separated from the 3D surfaces.

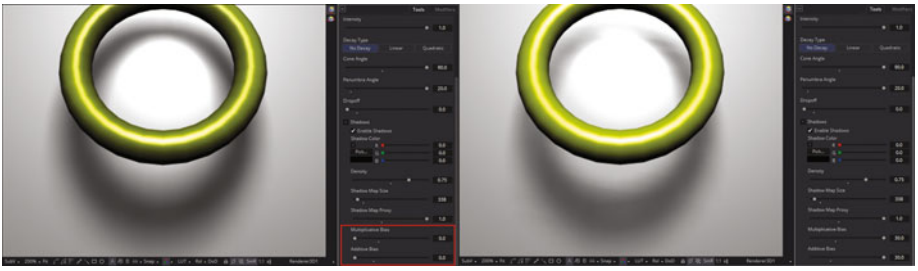

Figure 14-22. A range for multiplicative bias and additive bias

Shadows can be viewed as textures applied to 3D objects in a scene. This can result in Z-fighting, which results when a portion of a 3D object that should be receiving shadows renders on top of that shadow. When this happens, adjust the **multiplicative bias** until the majority of this Z-fighting is eliminated, then adjust the **additive bias** slider in order to eliminate the rest.

The spot light includes a **softness** feature, with options of **Constant**, **Variable**, or **None**, as seen in the bottom red box in Figure 14-18. Hard-edged shadows, set using None, render significantly faster than the other, softer shadow options. Hard-edged shadows would be jagged or aliased, unless your shadow map size is large enough. Shadow softness can be used to hide aliasing, rather than using large shadow maps to preserve memory and optimize for the GPU.

The **Constant** softness option will generate shadows where edge softness will be uniform across a shadow regardless of the shadow distance from the 3D geometry that is casting that shadow.

The **Variable** softness option will generate shadows that get softer the farther they get from 3D geometry that's casting the shadow. This is more realistic for visual effects; however, your shadow characteristics are more difficult to specify. When the Variable option is used, three additional controls, seen in Figure 14-18, allow for adjustment of **spread** (falloff), **minimum softness**, and **filter size**.

Increasing the filter size increases the maximum allowed softness for that shadow. Decreasing the filter size reduces render time, and might limit the softness of the shadow or potentially even clip the shadow. This value is a percentage of shadow map size.

A **"Force All Materials Non-Transmissive"** checkbox controls how light should pass through a semi-transparent material. This can play an important role in determining the appearance of the shadow that a 3D object casts.

Normally, this transmittance behavior will be defined in a 3D object's Material tab. You should override this by selecting "Force All Materials Non-Transmissive" in your spot light control panel. This will cause a shadow map produced by your spot light tool to ignore all transmittance behaviors for your 3D objects.

As you might have surmised, you will need to practice and experiment with all of these nodes, tools, and settings quite a bit before you will really become comfortable with them. The 3D components of Fusion are inherently complex—as 3D software can tend to be—and take some experience to master completely. But it is well worth it, especially where your VFX project pipeline is concerned, as it can help you achieve both realism as well as the wow! factor in your work which will set it apart from the crowd.

Summary

In this fourteenth chapter we took a look at the 3D compositing concepts, principles, and capabilities available in Fusion 8. We looked at 3D geometry tools, camera 3D, lighting, and shadows. We found out how to use a merge 3D node to create 3D scene aggregations, and at how to use a renderer 3D node to turn these from 3D data into 2D data.

In Chapter 15, you will learn about advanced 3D topics, such as **surfaces, materials, shaders, textures**, **special effects**, and other concepts and techniques we didn't get around to during this chapter. Fusion 3D compositing is a complex topic, and it will in fact take the next several chapters to cover all of the important 3D features available for use in your VFX project pipelines.

Chapter 15

VFX Pipeline 3D Rendering: Shader, Material, and Texture

Now that you have an understanding of the fundamental concepts, terms, and principles of the Fusion 3D compositing engine, it is time to get into some more advanced 3D concepts such as surfacing, environmental effects, shaders, materials, textures, and mapping, among other things. Some of this we covered back in Chapter 6, and now you will see it applied to your VFX project pipelines at an advanced level during the last half-dozen chapters in this book. That includes this one, in which we will make the geometry and lighting we learned about in Chapter 15 look more realistic by providing surfaces using materials.

We will look at how shaders are created using materials, textures, algorithms, and mapping, and at how they are applied to a 3D geometry-camera-lighting pipeline such as the one we created in Chapter 15. Since shaders are advanced, especially for a fundamentals title, we'll look at how to use Bin shader presets and how to dissect them in order to learn how they work, as well as at how to tweak them to optimize them for the VFX project pipeline.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_15

3D Shaders: Concepts and Terminology

Shader is the top-level term for surfaces that are added so as to be able to apply a "skin" to your 3D mesh geometry, which we will be doing in this chapter to the Gargoyle you first saw back in Chapter 14. Shaders are comprised primarily of **materials** and **textures**, and could also incorporate **effects algorithms** and **UVW mapping coordinates** to add realism and position the shader on the 3D mesh, respectively. These can get incredibly complex, as you will see during this chapter. There are even texture map design software packages available, such as Allegorithmic's **Substance Designer** and **Substance Painter**, which work in a similar fashion to Fusion's tile-based visual design process.

Material: Illumination Models to Define Appearance

Some people use the term *material* interchangeably with the terms *shader* and *texture*. However, in Fusion 8, as well as in other 3D software, **shader** is the term for a construct that contains the **material type** and the **texture assets**. At a core level, a material is based upon an **illumination algorithm** that defines how light will be processed when it hits the surface of the 3D object. Fusion gives VFX designers a plethora of tools for applying surface materials to 3D geometry, for applying textures to those materials, as well as tools for applying **UVW maps** to materials to show how the textures are to be positioned.

Algorithms that describe a material's response to light are termed **illumination models** and are named after the creator of the algorithm. They're found in the **Add Tool ➤ 3D ➤ Material** context menu. Jim Blinn created the Blinn-Phong algorithm, called **Blinn** in Fusion 8, Bui Tuong Phong created the **Phong** algorithm, Gregory Ward created the **Ward Anisotropic Distribution** algorithm, and Robert Cook and Kenneth Torrance created the **Cook-Torrance** algorithm. Each of these are 3D illumination model algorithms which process photons of light in a completely different fashion.

Each Fusion node (tool tile) that creates or loads a 3D mesh geometry asset into a merge 3D scene will also assign your **Blinn** material as a default. You can override this by using one of the **Add Tool ➤ 3D ➤ Material** nodes. This creates a custom 3D material tool (node) tile that you can wire into your pipeline.

Many of the more advanced materials will allow a greater degree of control over determining how your 3D geometry should react to light. As you will see, many material node tiles will provide inputs for many different types of mapping "channels," including diffuse or specular maps, bump maps, and environmental maps. Environmental maps can simulate reflection or refraction. There are also reflection maps and illumination maps. Materials can be combined in Fusion flow node editor in order to produce elaborate, and highly detailed, **composite materials**.

Textures: 2D Images Create Surface Characteristics

Materials can be enhanced through the use of **textures**, which are typically implemented using 2D imagery. Texture maps modify the appearance of a material on a per-pixel basis. This is done by connecting an image or other material to the inputs on a material tool in your flow node editor. Textures are used to define a "look" for your 3D object by adding photorealistic details such as color, reflections, transparency, illumination, bumps, or special effects. More complex textures such as sphere or cube maps, gradients, and fast noise algorithms are located in the Fusion 8 **Add Tool ➤ 3D ➤ Texture** context-sensitive menu.

Figure 15-1 shows a simple application of a 2D image as a texture for the image plane 3D node that we utilized back in Chapter 14. Let's finish that simulation now and show how image assets can be shadowed by 3D objects by using **Add Tool ➤ Loader** and wiring your **NikiFrame.png** 2D image into your image plane 3D node, as shown highlighted in red in Figure 15-1. Since Niki is now **texture mapped** onto the image plane, the shadow treats your 2D asset as if it were part of this 3D object, which it now is!

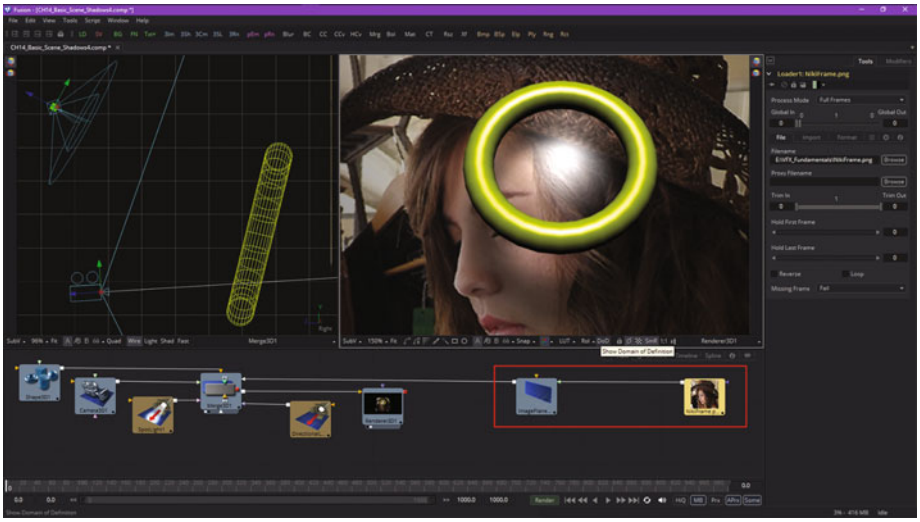

Figure 15-1. *Add an image loader to provide a plane primitive with a 2D texture map*

UVW mapping coordinates, assigned to your 3D geometry in the 3D software you are using to create your 3D assets, are what are used to conform and position any 2D imagery onto your 3D geometry assets.

When the renderer 3D node, which is also an algorithm, renders each pixel of the merge 3D node's scene, each material will modify the input data attached to it. It does this according to the data values of the corresponding pixels in the 2D image texture map that is assigned to each of the texture channel inputs allowed by that type of material's illumination-rendering algorithm.

If you have not done UVW map application in your 3D tool (software package), the UVW mapping tool in Fusion is called the **texture transform** tool. UV mapping is the method used to wrap a 2D image texture onto image plane 3D geometry. Similar to X and Y coordinates in a frame, U and V are coordinates for textures on planar 3D objects, or 3D objects mapped with two dimensions. UVW mapping is a three-dimensional version of UV mapping, and is the standard texture mapping coordinate system that is currently used for 3D and i3D applications in the 3D and VFX industry today.

Texture maps are used to provide data to material inputs, which then hand them to the material illumination algorithm to add characteristics to the surface shader, such as diffuse color and specular color, specular exponent, specular intensity, bump mapping, opacity mapping, glow mapping, reflection mapping, and so forth. The most commonly used texture map is diffuse color, which we've used in Figure 15-1 to add the image to the plane.

A material tool that technically outputs a material and not a texture can also be used instead of an image to provide your other shader texture input (texture map channel) options. This will allow increased design complexity and shader design flexibility that only allowing 2D image data would not permit.

Material data passed between tools are **RGBA samples** that contain no other information about the shaders or textures that produced them in the node upstream from the material input.

Composite Materials: Complex 3D Surface Shaders

As you are starting to realize, you can build complex 3D materials by connecting the output for one material tool to one of the material inputs of another downstream material tool, and even to a texture tool if required for a surface shader design.

When a material tool output is supplied as a data input, it's processed in exactly the same way as data from a 2D image; that is, the pixel array's RGBA values are used per pixel, just like they were a 2D image type of texture.

Think of this like the 2D tool nodes in Fusion requiring a renderer 3D node to process that 2D data stream before it gets to the 2D node. This material-to-material wiring will allow for very advanced surface shader compositing pipelines.

For example, if you need to combine anisotropic highlight features with a Blinn material, you could take the output of the Blinn material, including its specular highlight, and use it as a diffuse color for your Ward anisotropic material, which we'll be using next. This would "re-light" the Blinn material by using the Ward material as your compositing input data source.

If you don't want the output of the Blinn to be relit by the Ward material, you'll instead use the **channel Boolean** tool, which has the algorithmic capability to add the Ward material's anisotropic specular component to the Blinn material. This will be more difficult to set up (wire together), but would give you a greater degree of surface shader compositing control.

Next, let's take a closer look at how you should dissect the Fusion Bin's shader definitions that come with Fusion 8 to better learn how materials and textures work together to create surface shader looks (designs) for use with 3D geometry assets.

Exploring Fusion 8 Shaders: Bin Presets

The fastest way to both learn how to create shaders as well as learn how to use them in your VFX projects is to use the Fusion Bin to add shader definitions to your projects, and then tweak them to get the results that you desire. This is the best approach for a fundamentals title, as it allows you to use the power of Fusion and still learn how to wield this power without building shaders from scratch, which we'll leave to a more advanced title.

Gargoyle Project Setup: Lighting an FBX Mesh 3D

Let's use what we learned in Chapter 14 and set up a VFX project for use in exploring 3D surface shaders, material types, and texture map effects in the remainder of this chapter. I will import a Gargoyle using a FBX mesh 3D node, add a camera 3D node and a merge 3D node, and then add lighting and shadows to achieve a dramatic effect using a base white/gray surface color so that the shader does not interfere (initially) with my light setup. These next few steps will reinforce what you learned back in Chapter 14, only this time using a more advanced 3D asset for shading, as shown in Figure 15-2 along with the merge 3D context menu.

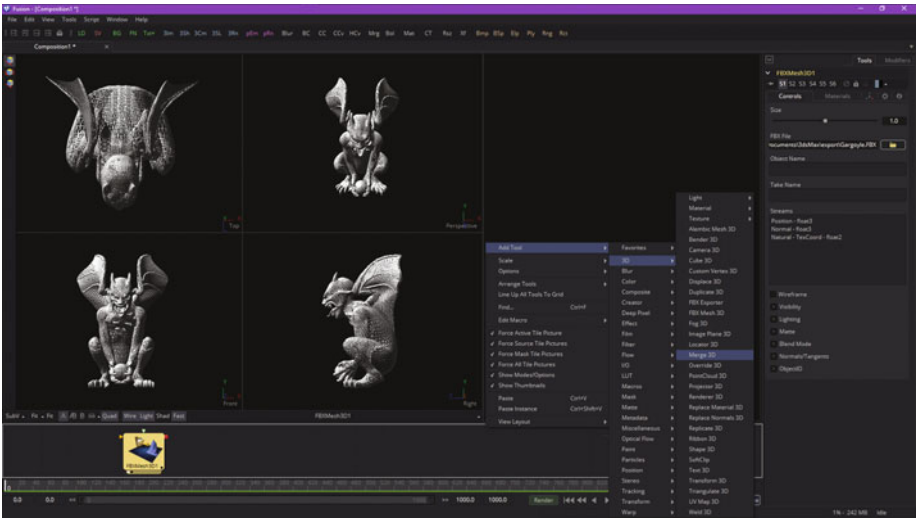

Figure 15-2. Import the Gargoyle.FBX file using the FBX mesh 3D node

Wire the FBX mesh 3D node into the merge 3D node, then add both a camera 3D node and a renderer 3D node into the merge 3D node, as seen in Figure 15-3. Enable lights and shadows in all nodes.

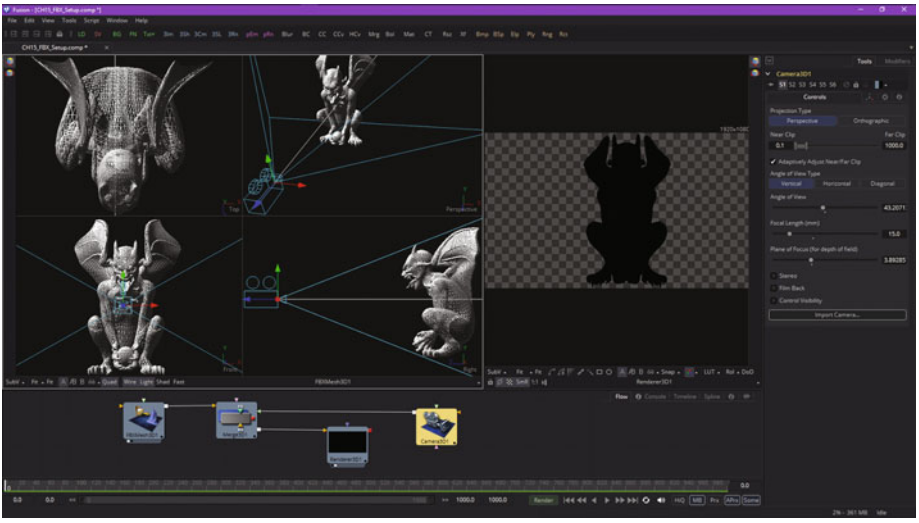

Figure 15-3. Create a basic 3D scene-rendering pipeline in Fusion

Position your camera in front of the Gargoyle in View1 and set your renderer 3D output to display in View2. Add a spot light node and position it as shown in Figure 15-4 for dramatic lighting. Set options to no decay, 90° cone angle, and 10° penumbra angle.

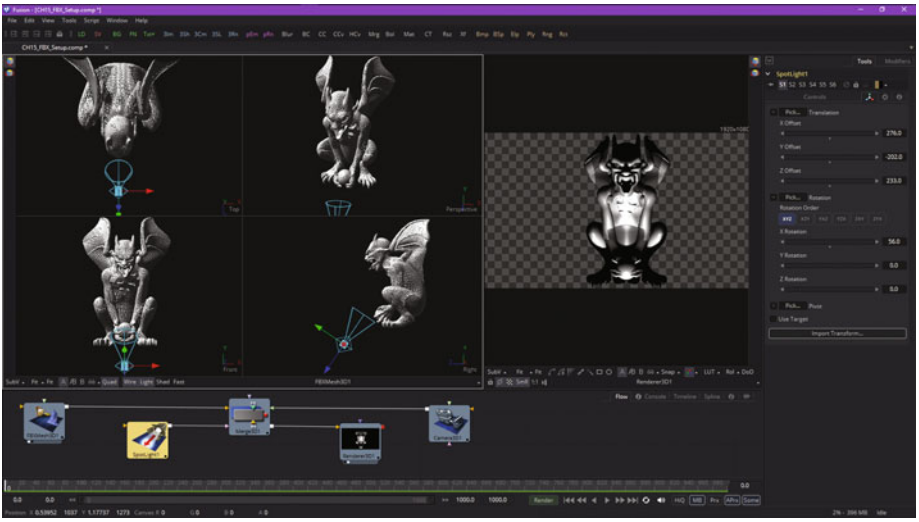

Figure 15-4. Add a spot light and position it for a dramatic lighting effect

As you can see in Figure 15-5, I doubled the shadow map's **quality** to **2048**, so now it actually renders when you change the settings. This can be seen in your renderer 3D tile, circled in red, which turns into a **progress bar** (which is pretty cool). As you can see in the control panel, I reduced **Shadow Density**, from 50 percent to 33.3 percent and set **Spread** to 3, **Minimum Softness** to **2**, and **Filter Size** to **0.03** to achieve a more cinematic shadowing.

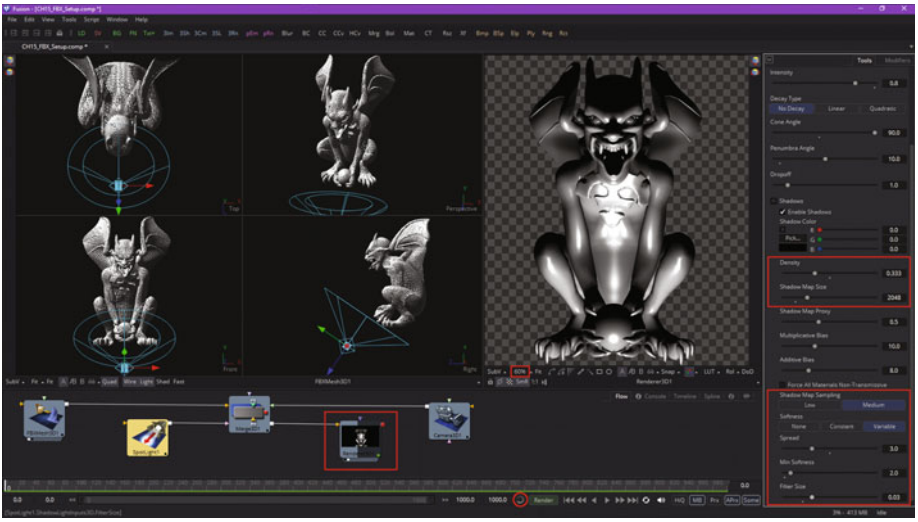

Figure 15-5. Set your shadow parameters for a realistic effect

Next, let's take a look at how some of the Fusion shader presets are algorithmically created to learn more about how you can use material and texture tool nodes in Fusion 8.

Sand Shader: Using FastNoise and BumpMap Tools

One of the ways in which you can see how to create Fusion shaders (materials + textures) is to go through the existing Bin shaders and arrange the tiles in your project to see how they function. I have set up the Gargoyle project for this purpose, so you can open the `CH15_FBX_Setup.comp` project and drag different shader presets into the base project, wire the shader to the FBX input, and then analyze how the nodes work by tweaking the parameters in the control panel tabs on the right-hand side of Fusion 8.

Let's use the **File ➤ Bins** to open the Fusion presets and select your **Templates on local host ➤ Shaders** section, shown in Figure 15-6. Drag the **Sand shader** into the flow node editor to add this group of material and texture nodes to your project.

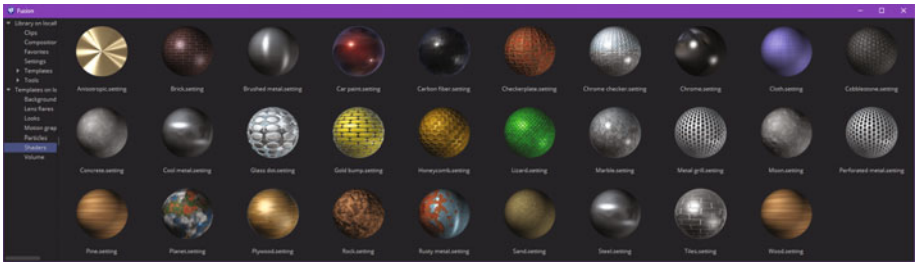

Figure 15-6. Open the Fusion Bin and select the shaders section

Click the **Open Group** icon, which looks like that window-sizing icon in the OS that switches between full-screen and sized windows, on the group tile to open the group window, as seen in Figure 15-7, so you can see and work with its contained nodes.

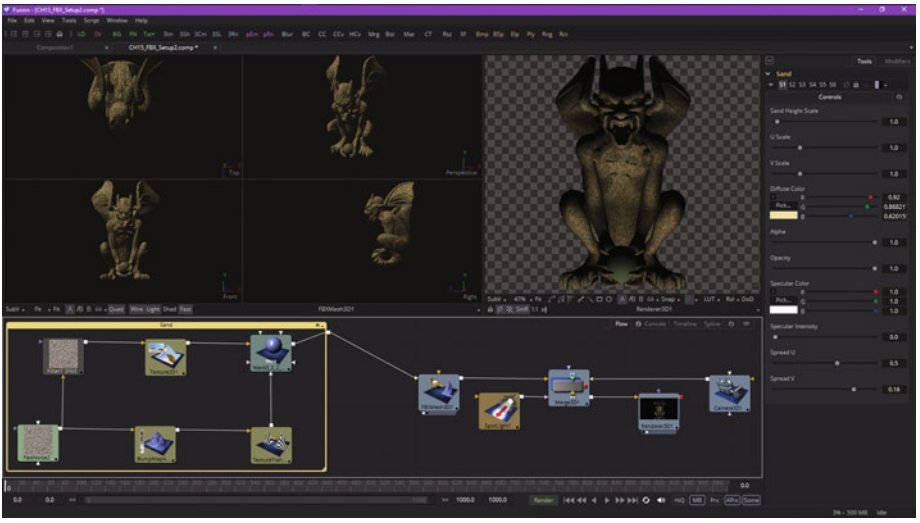

Figure 15-7. Drag the Sand shader into your project and connect

Wire the output for the Sand group node to your FBX mesh 3D node's **FBXMesh3D1.MaterialInput** triangle (input), which will add the surface shader (material and textures) to the gargoyle. This can be seen on both View1 mesh and View2 renders in Figure 15-7. If you select the Sand group window (yellow), you'll get global shader controls in a Fusion control panel on the right. You can use these if you like, but since we are learning shader design, we'll go inside the shader and select individual tools.

Select the FastNoise2 node tile and lighten the color by setting a **50 percent gray** color, instead of black, for the noise **Color 1** value, shown circled in red in Figure 15-8. As you can see in the renderer 3D View2 preview, this gives a more realistic skin for the Gargoyle—a smoother, skin-like surface appearance.

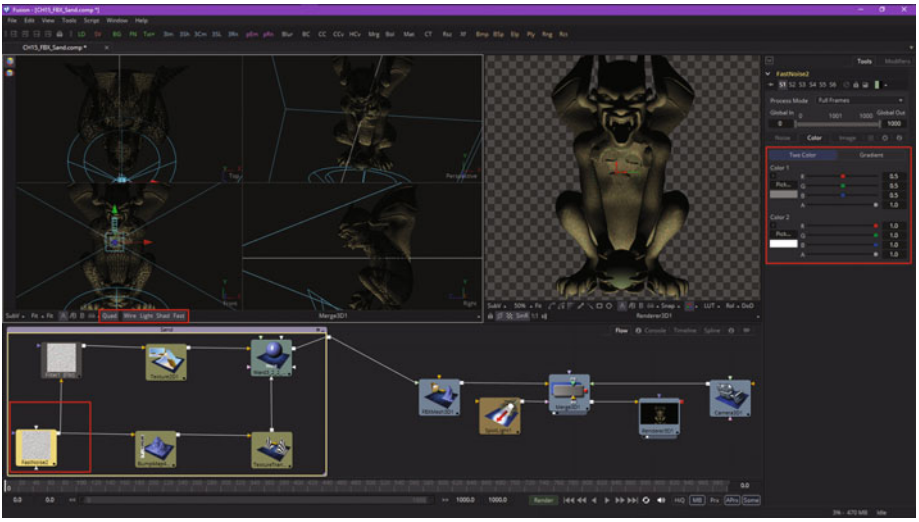

Figure 15-8. *Edit the FastNoise2 node color algorithm to 50 percent gray*

Next, let's select the **Filter1** node tile and enable the green color values, which will add back in some contrast in the noise, as can be seen on the left in Figure 15-9 in the preview provided by the renderer 3D node view. Select the **Ward material diffuse color** to make it pastel by increasing red and green to 100 percent and the blue to 80 percent, as seen on the right in Figure 15-9. This makes the statue look more realistic. I made the 3D statue shinier by adding **25 percent specular intensity** to the Ward material.

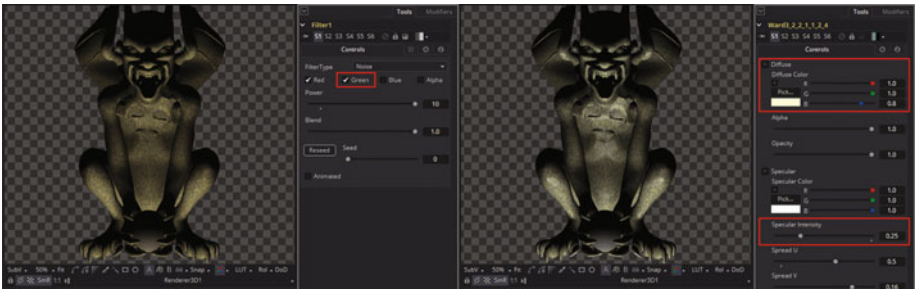

Figure 15-9. *Edit filter mode to add green and Ward node color*

Next, let's dissect one of the metal shaders to make the gargoyle statue surface look like it is made out of metal.

Brushed Metal: Add Reflect and SphereMap Tool

Open your base FBX project and add the **Brushed_Metal** shader from the Fusion Bin, then connect it to your FBX mesh 3D node, as is shown in Figure 15-10. Open the Brushed_Metal group and arrange the tiles so you can see the design progression. As you can see, this design is more complex and uses the same tools as the Sand shader, while adding **SphereMap** and **reflect** tools in the processing pipeline, as well as **rectangle**, **transform**, and **color correction**.

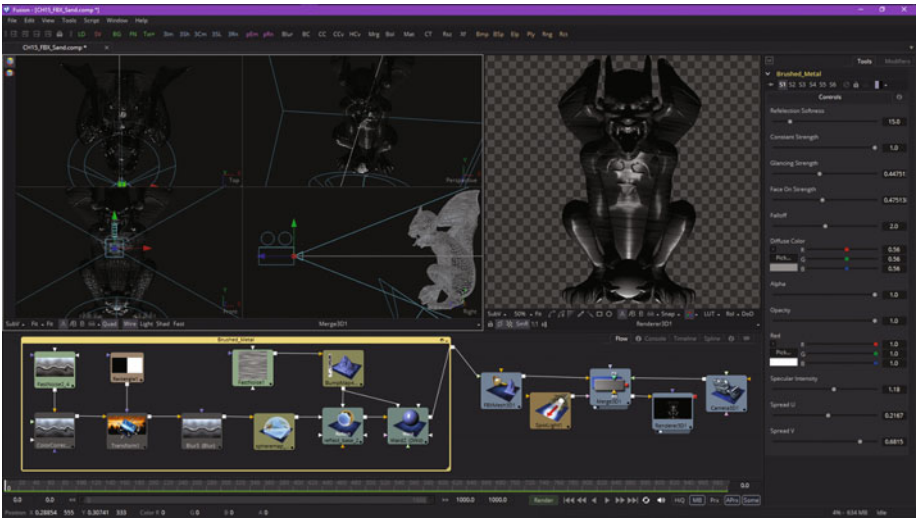

Figure 15-10. Add the Brushed_Metal shader to your FBX project

Let's change a few of the algorithm parameters so we can get rid of some of the banding that we see in Figure 15-10. You can do this by selecting the **FastNoise1** tile and decreasing the **Y scale** 400 percent from 300 to 60, and then increasing the **X scale** 300 percent to 6. I also selected **"Inverted"** to invert the noise, and set **Detail** to 2 and **Brightness** to 50 percent, as can be seen on the left side of Figure 15-11.

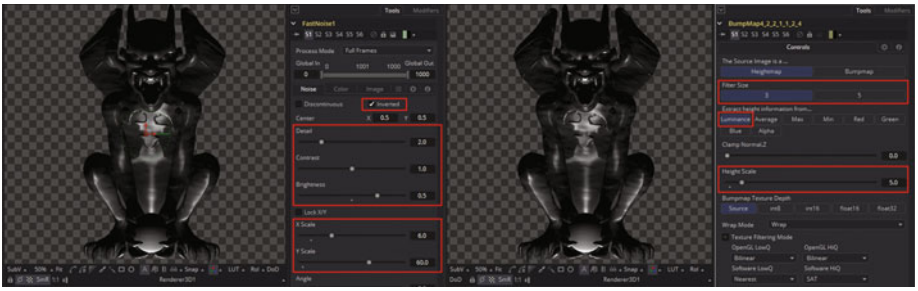

Figure 15-11. Tweak FastNoise node and BumpMap node parameters

Next, select the **BumpMap4** node tile and set **Filter Size** to 3, and then increase **Height Scale** to 5 to get more bumpiness. I also used a pixel **luminance** value to create the bumps, so the bumps match the light and dark areas of the shader, as is shown on the right side highlighted in red in Figure 15-11. This will give you a more realistic metal statue shader result.

Next, let's take a look at an even more complex shader, the **Lizzard** shader, which creates a lizard skin effect, and then tweak its parameters to better fit the Gargoyle statue project.

Lizard Skin: Add Crop, Blur, Ellipse, and ColorGain

Open your base FBX project and add the **Lizzard** shader from the Fusion Bin, then connect it to your FBX mesh 3D node, as is shown in Figure 15-12. Open the **Lizzard group** and arrange the tiles to show the processing pipeline. This design is even more complex, uses the same tools as the Sand and Brushed_Metal shaders, and adds the **crop** and **blur** tools into the processing pipeline, as well as the **Ellipse**, **MatteControl**, and **ColorGain** Tools.

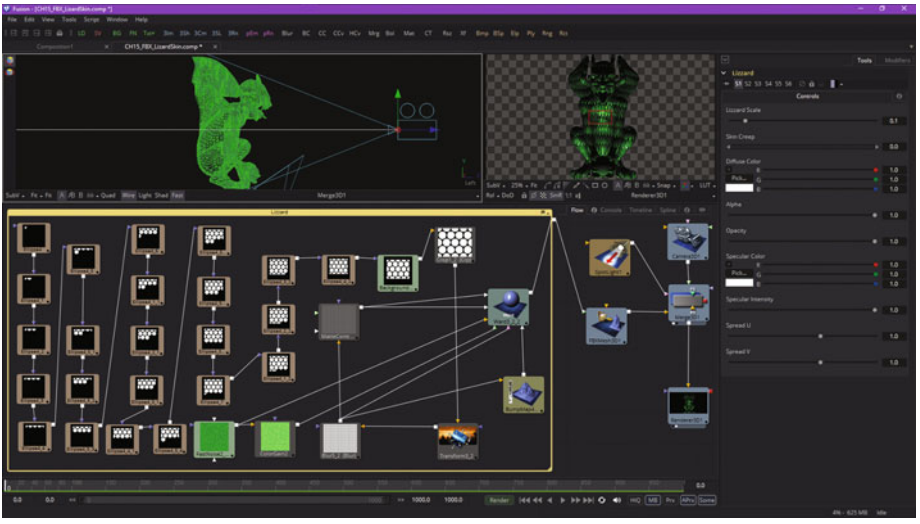

Figure 15-12. Add the Lizzard shader and wire it to your FBX mesh 3D node input

As you can see in Figure 15-12, even on a 2K resolution HDTV display, there's not much room for the working views! This is why I recommended a dual HD display setup, or a 4K UHD display setup, as VFX project pipelines can quickly get quite complex.

As you can see, highlighted in red on the right side of Figure 15-13, I decreased the **Lizard scale** from 0.1 to 0.03333, or 200 percent, to get a more realistic lizard skin effect. You'll see how this "wires inside" of this Lizzard shader design in a bit.

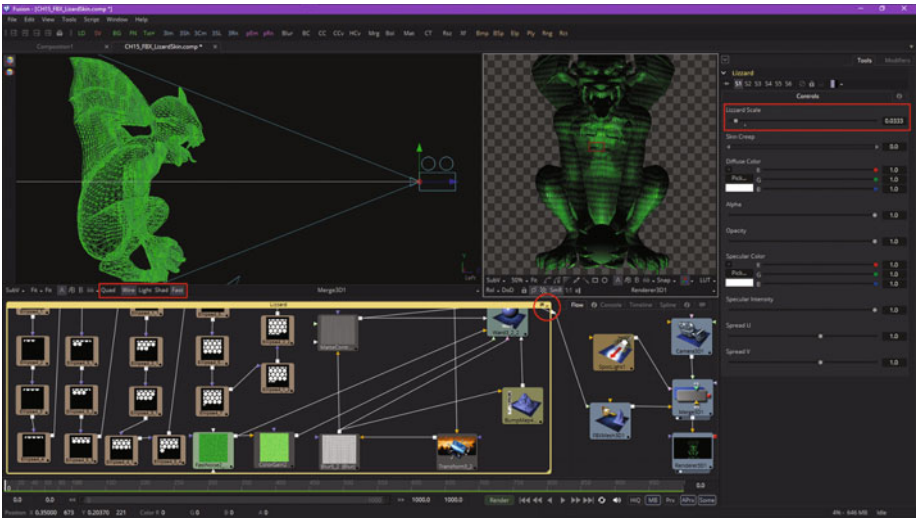

Figure 15-13. Resize your view UI and enable group-view panning

As you can also see, I've decreased the size of the group window so that I can better fit these merge 3D and renderer 3D viewports. Also notice I have turned off lights and shadows for the merge 3D (left) View1, so all you see in the 3D mesh is the color values from the shader. I've also selected a Group Window Local/Global View Pan icon toggle, shown circled in red, which allows you to pan a group view without affecting your flow node editor view. Make sure to learn all these Fusion GUI features!

Next, select the **Transform3** node tool inside the group and notice that a global Lizard scale parameter from that group has been passed into the Transform3 **size** parameter, showing you how that global control panel is wired into its component nodes' tools.

Let's also change the **aspect ratio** so the scales in this lizard skin are not so elongated, as this does not look as realistic. I decreased the aspect ratio 25 percent and set it to a value of 75 percent, or 0.75, as shown in Figure 15-14.

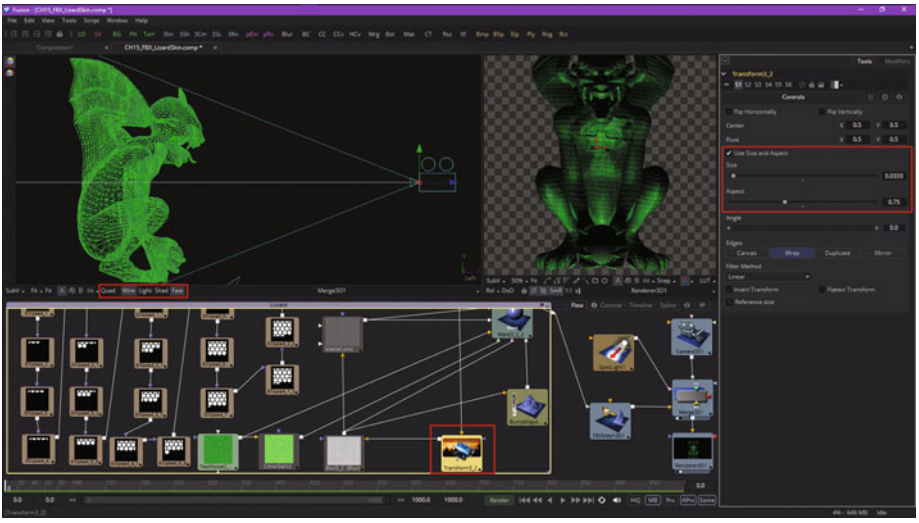

Figure 15-14. *Select Transform3 node and change the aspect ratio to 75 percent or a value of 0.75*

Spend some time exploring and tweaking this shader and its materials and textures so that you can see how they work together and the parameters that you have available to you.

Summary

In this fifteenth chapter we took a look at how to make your 3D composition look photorealistic using shaders, materials, and textures. We looked at how to dissect the Fusion Bin shader presets and got some more experience using the 3D composition UI settings, widgets, and tools, including FBX mesh 3D, camera 3D, merge 3D, renderer 3D, and lighting and shadows.

In Chapter 16, you will learn about using **text 3D**, since every VFX fundamentals title should cover titling effects.

Chapter 16

VFX Pipeline 3D Modeling: 3D Text-Title Creation

Now that you have an understanding of the fundamental concepts, tools, and workflow for 3D composition in Fusion, we should look at a specific application of this called **3D titling**, which VFX agencies are often approached to do for clients. Thanks to the text 3D tool, which we haven't yet taken a look at, but should, these projects can be done 100 percent within a Fusion 2D and 3D composition environment.

We will look at how to create cinema-quality 3D text and how to apply advanced effects to your 3D titling pipeline.

3D Text Tool: Create Professional 3D Text

Fusion has always had 3D text capability to provide a titling engine to VFX developers, as film credits and television shows have always required either 2D or 3D text production. The text engines (2D text and 3D text) are very similar in Fusion, but since we are in the 3D section of the book, we'll look at the more advanced text 3D node tool in Fusion, and how to utilize it to create professional 3D text and text titling visual effects.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_16

Classic 3D Text Extrusion: Basic 3D Text Modeling

Let's jump right in and fire up Fusion and create a new project. Right-click in the flow node editor and use the **Add Tool ➤ 3D ➤ Text 3D** menu sequence to add a text 3D node, as shown in Figure 16-1. Enter "VFX Fundamentals" into the **Styled Text** box in the control panel area. I used an **Arial** font and default **1.0 Size** setting. Click the view dot at the bottom of the tool and resize that viewport to fill the screen, as the aspect ratio is very wide for this text element. All of this can be seen in Figure 16-1, as well as in the **Extrusion** section of the Text3D1 node control panel, currently in its collapsed (closed) state.

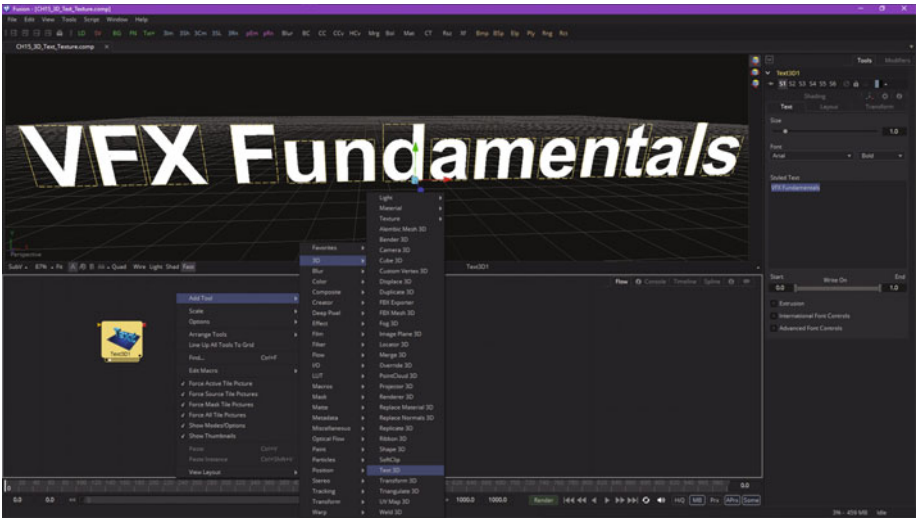

Figure 16-1. Add a text 3D node, enter "VFX Fundamentals" text

Now we can add the **merge 3D** node to create the 3D scene, and then wire the text 3D node into that. Then we can add the **camera 3D** node and wire that to the merge 3D node before adding a **renderer 3D** node and wiring it to the merge 3D node. Select View1 for the renderer 3D node and View2 for the merge 3D node, as seen in Figure 16-2.

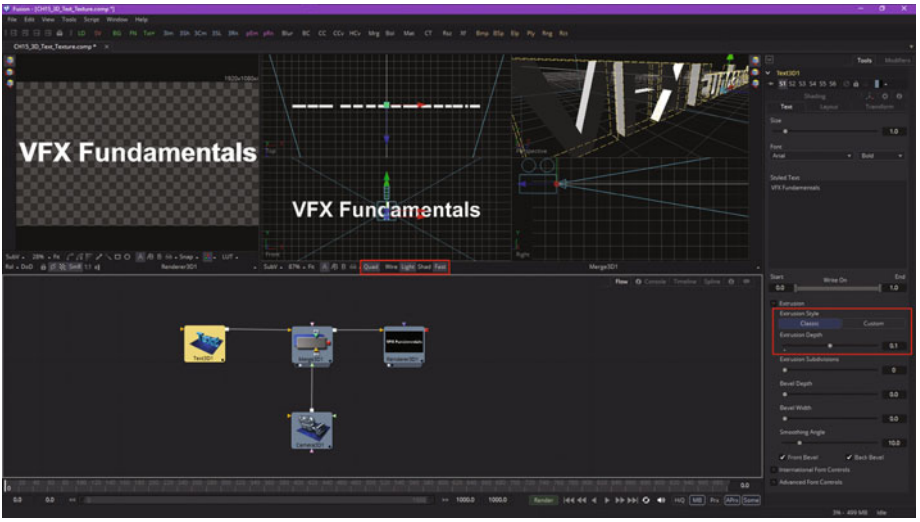

Figure 16-2. Create 3D scene and configure views to show output

As you can see in Figure 16-2, I have selected **Classic** or basic (simple) 3D text extrusion and used an initial **0.0125** value to extrude the text. I've opted to bevel both the front and the back, as you can see in Figure 16-3, since this is 3D text, and it may be seen from more than one angle. I am going to select both the **"Quad"** and **"Light"** options in the merge View2, so you can see the default scene lighting, as well as the camera positioning and the smooth, professional results that Fusion 8 should give you for your 3D text titling VFX project workflows.

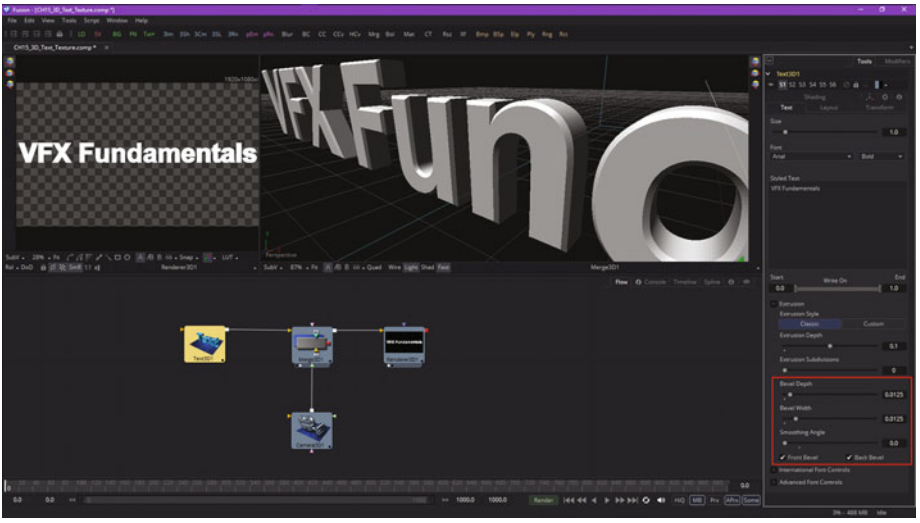

Figure 16-3. Set extrusion and beveling parameters

Next, let's add a spot light, so that we can cast shadows in this 3D text titling project. Use **Add Tool ➤ 3D ➤ Spot Light** from the flow node editor context menu, and wire the spot light up to the merge 3D scene, as is shown in Figure 16-4.

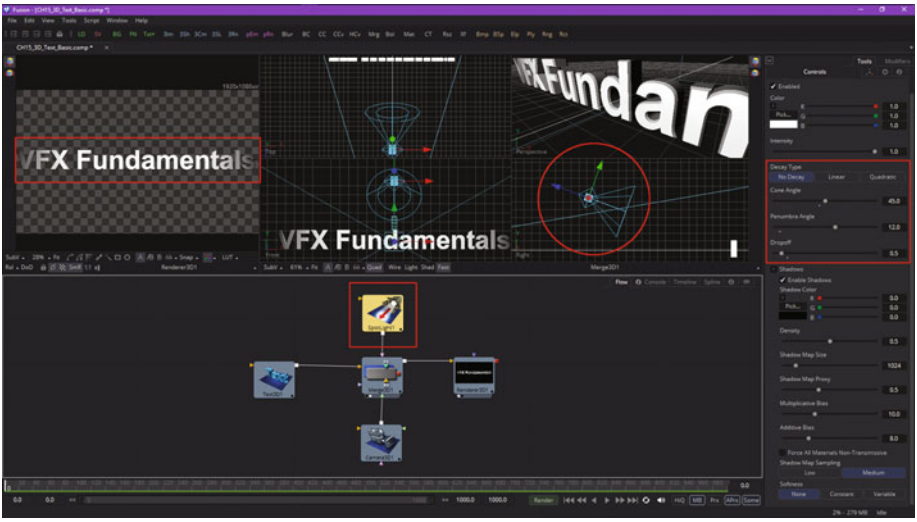

Figure 16-4. *Add a spot light, and set its two cones and falloff*

I initially used **No Decay** for a bright spot light effect and set **Cone Angle** to 45°, **Penumbra Angle** to 12°, and **Falloff** to 0.5 or 50 percent, as shown highlighted in red on the right side of Figure 16-4. The results of these settings can be seen in the renderer 3D View1, on the left side, as well as in the merge 3D View2 quad viewports, seen on the right in Figure 16-4.

To enable these lighting effects to show in the Front or in the Perspective quad viewports, or in both, you must click your mouse in each viewport and then select the "Light" option at the bottom of the Quad view.

These are the minimum settings that you would need in order to implement basic 3D text titling assets in Fusion 8. I will show you some more advanced ways to enhance the visual effect of the 3D text title over the rest of this chapter.

To see that this is indeed 3D mesh geometry, click the **Wire** button on the bottom of your merge 3D View2, and also turn off the **Fast** shader (for better wire quality), shadows (**Shad**), and the **Quad** view, so you are showing only the **Perspective** view. This is shown in Figure 16-5, highlighted in red at the bottom of the View2.

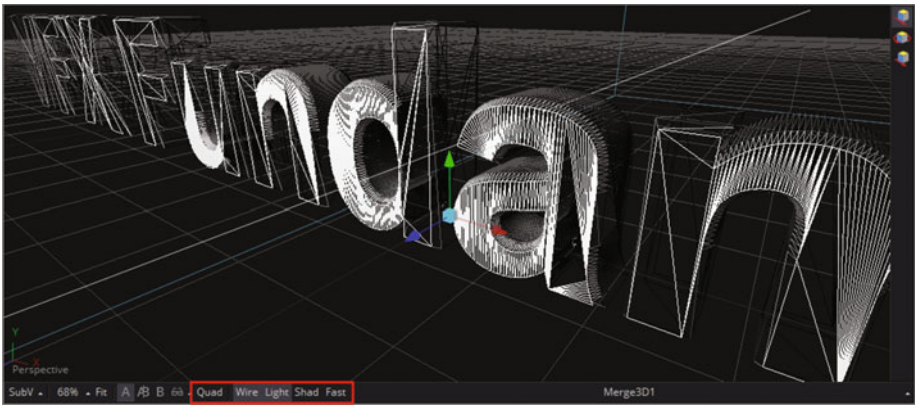

Figure 16-5. Turn on "Wire" and "Light" options to see the text's wireframe

You can see the beveling in the back of the 3D text with this current Perspective view; to see it better, turn off the "Light" option for the View2, as shown in Figure 16-6, which will now show every edge.

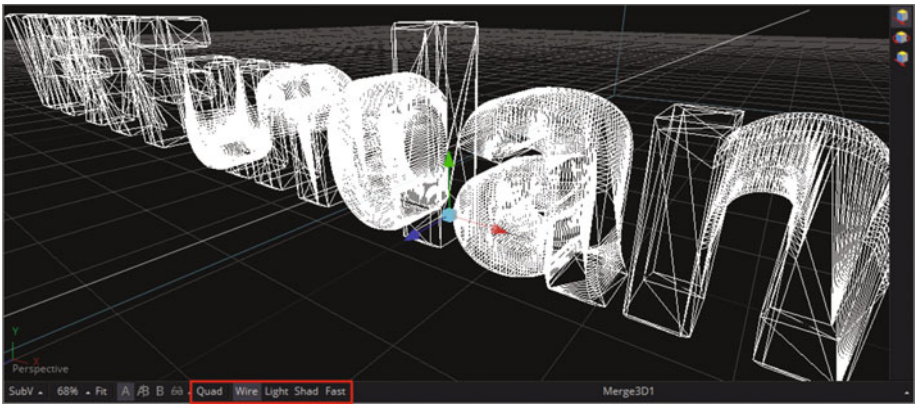

Figure 16-6. Turn off "Light" option to see beveling in the back of the text

Next, set the **Extrusion Subdivision** slider to **3** to make the mesh more complex, as seen in Figure 16-7. If you are going to **deform** (bend, twist, etc.) the mesh and need more data points, you will get a better (smooth) deformation result with more complex mesh. There should be another slider for adding subdivisions in the other dimension (between text shape vertices); hopefully, Blackmagic Design will add this in the future. The reason that Fusion's 3D text tool is not as seamless or advanced as other features is that it was acquired by Eyeon for the software, and was not coded from scratch. This may have to be done eventually in order to make this area of the software as professional as the rest of Fusion 8, and more seamless with other nodes (tools).

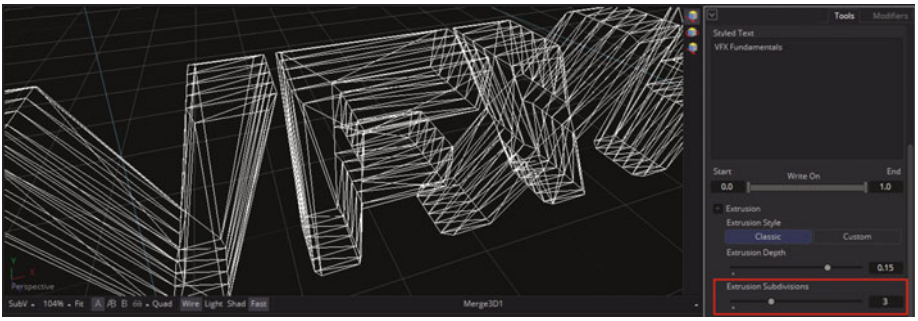

Figure 16-7. Set Extrusion Subdivision to 3, creating four sections

Next, let's take a look at how to create more advanced 3D text objects by using the **custom** 3D text option. This allows a spline-based approach to defining an **extrusion profile**.

Custom 3D Text Extrusion: Complex 3D Text Model

To give us more flexibility in modeling and animation, and to change this wide aspect ratio for your logo, let's make *VFX* and *Fundamentals* two separate nodes. Edit the **Text3D1** node to be just *Fundamentals*, as shown in Figure 16-8, and also edit your spot light parameters so as to fit the shorter 3D text span, as shown circled in red at the right in Figure 16-8.

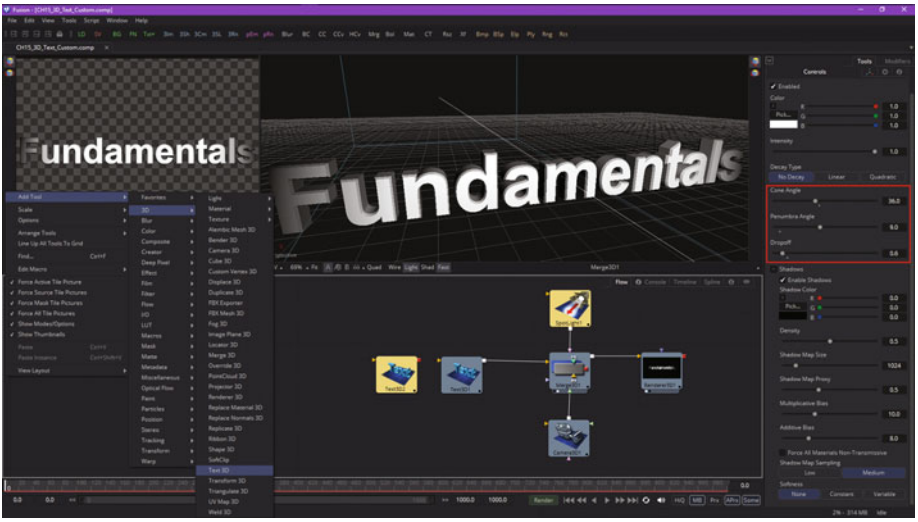

Figure 16-8. Make Text3D1 Fundamentals, and add a Text3D2 node

I set Cone Angle to **36**°, Penumbra Angle to **9**°, and Falloff of **0.6** to achieve the dramatic lighting effect for this bottom *Fundamentals* portion of the logo. I'll use Text3D2 to add a larger *VFX* portion to the logo that will be centered directly above the much longer *Fundamentals* lower component.

To do this, use an **Add Tool ➤ 3D ➤ Text 3D** context menu to add a Text3D2 node, also seen in Figure 16-8. I combined two screenshots into one (there are a lot of figures in this chapter), in case you are wondering why both Camera and Text3D are selected.

Wire the Text3D2 into your merge 3D scene node, as shown in Figure 16-9, and set the text value to "VFX" and the **size** to be twice as large as *Fundamentals*, or a value of **2.0**.

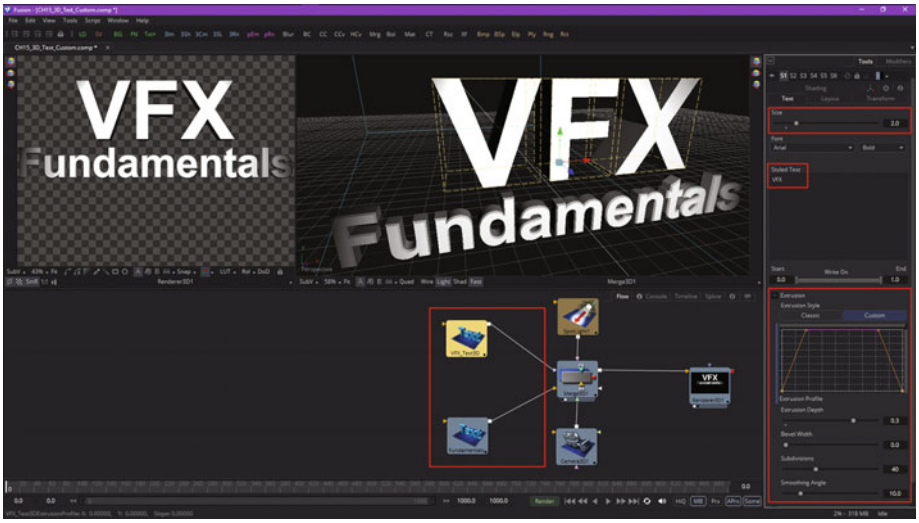

Figure 16-9. *Rename text 3D nodes; set VFX text basic parameters*

Open the **Extrusion** section of the Text3D2 control panel, and set **Extrusion Depth** to be twice the Text3D1 extrusion depth, or a data value of **0.3.** Fusion sets default **Subdivision** and **Smoothing Angle** data values to **40** and **10.0**, respectively.

Also notice in Figure 16-9 that I've renamed the Text3D1 node to be *Fundamentals* and the Text3D2 node to be *VFX_Text3D*.

Also shown in Figure 16-9 is your **Custom Extrusion Style** button. This allows you to spline model a 3D text profile. Once you select this, you'll see a starting spline profile setting in the **extrusion profile graph**. To see a detailed version of this spline, select the **Spline** tab, as shown in red in Figure 16-10.

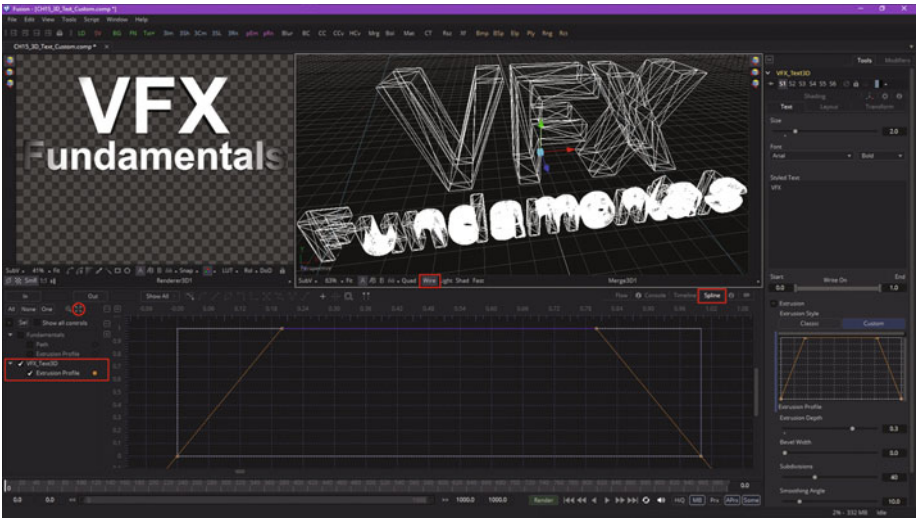

Figure 16-10. *Use a spline editor to edit the extrusion profile*

When you first open the Spline tab, you'll see a vertical line, a compressed version, which you can expand using the **Zoom View** icon, seen circled in red on the left in Figure 16-10. I also set the merge 3D scene view to **Wire** mode, so that you can see that Fusion puts polygons (edges and faces) in place where the points in the spline are located. If you use curves, Fusion will add points accordingly to approximate this new curvature.

Let's dive right in and see how it works! Click once in the center of the top straight section of the extrusion spline and pull down to create a "V." Set the VFX_Text3D node view dot to View1 and position your 3D view as seen in Figure 16-11 so that you can see what is going on with your actual 3D model. If you do not see a result, then set **Bevel Width** to **0.05**. The **custom** 3D text modeler seems to need a bevel to be set in order to do its thing! Add a second point to the straight top spline, creating a flat-bottom ditch, as seen in red on the right of Figure 16-11.

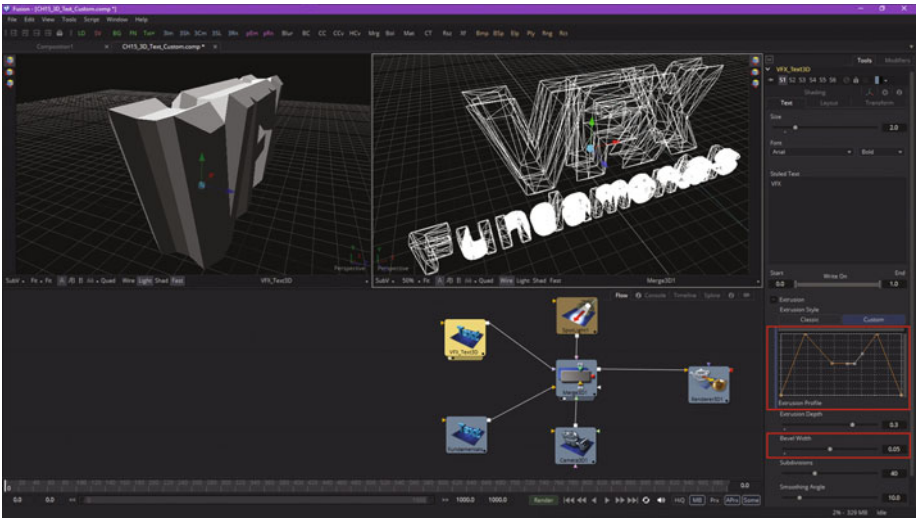

Figure 16-11. Set Bevel Width to 0.05, and create a flat-bottom ditch

As you can see in View1 (text 3D node preview), the spline is now defining the extrusion of the *VFX* part of the logo. Now, click the Spline tab; we will use the grid and do some more detailed extrusion spline modeling to create some really great 3D text modeling effects.

Add points on either side of the ditch and pull the flat part up and in, and the new points down, to create a “W” shape, as is seen in Figure 16-12. As you can see in the preview, this gives you a professional extrusion effect. Notice Fusion 8 only adds points and edges as needed to create this 3D text object.

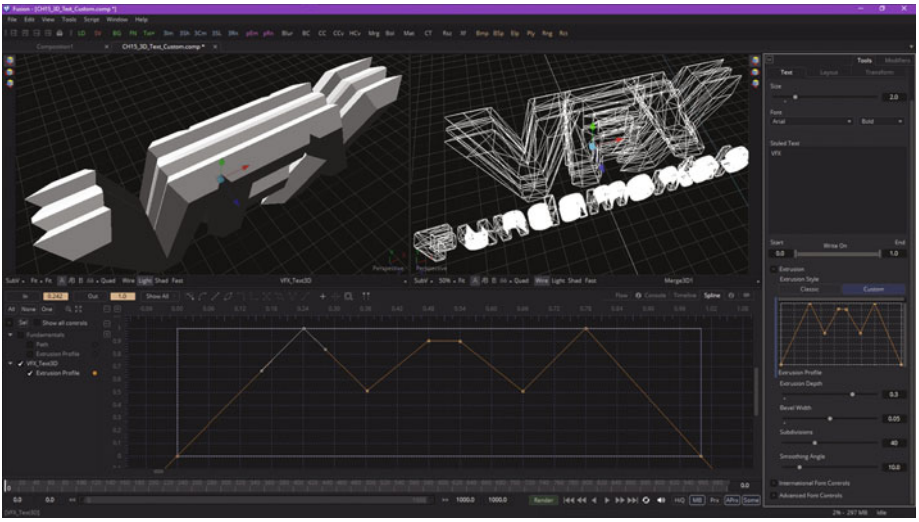

Figure 16-12. Create a "W" straight-edged shape in the spline editor

Add two more points to make a **notch** in the middle of the spline, as is shown in Figure 16-13. Keep all sections straight by moving only the points at this point (no pun intended), and not moving the tensioning handle (at least not yet). I used the spline editor grid intersections so that you can copy my work.

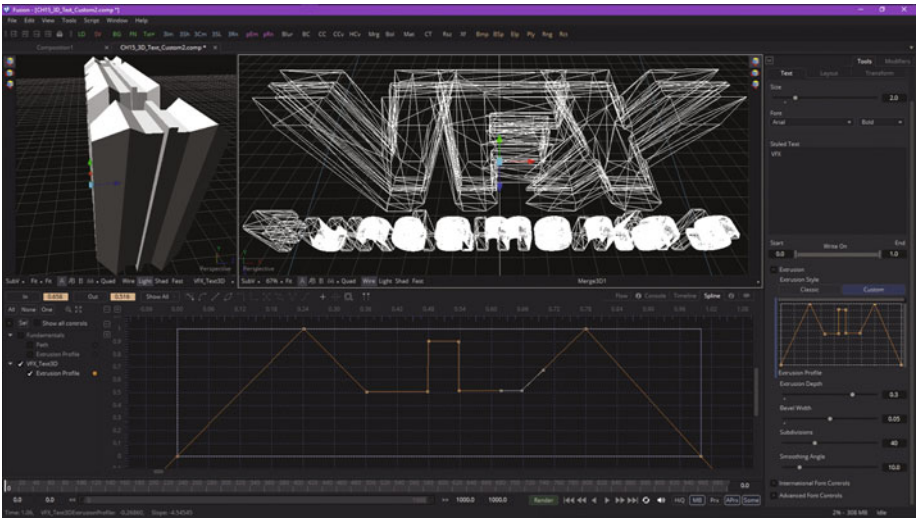

Figure 16-13. Create a notch in the center of the extrusion profile

Select the two corner points away from the notch, to pull out your tension handles. This will create a nice curve for the bottom of the trough, as can be seen in View1 in Figure 16-14.

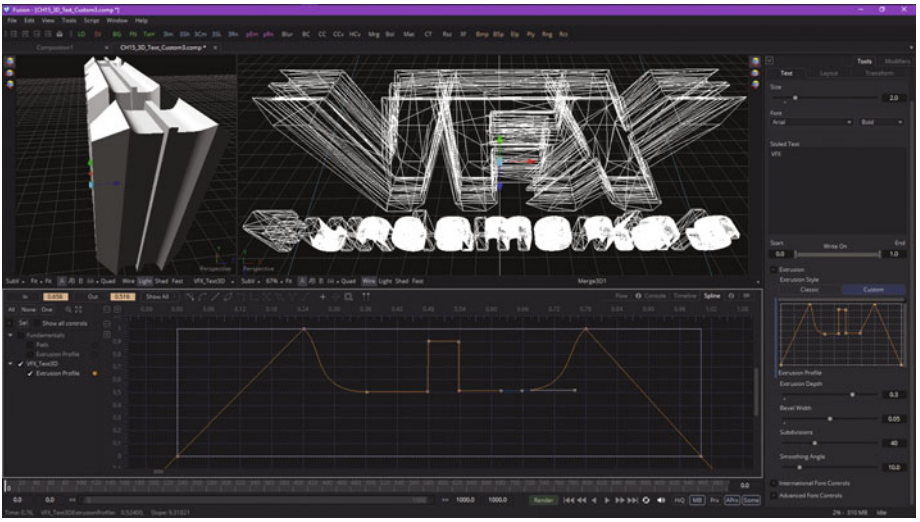

Figure 16-14. *Create a smooth trough bottom using the tension handles*

Smooth the transition into the trough with the top-point tension handle, creating a straight slope as seen in Figure 16-15.

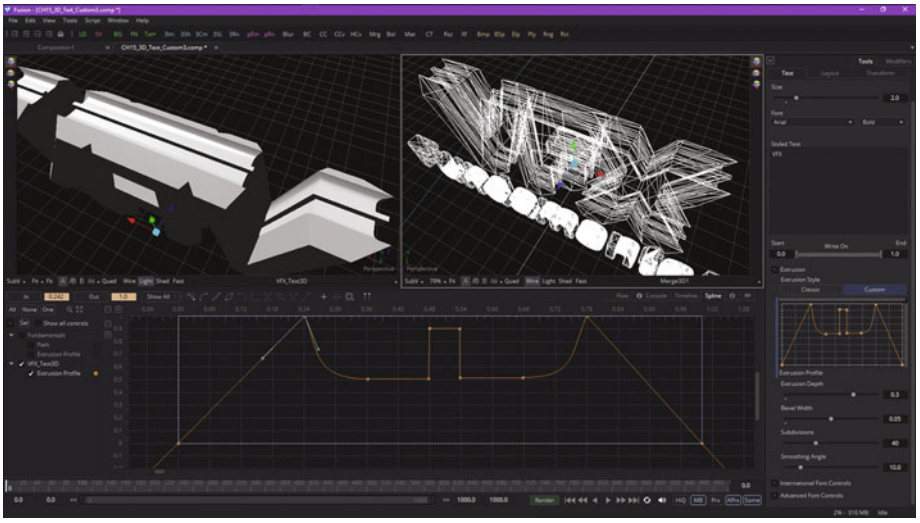

Figure 16-15. *Pull tension handle down to create straight slope*

Next, pull the outside spline tension handle up for each of the top spline points, creating a shape that looks like the Batman logo, at least on the outside (wings) part.

This adds a professional result at the top of the inner trough curvature, as you can see in Figure 16-16, and shows you can create a **curved outer bevel**, called a **chamfer** in 3D modeling terminology. Also notice that Fusion 8 added points and edges to the 3D text mesh object to approximate this curve, which must be using "smoothing groups," as it looks infinitely smooth in View1 (the text 3D node view) but faceted in View2 (the merge 3D node view). Smoothing groups tell your renderer where to smooth, and where to have seams, in the surface geometry topology rendering.

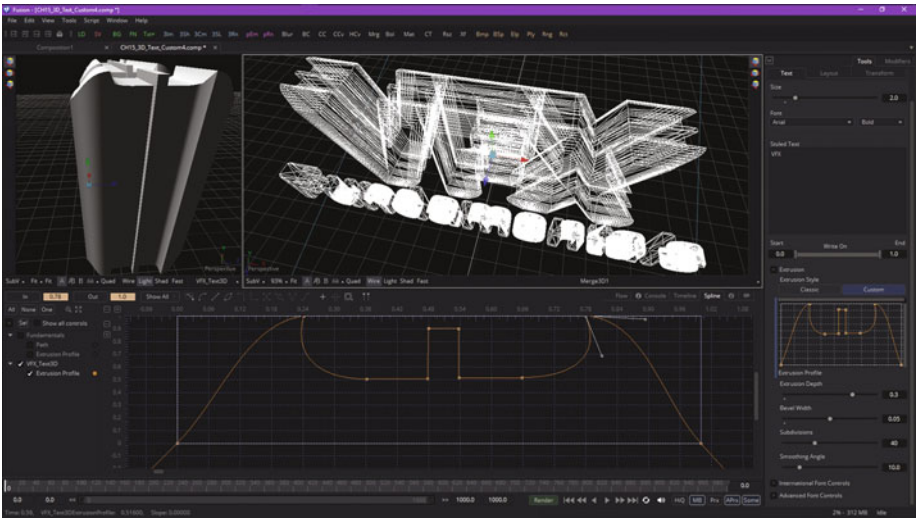

Figure 16-16. Lift outside tension handle, creating a Batman-esque curve

Be sure to experiment with 3D text node modeling features as well as with the spline editor, as both are important to VFX.

Summary

In this sixteenth chapter, we took a look at how to do 3D text spline modeling in Fusion 8 for use in titling sequences for your VFX projects. We looked at how to set algorithm parameters to have Fusion 8 model "classic" 3D text objects, and then looked at how to use splines to create more complicated custom 3D text elements, as well as how to combine the two into one VFX project.

In Chapter 17, you will learn more about how to go about creating **3D text-titling animation effects** using Fusion 8.

Chapter 17

VFX Pipeline 3D Animation: 3D Text-Titling Modifiers

Now that you have seen how to model 3D titling assets using the Fusion 8 text 3D node tool, the logical next step is to see how to use Fusion to create 3D titling animation sequences, as you will likely be asked to do this by a client at some point in time. We'll continue to work on that titling project started in Chapter 16, including refining and texturing the 3D titling assets that you created using an extrusion profile editor.

We will look at how to space out 3D text titles in 3D space in different ways, how to apply textures to different parts of the 3D titles using images and color values, and how to animate 3D logos out of the distance into place on the screen by using powerful Fusion algorithms called **modifiers**. In Fusion you can instruct one tool's output to modify another, and Fusion 8 also comes with a collection of modifiers to use with your VFX projects. These replace the need to keyframe a data value, or a collection of data values, by doing it with an algorithm. The result, either way, is 3D animation, and we will be creating 3D animation in this chapter using both approaches.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_17

3D Titles: Spacing, Texture, and Animation

Impressive 3D text titling assets can be made even more impressive in Fusion 8 by texturing them using your own custom imagery, and by animating them onto the screen in one fashion or another. You'll also need to make sure they are pleasing to the eye, which means they are spaced apart so that they can be read, and so that your 3D text characters can animate independently of each other, without crashing into each other (unless that is part of your intended visual effect).

Character Spacing: Give Your Logo Some Space

The first thing we want to do is to space the *VFX* part of the logo out, as it had become too crowded, and even overlapped, due to an extrusion profile 3D modeling process in the previous chapter. Spacing out can be done in the Transform tab, seen in Figure 17-1 circled in red, by increasing the character spacing by 16 percent, to a value of **1.16**.

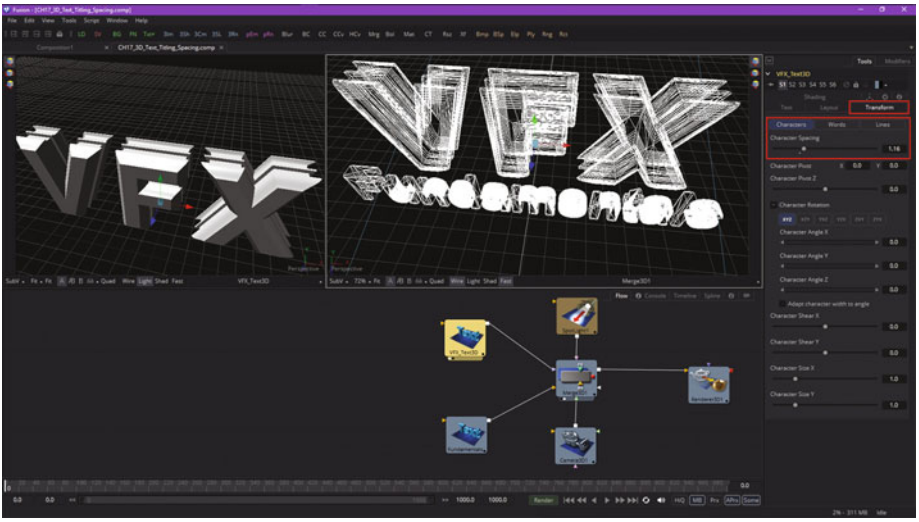

Figure 17-1. *Expand character spacing by 16 percent on the VFX portion of the logo*

As you can see, this makes the VFX portion of this logo easier to read, and closer to the width of the *Fundamentals* logo text, which, as you can see in View2, looks more professional than the original from Chapter 16.

Next, let's take a look at how to texture map your logo's components (faces and bevels) using images, which you can supply to the 3D text node.

Texture Mapping: Skin Your Logo Using 2D Images

The next thing that I want to show you before we get into 3D animation and using modifiers is how to give the 3D text tool a custom texture map image using the **Shading** tab's **Material** section, which has an image source **Clip** button and **Color file** selection widget, which are all shown, highlighted in red, on the right side of Figure 17-2. Notice I have selected **Material** as the type of image, and I have left the **"Use One Material"** option selected for now. Later, we'll deselect this option and specify different texture map imagery.

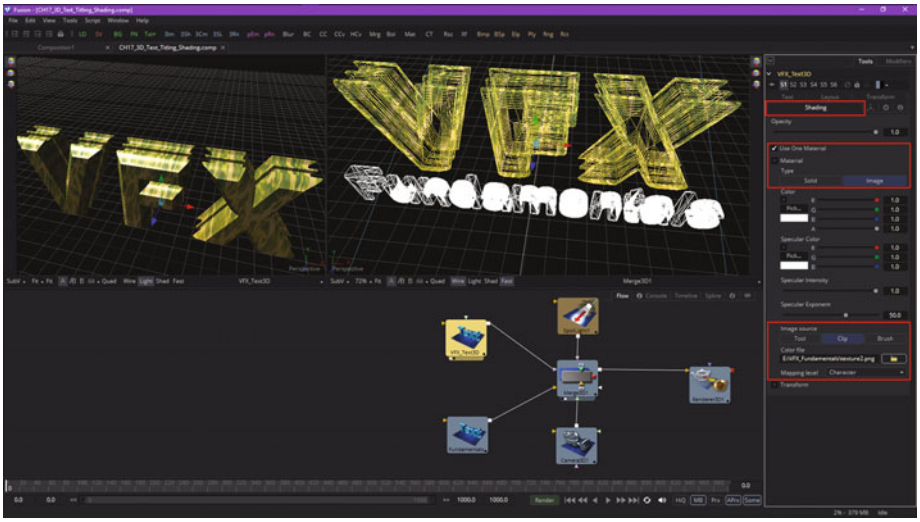

Figure 17-2. Use Shading tab's Material section option: Clip texture map

As you can see, the 3D merge and 3D text tool views also reflect this texture map. Deselect the "Use One Material" option, which will reveal the **Bevel Material** section of the Shader tab. I selected a blue aquatic image for the bevel texture map, seen in Figure 17-3, and now the bevel for the VFX logo looks blue.

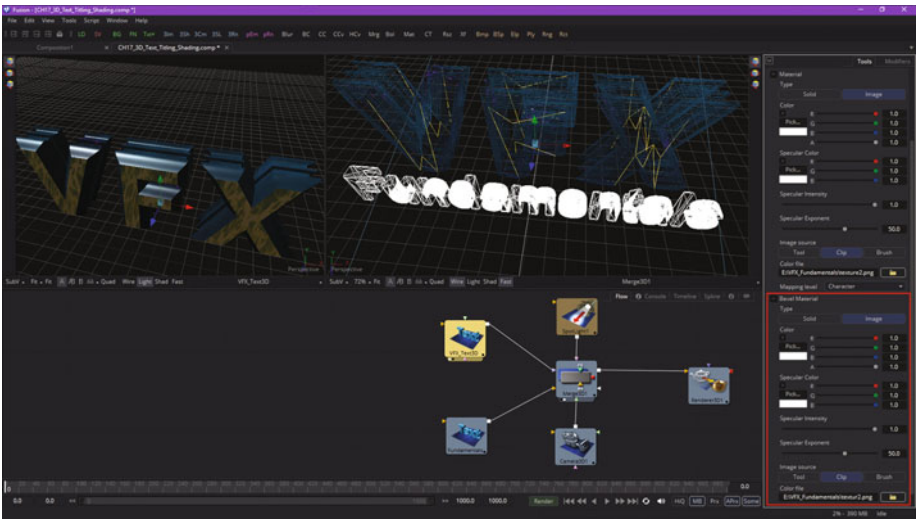

Figure 17-3. Add a blue image texture map to the Bevel Material section

Notice the name for my second texture map is **`textur2.png`** and not `texture2.png`. My reason for doing this is that Fusion won't see `texture2.png`, as it sees it as the second image in a series, and so I renamed it, to remove the *e*, at which time the second texture loaded and mapped correctly.

Let's use the *Fundamentals* portion of this logo to look at the other type of texture mapping, which uses color values instead of image (texture) maps to define diffuse and specular color. I used pink for the faces and yellow for the bevels, and I left the specular highlights as white, since the spot light is also white in color value. I have highlighted these color value settings as well as the deselected "Use One Material" option in Figure 17-4, using red.

Figure 17-4. Set a pink diffuse color and yellow specular color

Next, let's take a look at animating this text titling.

3D Title Animation: Using Keyframes and Modifiers

Fusion 8 seems to have made a slight change in accessing modifiers, which used to be accessed by right-clicking in the Styled Text dialog itself. As you can see in Figure 17-5, doing this now will give you the correct **Cut-Copy-Paste-Delete-Select** context menu, and so you now need to right-click on the **Styled Text GUI** label itself, in order to access this modifiers context menu, as is shown in the far left and center sections of Figure 17-5. I also show the **Layout** tab and **Frame** button I selected on the far right side in Figure 17-5.

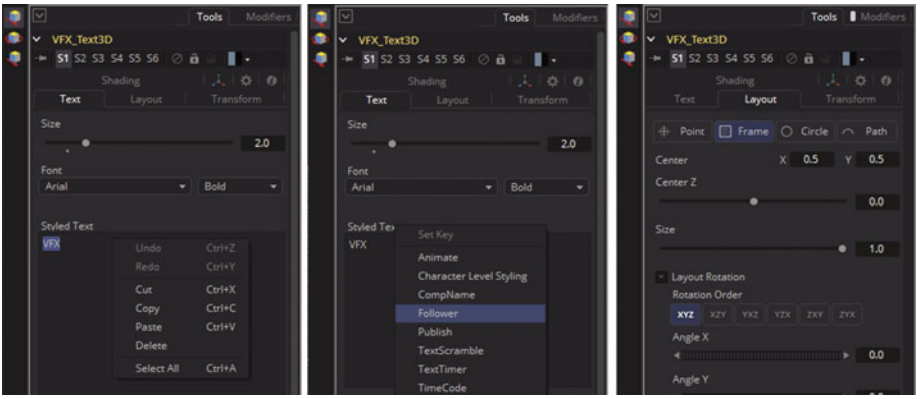

Figure 17-5. Right-click on your Styled Text label to open menu

The **“Frame”** option allows you to treat each character as a framed 3D object. If you select the **“Point”** option, one of the vertices (points) used in a 3D text will be used as a keyframing reference. You can also use the **“Circle”** or **“Path”** options to reference 3D text.

The Follower modifier only works for the **Text** and **Text 3D** tools. Select the “V” character, shown in View2 in Figure 17-6, and then the Timeline tab so you can start keyframing. Then, right-click on the **Styled Text** GUI label and select the **Follower** modifier.

Figure 17-6. *Select the Follower modifier for your 3D titling*

The Fusion Follower modifier contains algorithms that allow you to implement motion graphics effects, which is why you will be using it during this chapter on 3D text titling effects and animation. A Follower works on the premise that you can animate just one character and its parameters, and the other characters in that 3D text object (or a 2D text object) will then mimic or follow those animation characteristics, usually with some delay applied using, you guessed it, a **Delay** parameter.

As you have seen in Figure 17-6, the modifier is applied by right-clicking on the Styled Text field’s label for the 3D text tool (or text tool) and then selecting the “Follower” option. You will access the Follower modifier algorithm settings using the Modifiers tab, and its Timing, Text, Layout, Transform, and Shading Tabs. Click on the **Modifiers** tab, shown circled in red at the top right of Figure 17-7, and go to the **Shading** tab, then open the **Transform** section, as is also seen in Figure 17-7. Right-click on the **Offset Z** control and select **Animate** so that we can make the *VFX* part of the logo zoom in from the horizon. Make sure to set the timeline’s playhead to **0**, as is shown in Figure 17-7.

Figure 17-7. *In the Shading tab within the Modifier tab, go to the Transform section, and animate Offset Z*

The animation for the characters in your 3D text object is done using controls in these Timing, Text, Layout, Transform, and Shading tabs. Note that changing a value in these tabs will have no result at all unless you have enabled Fusion's **Animate** function. The data value needs to be animated in order for these effects to produce the animated 3D text titling effects we're creating.

Pull the **Offset Z** slider to the left, and watch your *VFX* logo disappear into the distance. Leave a keyframe value of **50**, or greater if you like, on frame **0**, as shown in Figure 17-8.

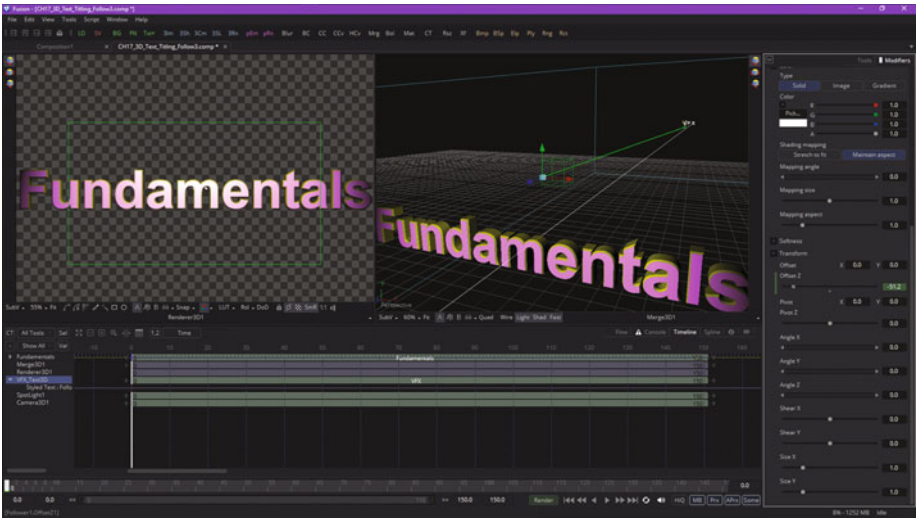

Figure 17-8. Drag Offset Z slider to the left, to a value of 50

Next, set the playhead at **frame 150**, as shown in Figure 17-9, and set the **Offset Z** back to **0**, bringing your logo in.

Figure 17-9. Set the playhead to frame 150; set Offset Z to 0

Notice that the V is now animating in, and the F and the X are now following 12 frames behind (each character), which—as you can see in each of the views in Figure 17-9—looks great.

This is because I set my initial timing delay to a value of 12, which equates to 12 frames. To space your follow closer, use a lower data value for the Delay slider.

A Follower modifier's **Timing** tab contains all of the timing-related parameters, including the **Range** parameter, which specifies whether characters should all be influenced, or if only a selected character range should be influenced. You can even marquee (drag to select) any range of characters directly on the screen and specify a range.

An **Order** parameter will determine the order in which the characters will be influenced. Notice that empty spaces are all counted as if they're visible characters. Available options are left to right, which we're using, and which causes a procedural animation to progress from left to right relative to included characters. Right to left does the opposite of this. Inside out will animate symmetrically from a character's center point toward the outside. Outside in animates symmetrically toward a center point. Random but one by one will be applied to randomly selected characters, only influencing one character at a time.

Completely random animates randomly selected characters and influences multiple characters at a time. Manual curve allows a character's behavior to be specified with sliders.

The **Delay Type** parameter should determine what type of a delay will be applied to the animation. The two options include "Between Each Character," where the more characters there are for your text string the longer the animation will take to the end, and a "Between First and Last Character" option, where, no matter how many characters are in your text, the animation will always be completed in your specified time duration.

Finally, Figure 17-10 shows a visual effect of different settings of the **Delay** slider, which spaces the character follow effect closer or father apart based on a numeric value. If you want to use even numbers, click on either side of a slider dot.

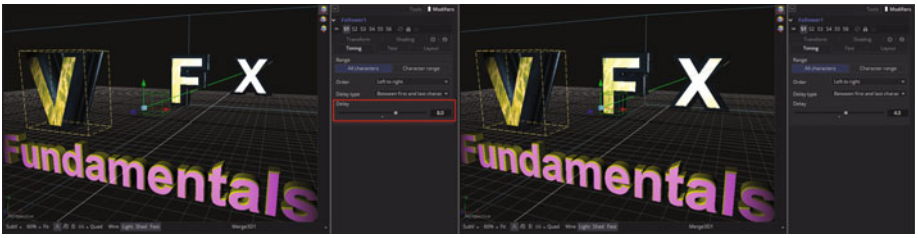

Figure 17-10. Use the Delay slider to control Follower spacing

I tried settings of 4, 8, and 12, all of which worked fine for this visual effect. Figure 17-11 shows that I had to set the end range, which I had set previously to 150, to 154 due to the fact that the Follower modifier slowed the arrival of some of my characters, which didn't fully come to rest until frame 154, so I had to extend the 3D animation range for the renderer 3D node.

Figure 17-11. Extend the timeline animation range to frame 154

I also showed the timeline legend on the left, which will show the "path" to your animated element, which is **VFX_Text3D ➤ Styled Text ➤ Follower1 ➤ Offset Z**.

Next, let's take a look at how to create complex **compound animation** by adding other animation channels to your timeline.

Complex Animation: Animating Multiple Attributes

Since we are looking at 3D text titling animation in this chapter, let's add some more channels, or tracks, to our timeline editor view, and have our V letter spin around three times as it comes in from the horizon, for an even more impressive visual effect. This would be done by using the **Shading > Transform > AngleY** track, as you can see, highlighted in red, on the left side of Figure 17-12. As you know from the previous section, this is done by right-clicking an **AngleY** parameter in the Shading tab's **Transform** section and selecting **Animate**, with your timeline set to frame **0**. Enter **-1080**. to set your keyframe, then drag your playback head to frame **150**, and set the AngleY data value to **0**. When you drag the playback head back and forth, your V will now rotate!

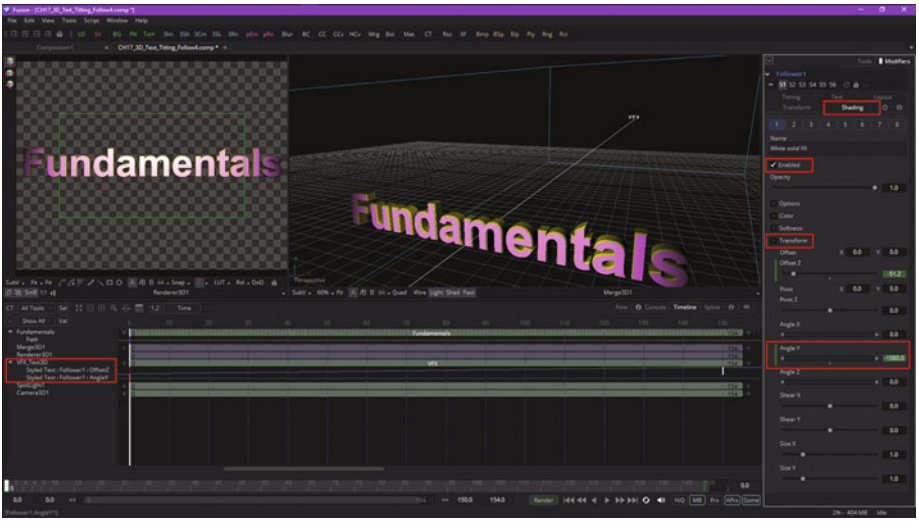

Figure 17-12. Animate the AngleY parameter, from -1080° to 0°

As you can see in Figure 17-13, the Follower modifier is also following this Y rotation, lagging four frames, using the other two characters. The 3D text titling sequence is far more impressive than it was before due to this Y rotation.

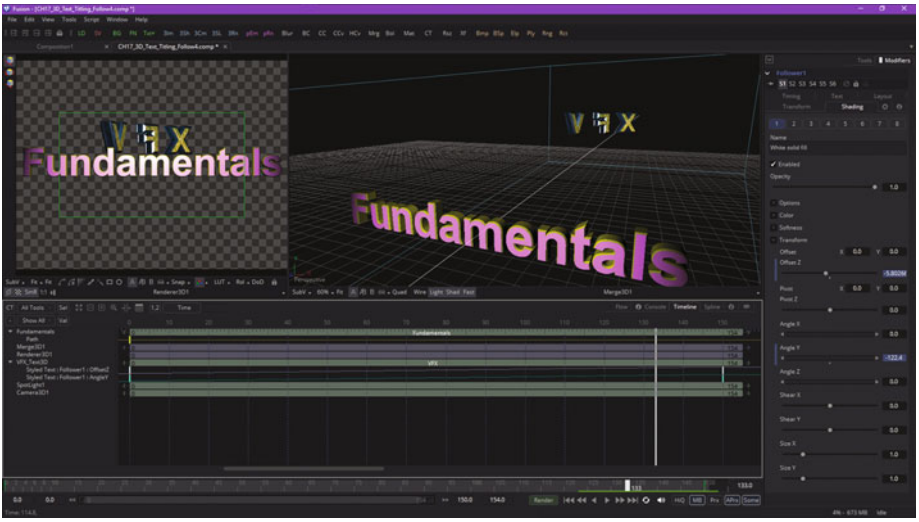

Figure 17-13. Drag the playback head to frame 133 to see a rotation

As you can see in Figure 17-14, the V lands on frame 150, in the correct position and rotational declination, while the F and X characters continue to rotate in until Frame 154.

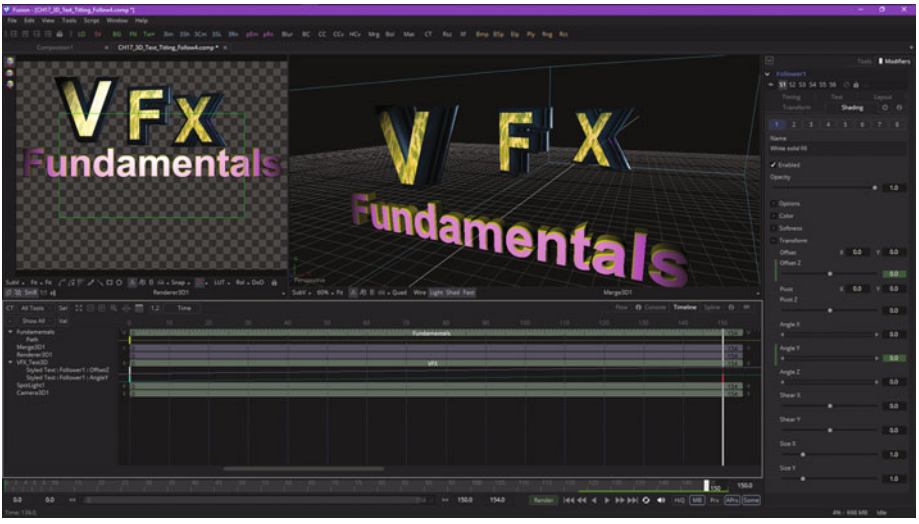

Figure 17-14. The V character finishes animating at frame 150

Let's make this even more realistic by **fading in** the *VFX*, by animating the Opacity parameter, as is seen in Figure 17-15.

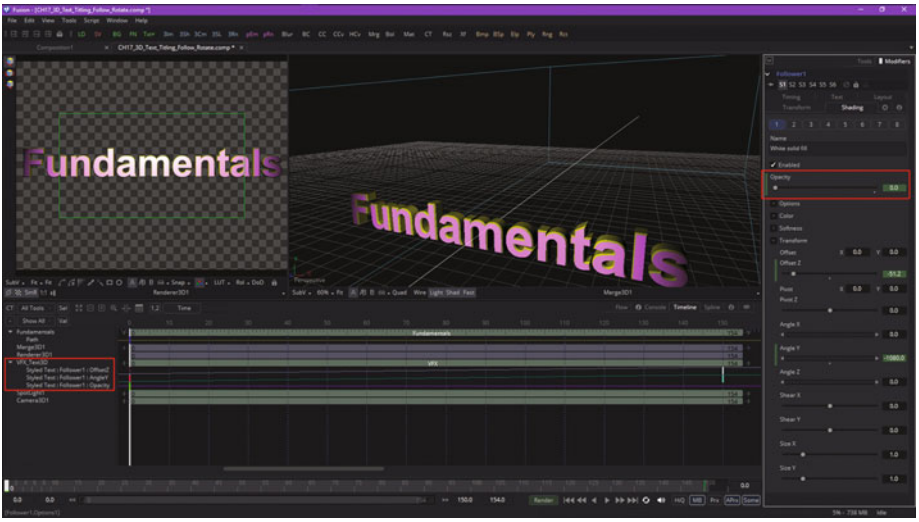

Figure 17-15. *Animate the VFX Opacity parameter, from 0% to 100%*

As you know by now, the work process seen in Figure 17-15 is to set the timeline to frame **0**, right click on **Opacity** and select **Animate**, and set **Opacity** to **0.0**. Next, drag the playhead to frame **150** and set Opacity to **1.0**. Drag the playhead back to frame 0, and watch the translate, rotate, and fade-in effects happen all at the same time. Whereas the move and rotate effect works well using linear animation curves, or splines, the fade in needs to happen more slowly in the distance, and faster in the foreground, so that the fade can actually be seen happening when the 3D text is visible, as shown in Figure 17-16, after the animation curve has been adjusted using Bezier splines in Fusion's spline editor.

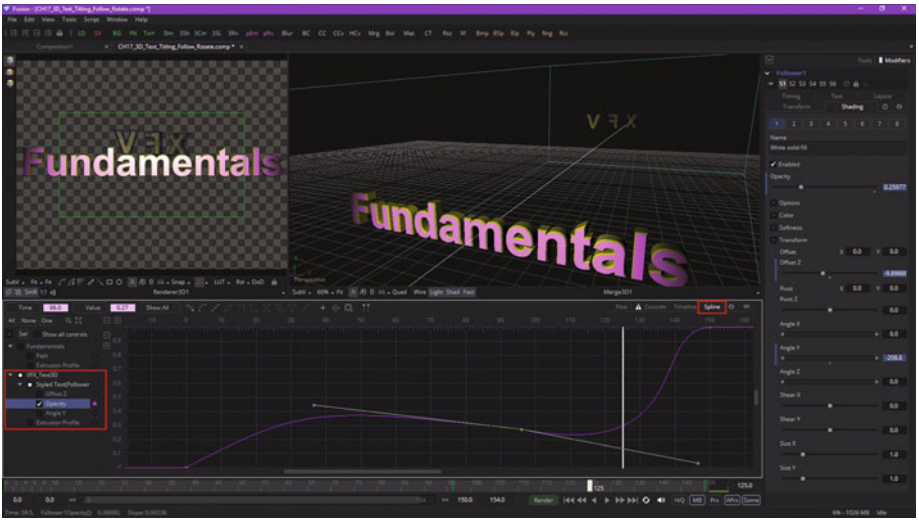

Figure 17-16. *Use a spline editor to make the fade in more visible*

You can add even more Follower animation to this 3D text titling, if you want more practice using the Fusion Follower modifier.

Summary

In this seventeenth chapter, we took a look at 3D text titling animation, modifiers, and texture mapping. We spaced out the *VFX* logo to fit the titling project better and prevent overlap, and texture mapped it using both imagery and color values. We added a Follower modifier and keyframe values to create basic titling animation, using two keyframes to create a robust result. I then created complex animation by keyframing rotation and opacity.
In Chapter 18, you'll learn about Fusion's powerful **3D particle systems**.

Chapter 18

Advanced VFX Pipeline Effects: 3D Particle Systems

Now that you have an understanding of the fundamental Fusion 8 VFX 2D and 3D features, it is time to get into some more advanced 3D effects topics, before I end the book with a chapter discussing how your VFX content can be published across popular devices. Some of the most powerful tools in both 2D and 3D VFX are **particle systems**, which allow you to control thousands of individual objects acting in concert together to create special effects. Particle systems are powerful as well as advanced, and often involve several areas of physics, which we will cover in Chapter 19, as this is a complex topic that requires two chapters.

In this chapter, we'll look at how particle systems work, using "emitter" algorithms that emit and distribute particles, along with particle effects processing algorithms, which then affect each particle's life, trajectory, scale, opacity, speed, spin, collision, bounce, drag, and similar physical characteristics.

Common Controls: 3D Particle Attributes

There are some global particle parameters that apply to each particle system (and particle physics) tool in Fusion 8, and it is logical for us to cover these first, as they clearly represent an overview of many of the core attributes of a particle system, and thus they are a great place to start. Each particle tool will have its own primary **Controls** tab with the parameters that are unique to that particle tool, as shown circled in red in Figure 18-1. Emitters will

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_18

have a **Region** tab and a **Style** tab, as shown in Figure 18-1, and effects will have a **Region** tab and a **Conditions** tab that contains the conditions for applying the particle effect to the particle emitter. We will cover the Effects (Conditions) tab after we cover the Emitter tabs. Before we start a project, we will discuss the particle render node, which is similar in function to the renderer 3D node tool that renders 3D scene components.

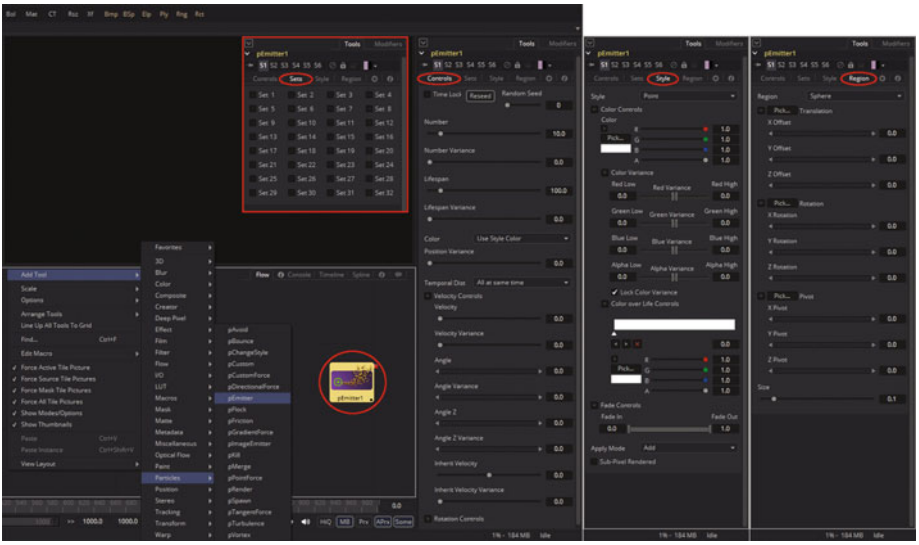

Figure 18-1. *Four control panel tabs used for particle emitters*

Let's look at the particle **Region** tab first, as this tab will allow you to confine the placement of the particle system.

Particle Regions: Where Your Particles Will Exist

Particle regions can be used to restrict a particle tool effect to a 3D (geometric) region or to a 2D (planar) region. It is used to determine the area in which particles will be created using a particle emitter (pEmitter) node tool. Fusion has seven types of regions, each featuring its own control panel layout, accessed using the Region drop-down selector, shown in the Region tab on the far right side in Figure 18-1. The **All** region will allow particle to be created everywhere; for 2D mode, the particles will be created anywhere within the boundaries of the image, and in 3D mode this region describes a cubic area.

The **Bézier** region uses a user-generated polyline to tell the particle emitter where to create the particles. Bézier mode will work in both 2D and 3D particle system modes. Remember that the Bézier polyline region will be created in 2D space. In order to animate over time a shape created using

a polyline, or to connect a polyline to another polyline, you'll need to right-click on the Polyline label, located at the bottom of the control, to select an appropriate option from the polyline context-sensitive menu.

Bitmap data output from one of your 2D tools in the node flow can also be used for a region where particles are emitted.

3D cubic geometry can also be used to determine a region within which particles are created, as can be seen on the right in Figure 18-1. The height, width, depth, and X, Y, Z positioning can all be determined by a VFX designer and be animated over time.

3D sphere geometry can also be used to determine regions in which particles are created, as can be seen on the right in Figure 18-1. The sphere X, Y, Z offset, pivot-point location, and rotation can all be determined by a VFX designer and be animated over time.

The **line** region allows you to determine where a particle or particles should be created. This line should be composed of two endpoints, which can be connected to paths or trackers if necessary. As with the Bézier region, this region type will work in 3D, but the line itself will only be created and adjusted in 2D space.

A **3D mesh** region can also be used. In mesh mode, a region can also be restricted by using the Object ID slider. FBX model import with FBX mesh 3D is the norm here, but alembic 3D models (vertex point cloud data) can also be used, with the alembic 3D mesh node tool usage.

The **rectangle** region is like a cubic region, except that the rectangle region has no depth (that is, no Z coordinates). Unlike other 2D emitter regions, this region could be positioned and rotated in Z space, allowing for some impressive "2D within 3D" visual effects. Next, let's take a look at particle styles.

Particle Style: Particle Color, Fade, and Lifespan

The particle Style tab can be found in the particle emitter, particle spawn, particle change style, and particle image emitter. In the Style tab, both the type and the appearance of particles are determined. The Style tab provides controls that affect the appearance of your particles, allowing the "look and feel" of particles to be tweaked by a VFX designer. This can even be animated over time.

A **Style** drop-down menu allows you to define the **particle style type,** providing settings for different types of particles supported by Fusion's Particle Tool Suite. Each style type will feature its own specific particle controls, as well as controls that it shares with other style types.

A **Point** style type is the most commonly used type of particle and produces particles that are one pixel in size; thus, processor power is highly optimized. Controls that are specific to Point style include Apply Mode (blending) and Sub-Pixel Rendered.

Both the **Bitmap** and **Brush** style types generate particles based on image data. The Bitmap style type uses image data from another tool in the node flow, and the Brush style type uses TGA (Targa) images that are kept in the `Fusion8/Brushes` directory.

A **Blob** style type creates large, soft, spherical particles with Color, Size, Fade Timing, Merge Method, and Noise controls.

The **Line** style type creates straight-line type particles with an option to set a **falloff** value. A **Size to Velocity** control described under Size Controls is often used with the Line type. A **Fade** control adjusts the amount of falloff spanning the length of your line particle type.

A **Point Cluster** style type creates small clusters of one-pixel particles. The Point Cluster style is similar to the Point style, and is more memory efficient if large quantities of particles are required. This style type will share controls with the Point style. Additional controls specific to a Point Cluster style type are Number of Points and Number Variance.

The **NGon** style type is new in Fusion 8, and gives you an NGon generator where you can select the **number of polygon sides** (hence NGon) and how star-shaped an NGon may be. This is termed NGon Starriness in your Fusion NGon control panel. There are also NGon type presets, including Circular, Circle, NGon Solid, NGon Star, Soft Circular, NGon Shaded In, and NGon Shaded Out.

The Style tab's **Color Controls** section features RGB **color** controls, which allow you to select RGBA color and alpha values for each of the particles generated by the emitter. You can set a range of colors as well using the Color Variance section, which contains four Low–High dual-range sliders for each RGBA value.

For example, setting a red variance range of -0.2 to +0.2 will produce colors that vary 20 percent on either side of the red channel value set in the color picker just described. To show a color space as 8-bit values (between 0 and 255) or as 16-bit values (between 0 and 65,535), change the values used by Fusion by using the **Show Color As** preference. This can be found in the **General** tab in the Fusion 8 **Preferences** dialog.

The **"Lock Color Variance"** checkbox locks color variance to your particles. Unlocking this allows your color variance to be applied differently across each color channel, yielding a wider range of color variation.

The **Color over Life Controls** section contains a standard color gradient widget that allows for a selection of a range of color values. The particles will then conform their coloring to this gradient color-stop collection during their life span.

The left end on a gradient represents the particle color at birth, and the right end shows the color of your particle at death. Additional points that are added to the gradient widget would cause a particle color to shift multiple times throughout its lifespan. This type of control can be used for explosions and other fire-related effects. A gradient widget can be animated over time (right-click on the control and select Animate). The timing for color stops on a gradient will be spline-controlled.

A single Color Over Life spline controlling the speed at which your gradient itself changes can be edited in the spline editor. You could also right-click on your gradient widget and then select a From Image modifier, which will create a gradient from the range of colors in an image specified using a **straight line between two points within that image**. Pixel colors under that line will form the color gradient color stops.

The **Size Controls** section of the Style tab contains size sliders as well as the **Size over Life** spline editor graph. The **Size** and **Size Variance** sliders are used to specify the size, and the amount of size variation for each particle. Point style doesn't need size control, as each point is a pixel, and its size is controlled by the pixel pitch (size) of each user's display.

If the Bitmap particle style is used, a value of 1 indicates that each particle should be exactly the same size as the input bitmap resolution. A value of 2 should scale the particle up in size by 200 percent. For the best particle visual quality, try to make the input bitmap as big as the largest particle produced by the particle system. Remember from the theory Chapter 2 that down sampling by 2X or 4X (or no resizing at 1X) creates the best visual quality.

For a Point Cluster style type, the size control adjusts the density of the cluster, or how close together each particle will get. Essentially, this specifies the size for the cluster.

There are also size sliders that are used to adjust size for the particles based on the velocity or Z depth. The **Size to Velocity** increases the particle size relative to a velocity, or speed, of a particle. The velocity of the particle will be added to its size and then scaled by the value of the slider, in case you are wondering how this particular algorithm functions. If a 1.0 is specified using this control, and you have your particle traveling at 0.25, this algorithm will add another 0.25 to your size (Velocity times SizeToVelocity plus Size equals New Size). This control will be used to adjust the size of any style type, but it is the most useful for Line styles.

A **Size Z Scale** slider specifies the degree to which a size for each particle would be increased or decreased according to its depth—that is, its position in Z space. The effect of this will be to exaggerate (or reduce) the impact of the perspective. The default value is 1.0, which provides a relatively realistic perspective effect. Objects on the focal plane (Z = 0) would be actual size, and objects farther along Z would become smaller, while objects closer along the Z axis would become larger.

A **Size Over Life** spline control graph allows you to draw a spline that determines the size of a particle throughout its life. The vertical scale represents the percentage of the size control, and the horizontal scale represents a percentage of the particle's life. It is also possible to view and edit the spline graph in the spline editor, just like you did in Chapter 16.

The **Fade Controls** section contains a simple range slider that provides a control for fading a particle at the start and at the end of its lifetime using the **Fade In–Fade Out** dual-range slider widget.

Increasing the Fade In value will cause the particle to fade in at the start of its life. Decreasing the Fade Out value will cause the particle to fade out at the end of its life. The control values represent a percentage of the particle's overall life; therefore, setting Fade In to 0.1 would cause a particle to fade in over the first 10 percent of its total lifespan.

The **Merge Controls** section contains two particle sliders that affect the way individual particles are blended together. The Subtractive–Additive slider allows you to set a blending range from subtractive to additive, and the Burn-In slider will cause particles to over-expose, or become blown out, after they are combined. None of the merge controls will have any effect when used in a 3D particle system, as blending modes are a 2D compositing phenomenon.

The **Blur Controls [2D]** section of the Style tab contains a set of seven particle controls that would be used to apply a blur to individual particles. Blurring can be applied globally, by age, or by Z-depth position. None of the blur controls would have any effect when used in a 3D particle system.

The **Blur [2D]** and **Blur Variance [2D]** sliders will apply blur to each particle independently before these particles are merged together. The Blur Variance slider randomizes the amount of blur applied to each particle to add realism to the VFX. The **Blur Over Life** spline graph controls the amount of blur that is applied to a particle over its life. The vertical scale represents the percentage of the blur control, while the horizontal scale represents a percentage of the particle's life. It's also possible to view and edit the spline graph in the spline editor, just as you did in Chapter 16.

A **Z Blur (DOF) [2D]** slider applies blur to each particle based on its position along a z-axis. A **DOF Focus Range** slider can be used to determine how the degree of focus is calculated for the Z blur. Lower values along Z are closer to the camera, and higher values are farther away. Particles within the range that you set in the DOF Focus Range slider will remain in focus, and particles outside the range will have the blur defined by the Z blur control applied to them.

Different particle style types will have different **Apply Mode** settings. For example, Point or Point Cluster style types have Add or Merge, which can be applied to 2D particle systems. With **Add**, the overlapping particles are combined together using a blending algorithm that sums color values for each particle. With **Merge**, any overlapping particles will be merged together.

The Point or Point Cluster style type **"Sub-Pixel Rendered"** checkbox determines if a point particle would be rendered using sub-pixel precision, which provides smoother-looking motion but blurrier particles, which will take longer to render.

The Point Cluster style type features a **Number of Points** slider, which determines how many points are in each point cluster, and a **Variance** slider, used to randomize this value.

The Bitmap style type has more complex options, including an **Animate** drop-down widget with options for **Over Time**, **Particle Age**, and **Particle Birth Time**. There are also sliders to set **gain** as well as **time scale** and **time offset** data values. A Brush type also has a Gain slider, used to apply corrections to an overall particle gain, or contrast against the background content (backplate, in our case, Toronto). Note that this parameter can be used with imagery used as a bitmap style or as a brush style. High Gain values produce brighter particle images (or brushes), while a low Gain value will reduce both the brightness and the transparency for the image or the brush.

The Brush type also has a **Brush** drop-down to select brush images, and a **Use Aspect Ratio From** drop-down with settings for Image Format, Default Frame Format, and Custom.

The Blob style type has a **Noise** slider, which introduces a grain-type noise into the blobby particle system.

The Line style type features a **Fade** slider, which adjusts the translucency falloff over your line particle's length. The default value of 100% (1) causes a line to fade out completely by the end of the length.

Particle Sets: Grouping Particles Together

Particle sets allow you to manage individual groups of particles, much like using a material ID to assign different texture map attributes to different faces on a polygonal mesh. The Sets tab is shown in both Figure 18-1 and Figure 18-2, as it has to be available to both emitters (to specify as a set) and effects (to specify what particle sets to process). This allows complex particle systems to be assembled that mix particles together in 3D space or 2D space but allow you to control sets of particles in such a way that they do not affect each other, but rather co-exist together in the same particle physics simulation. We will take a look at using particle sets in the simulation that we create in this chapter and the next, which covers advanced physics. As shown in Figure 18-1, Fusion provides 32 different particle sets that you can utilize to control the configuration of individual particle system dynamics. It's important to learn how to leverage particle sets, just like it's important to learn how to leverage material ID for texture map creation and layer grouping for use in digital image compositing pipelines. If you are interested in learning more about image compositing, check out *Digital Image Compositing Fundamentals* (2015) at www.Apress.com.

Particle Conditions: How Particles Will Be Affected

Particle effects, such as the directional force effect we will be using during this chapter, feature a **Conditions** tab. This tab contains parameters that specify how that particle effect algorithm will be applied to the particle emitter before it is sent to the particle rendering engine, which we'll cover next. I added a directional particle node and wired it to the emitter, as shown in Figure 18-2, which opened the particle effect panels, which we've covered (controls, sets, and regions), and the Particle Conditions tab, which I'll cover in this section of the chapter.

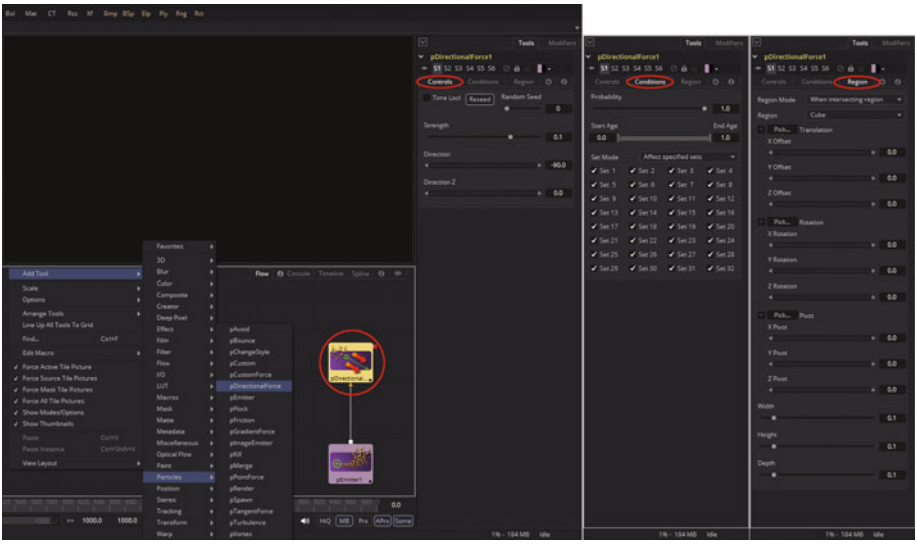

Figure 18-2. Three control panel tabs used for particle effects

A **Probability** slider sets a probability as a percentage chance that the effect will affect a particle. A default value of 100% (1.0) means all the particles will be processed by that effect algorithm. Probability would be calculated for particles on each frame. A particle that is not affected by an effect for one frame has the same chance to be affected in the next frame.

The **Start Age**–**End Age** range slider is used to conform the particle effect being applied to the emitter to a specified percentage of each particle's lifespan. For example, to restrict the effect to the final third of a particle's life, set a start value of 0.67 while the end value remains at 1.0. The effect will be processed on frames 67 through 100, or the final third of the particle's life. The **Set Mode** specifies how an effect should be applied (all particles or include or exclude the selected set).

Particle Renderer: Making Particle Systems Visible

In order to see particle emitters and effects, you will need to add a **pRender** node downstream from the other particle nodes, as shown in Figure 18-3. Particle render is where you specify the particle system dimensions, either 3D or 2D—which, as you have seen, affects the usage of blend modes. A pRender tool converts the particle system to either an image (2D) or geometry (3D). The default is a 3D particle system, which must then be connected to a renderer 3D node to produce an image. This allows particles to be integrated with the 3D scene before they are rendered along with the other 3D scene components.

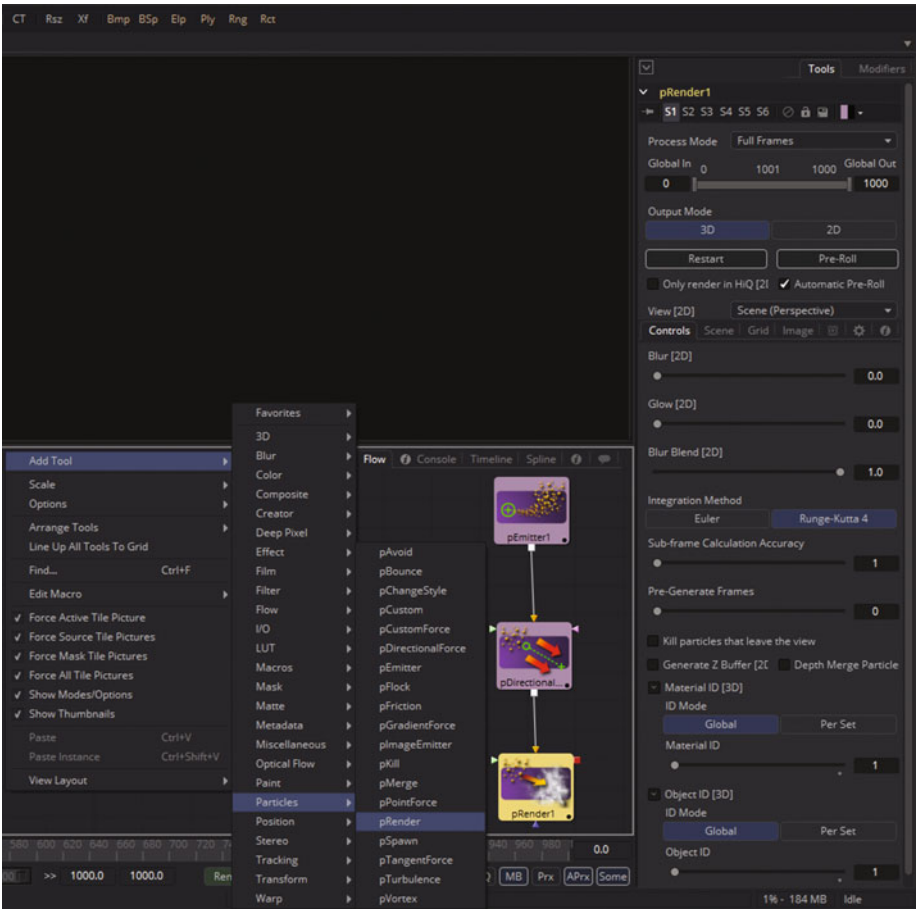

Figure 18-3. *Particle rendering menu sequence and control panel*

Output mode buttons allow you to select between **3D** or **2D** output. While the pRender defaults to 3D output, it can be made to render a 2D image sequence. If pRender is not connected to a 3D-only or 2D-only tool, you could also switch it by selecting **View ➤ 2D Viewer** from its display view context-sensitive menu.

For 3D mode, the only settings in the pRender tool that have an effect include Restart, Pre-Roll, or Automatic Pre-Roll. Sub-frame Calculation Accuracy and Pre-Generate Frames can also be used in 3D mode. All the other settings affect 2D rendering.

The pRender node has a camera input on its node, allowing the connection of a camera 3D node tool. Camera nodes can be used in both 2D and 3D modes to control the **viewpoint** used to render the output image for your particle system VFX pipeline, just like a real camera controls your view (point).

When a pRender tool is selected and its view dot is set, all the in-view control widgets for particle tools connected to it are seen in the display view. This provides a visual view of the forces being applied to the particle system as a whole.

The **Restart** and **Pre-Roll** buttons are used to tell a particle system algorithm the position of each particle on prior frames so that it can calculate the effects of the forces applied to them on the current frame. The controls found here are used to help accommodate this, providing methods for calculating intervening frames. Clicking on the Restart button will restart your particle system at the current frame, removing any particles created up to that point and starting the particle system from scratch at the current frame. Clicking on the Pre-Roll button will cause the particle system to recalculate starting from the beginning of a render range up until the current frame. It will not render the image produced by this calculation; Pre-Roll can only calculate the position for each particle. This provides a way of ensuring that any particle displayed in your viewports will be correctly positioned. If a pRender tool view is on when the Pre-Roll button is clicked, the progress of the pre-roll is seen with particles being shown using point style to save on system-processing overhead.

The **Automatic Pre-Roll** selection tells a particle system to automatically pre-roll particle calculations to support your current frame whenever the current frame changes. This prevents your needing to manually click the Pre-Roll button whenever advancing through time in increments larger than one frame. This progress of a particle system during automatic pre-roll is not displayed in your pRender view so as to prevent visual workflow interruptions.

There are **post-render output** settings for applying blur, glow, and blur blend as well as settings for integration method, subframe calculation accuracy, and pre-generating frames. There are options to **kill** particles that leave your view, death merge particles, and to generate a Z buffer for your particle system. The **Blur**, **Glow**, and **Blur Blend** sliders apply Gaussian blur, glow, and blending to the 2D particle system image as it is rendered.

This can be used to soften 2D particles or to blend them together. Your result is the same as adding a blur node after a pRender tool in the flow node editor.

The **Sub-Frame Calculation Accuracy** slider will specify a number of sub-samples taken between frames when calculating the particle system. High values will increase the accuracy of your particle-system algorithm's calculations and will increase the length of time that it will take to render the particle system.

The **Pre-Generate Frames** slider tells the particle system to pre-generate a specified number of frames before your first frame. It is used to give your particle system an initial state from which to start calculating its particle system algorithms.

Clicking the **"Kill particles that leave the view"** checkbox would kill all particles that leave the visible boundary for a 2D image. This can be used to speed up render time. Particles destroyed with this option will not return, so make sure this is what you want to do for your VFX particle simulation objectives.

Selecting the **"Generate Z Buffer"** checkbox will cause the pRender tool to produce a Z buffer channel for your image. The depth of each particle is represented in the Z buffer. This auxiliary channel can then be used for 2D depth-compositing operations like depth blur, depth fog, or downstream Z merging. This option increases rendering time for a particle system, as you probably surmised.

Clicking the **"Depth Merge Particles"** option causes the particles to be merged together using depth-merging algorithms rather than using layer-centric blending techniques.

Particle System Project: Adding Core Nodes

Assuming you have followed Figures 18-1 through 18-3 and added **pEmitter**, **pDirectionalForce**, and **pRender** nodes using the **Add Tool ➤ Particles** context menu, let's add an image next that we can composite the particle system over, since many times that is what you are going to want to do for a VFX project pipeline. I used `pexels.com` to find a nice, free for commercial use image of the Toronto skyline taken by photographer Amarpreet Kaur. I used an **image loader** to load this into my project and an **image saver** to save out the particle system animation. We will add in other nodes as we create this project over the next couple of chapters. Figure 18-4 shows the image I chose on `pexels.com` by clicking on it. We can use Fusion tools to adjust this image as needed to fit our particle system animation, adding nodes into our VFX pipeline as we go. This is one of the most impressive aspects of Fusion 8's visual-programming paradigm—being able to build and refine a VFX project, and its visual results, by adding new nodes and wiring them together.

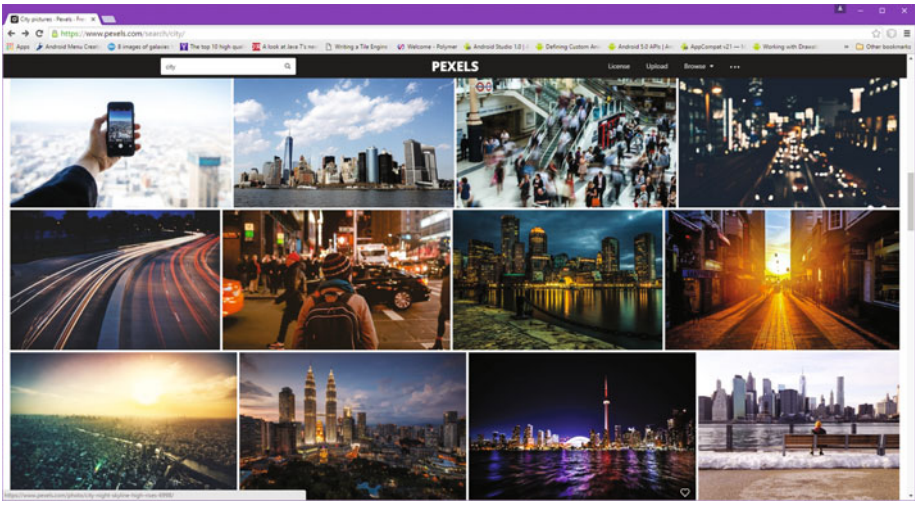

Figure 18-4. Go to pexels.com and find a good image to use as a background plate

Click on the HD image you want to use, and download it to your project folder; in my case, this was `VFX_Fundamentals`. The work process I used is shown in Figure 18-5, on the right side.

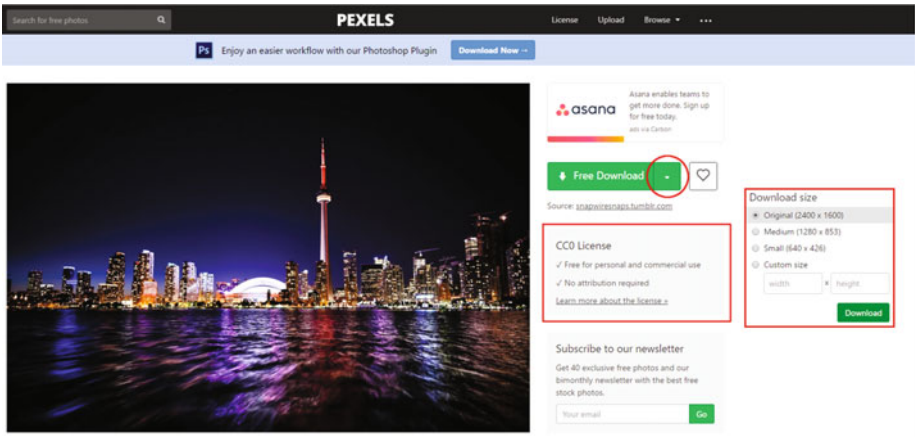

Figure 18-5. Download a royalty-free, HD, high-resolution image

Next use **Add Tool ➤ Composition ➤ Merge** to add a node so as to composite your particle system's foreground over your cityscape background. Wire the cityscape loader into the merge node, wire the merge output into a CityScape.MP4 saver, and wire a **pRender** output into the **Merge1.Foreground** input, as seen in Figure 18-6.

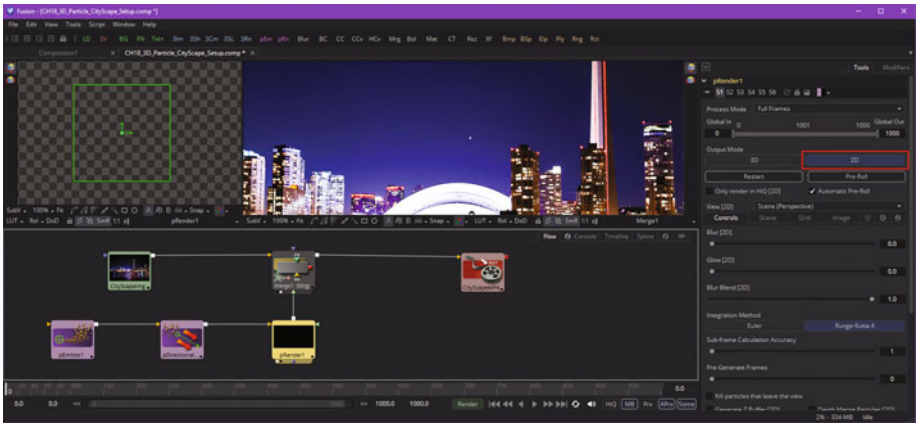

Figure 18-6. Add an image loader, image saver, and a merge node

Next, set a **Number** value of **1** (also pick a **Random Seed**). I started out with **Lifespan** at **50** and a **Velocity** of **0.15**. I set an **Angle** value of **90°** and an **Angle Variance** and **Z Variance** of **10°**, as shown on the right in Figure 18-7, highlighted in red.

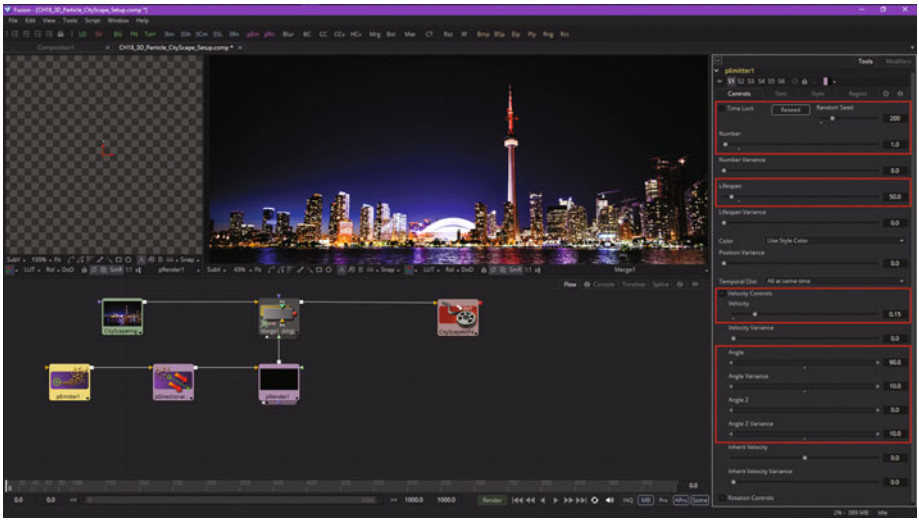

Figure 18-7. Specify the starting pEmitter particle control settings

Next, select your **Region** tab, and select **Line** from the **Region** drop-down menu, as shown in red in Figure 18-8.

Figure 18-8. Specify the line emitter region, and place it on the Toronto City Waterfront (beach)

Place the line right on the Toronto waterline, or beachfront, as shown by widgets on the left and right side of the image in Figure 18-8. I set a particle type of **Brush** in the **Style** tab, shown in Figure 18-9, so I could see particle trajectories and placement, as points were too small (due to sub-pixel rendering) on this 2400 x 1600 image.

Figure 18-9. Specify the Brush particle style so that you can tweak its parameters

You may have noticed in Figures 18-8 and 18-9 that there is a resolution disparity between our 2400 x 1600 image and the particles' rendering view, which is labeled with an HDTV 1920 x 1080 resolution. Clearly, our particle canvas must match up with our image. I'm getting imprecise results as I play the particle system, and other than my as yet unrefined settings, this would be the most significant reason for that. Select the pRender node, as is seen in Figure 18-10, and conform the width and height to the 2400 x 1600 image. You will see that you have to readjust your Line emitter, which is why the particles were not emitting correctly relative to the backplate image. Once you do this, the pRender preview and merge View2 will match up, the particles will correctly emit from the line emitter placed along the Toronto beachfront, and we are now ready to continue.

Figure 18-10. Synchronize the resolution for the pRender node tile to match backplate image

We'll still need to tweak settings during the entire VFX pipeline creation process, but let's take a look at your pSpawn node, so we can add a trail of sparks from the primary particle.

Particle Spawn: Particles Birthing Other Particles

The **particle spawn** or pSpawn tool can turn each of your particles into its own parent emitter, allowing a particle to produce child particles of its own, which is why it is called Particle Spawn. The original particle will continue to exist until the end of its lifespan, and each of the child particles that it emits (spawns) during its lifespan becomes wholly independent, with its own lifespan, and its own particle characteristics, which we are covering during this chapter.

As long as a particle is processed by a pSpawn algorithm, it can continue to generate child particles. Be sure to conform the settings for this tool to limit particle generation, as new particles generated must then be managed by the particle system, using your valuable CPU power. Settings that can limit runaway child particles include start or end age, probability, sets and regions, and pEmitter parameter animation, so that particles are in play and being operated on by the CPU only when necessary.

Right-click in the flow node editor and use an **Add Tool ➤ Particles ➤ pSpawn** menu sequence to add a particle spawn node to your project, as is shown in Figure 18-11. Also make sure an **end range** has been set in the time ruler. I set **250** (as seen in Figure 18-9) as my particle system duration, at least for now.

Figure 18-11. *Add a pSpawn node, using Add Tool ➤ Particles ➤ pSpawn*

This pSpawn tool has a significant number of parameters, most of which mirror parameters in the pEmitter tool. pSpawn is both an emitter and an effect and, as such, has tabs for Controls, Sets, Style, Conditions, and Region. There are several parameters unique to this pSpawn tool. Let's discuss the new parameters first, before we set our initial parameters.

The **"Affect Spawned Particles"** checkbox, when selected, will cause particles created by this spawn algorithm to become seeds for the spawn tool on subsequent frames. This can exponentially increase the number of particles in the particle system on each subsequent frame of your particle system timeline (mine is 250) and can therefore affect render time significantly. This option should only be used when absolutely necessary in a VFX project.

The **Velocity Transfer** slider specifies how much velocity from the parent particle will transfer to the child particles that it spawns. The default value of 1 causes each new particle to adopt 100 percent velocity and direction from the parent particle. Lower velocity transfer data values should transfer less of the original parent particle's motion attributes to child particles and therefore a value of (or near) zero will create independent child particles, with a mind of their own (if we adhere to the parent-child paradigm we are using here).

Let's set our **Number** of (Spawn) Particles to **5** and **Angle** and **Angle Variance** to **360°**. I used Velocity of **.002** and **Velocity Transfer** of **0.03** so that the parent particle leaves a trail of child particles. As you can see in Figure 18-12, this is working well, but my parent particles are still leaving the screen, so my **Gravity** particle effect parameter is still not set properly. As you can see, creating a particle simulation is a refinement process.

Figure 18-12. *Set pSpawn parameters for Number, Angle, and Velocity*

Figure 18-13 shows the areas that I looked at to get the gravity, which is provided by a **-90° directional force**, to work properly. First, I set a longer particle lifespan and variation, to see if the particles would start to fall. They didn't. I next set the **Region** for this **pDirectionalForce** to **affect all** of the particles in the simulation. That did not work either! So I looked at the **pEmitter Sets** tab, finding that I hadn't assigned any particle set to these particles! After I selected **Set 1**, my gravity worked perfectly, and I was ready to proceed in working on the VFX project. It is best to perfect portions of the pipeline before adding more complexity, which is the work process I am using during this book.

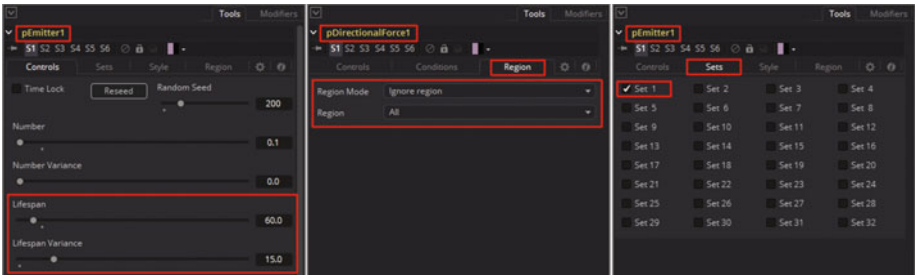

Figure 18-13. Tweaking the settings to get the particle gravity to work optimally

Since Spawn is an **effect** as well as an **emitter**, let's set the **Spawn Sets** tab to only affect these **Set 1** particles. Now we are ready to add more characteristics to the particle system. I want to show you how to create exploding effects, so let's add a second pSpawn node to do this. Wire your **pSpawn2** in between the pSpawn1 and pDirectionalForce1 nodes, and set your **Number** to **250**, **Variance** to **10%**, **Lifespan** to **40**, and **Lifespan Variance** to **20%**. In your **Velocity** section, set **Velocity Transfer** to zero, **Velocity** to **0.125**, **Velocity Variance** to **5%**, **Angle** and **Angle Z** to **0**, and **Angle Variance** and **Angle Z Variance** to **360°**. Leave the **Region** set to **Affect All Regions**.

Figure 18-14. Add a second particle spawn node and set the parameters

Next, let's configure your Set and Condition parameters!

As you can see in Figure 18-15, we are going to set this second particle spawn algorithm to label its particles as **Set 2** so that we can apply different physics attributes, which we are going to be covering during the next chapter, to the explosion and not affect the particle trail created by the first particle spawn algorithm. This is done by using the **Sets** tab, seen on the left side of Figure 18-15. We also want this explosion to occur when the parent particle dies, or just before it does, at a **99%** of lifespan point in time, which is shown in the **Conditions** tab and set using the **Start Age** slider. We also want to use the **Set Mode** of **Affect specified sets** and then set the **Set 1** as the set to be affected, as is shown on the right side in Figure 18-15.

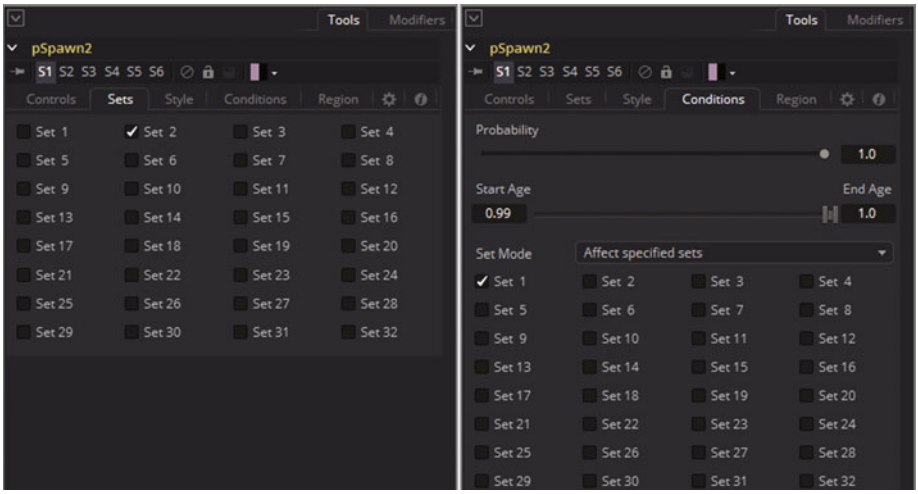

Figure 18-15. Set as Particle Set 2 and set to Affect Particle Set 1

Notice that if we change particle system parameters requiring lots of algorithmic processing, the pRender tile turns into the **progress bar** for particle processing upstream of it, as shown circled in red in Figure 18-16. Now we are ready to tweak our pSpawn2 settings to refine this explosion effect, before we finish up with this chapter.

Figure 18-16. pRender node tile preview progress bar shows particle system processing

Now we have a nice, complex particle system simulation that uses only one primary emitter to create hundreds of sparks, particle trails, and particle explosions, all affected by gravity. We will be covering other physics algorithms in the next chapter. This **pDirectionalForce** algorithm can be used to simulate many physical systems other than gravity, so I am covering this topic in this chapter, since we have a lot more to cover in Chapter 19. As you see in Figure 18-17, your happy face placeholder particles are shooting up and falling back and then exploding as intended, and all we have to do is reduce the number of particles used, and refine some of the variation parameters to get more realism into the particle simulation. I will always try to reduce the number of particles used as much as possible, while still getting the desired visual effect, for data footprint and CPU optimization reasons.

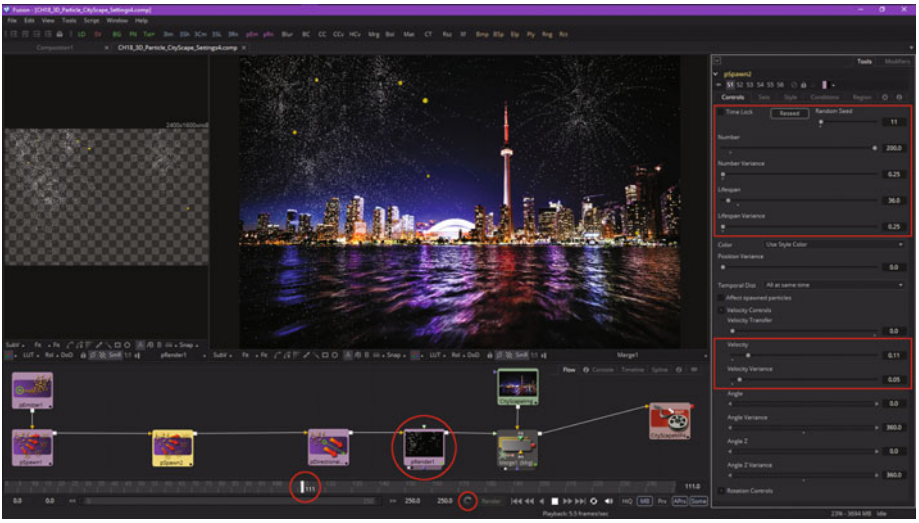

Figure 18-17. Explosions can be seen rendering in the pRender node, View1, and View2

Before we finish up with the chapter, let's take a break from particle systems and add a node or two on the imaging side of our VFX pipeline, to better optimize the background image quality, so as to have good contrast for use with the particle system simulation.

Background Plate Optimization: ColorCorrector Node

I'm going to use a **color correction** algorithm to darken the sky and brighten the lights in the buildings and their reflections on the water. I'm thinking of refining the particle system into a fireworks simulation, and sparks look better in front of a darker sky.

As you can see in Figure 18-18, I have added an **Add Tool ➤ Color ➤ Color Corrector** node, and wired it between the loader and merge nodes,

reconfiguring the node tile flow during the process. We have created an incredibly robust VFX project using only nine nodes here in Fusion 8!

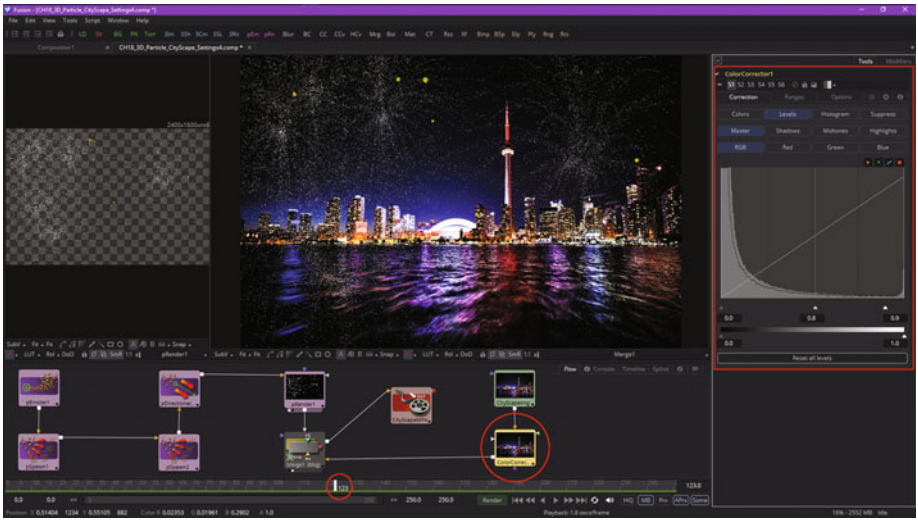

Figure 18-18. *Darken the sky and brighten the color reflections in the backplate using a ColorCorrector Node*

I pulled the middle tones **0.8** to the right to darken the sky, and I pulled the highlights to the left **0.7** to saturate the color reflections on the water, as you can see in Figure 18-18.

I wanted to set up this pipeline now so I could tweak it more later, when the particle system work is complete, and it can be used as a visual reference against what I am doing with this ColorCorrector1 node. That is what is great about how you build your VFX processing pipeline in Fusion 8 using node tool tiles, as you have fine-tuned control over your pipeline, the VFX project it creates, and how the assets and intermediary assets (things like masks and geometry) are displayed in Fusion.

Summary

In this eighteenth chapter, we took a look at particle system dynamics, concepts, forces, emitters, sets, regions, settings, and rendering. We looked at all of the various parameters used to set up emitters and effects, how to render particle systems and composite them over imagery, and how to use sets to combine multiple particle systems seamlessly as if they were one particle system simulation. We looked at different types of emitters and used a brush emitter as a proxy to refine settings. We also got some more practice with digital image adjustment. In Chapter 19, we will look at **particle system physics**.

Chapter 19

Advanced VFX Pipeline Physics: 3D Particle Physics

Now that you have an understanding of the fundamental concepts, terms, and principles of particle systems, let's take a look at some of the advanced physical system tools that come with Fusion to take your particle system simulations to the next level.

We will look at how to apply forces such as friction and turbulence with particle physics algorithms in combination with Fusion modifiers and image and transform processing algorithms.

Particle Physics: Natural Force Algorithm

There are a number of algorithms in Fusion's 3D (and 2D) particle system that apply forces calculated using physics equations. Did you pay attention in high school in physics class? I didn't. Wish I had. Relearning it all is a blast via VFX particle system simulations, I can tell you that. Core physics algorithms (tools) in Fusion include nodes for **pBounce**, **pFriction**, **pTurbulence**, and **pVortex**, as well as forces that can be used to simulate a number of phenomena that we see in nature, including **pDirectionalForce** (gravity), **pTangentForce** (rebounding) and **pPointForce** (directional particle simulations, such as conical spray).

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_19

Friction: Applying Resistance to Velocity and Spin

The pFriction tool applies resistance to the motion of a particle, slowing the particle's motion through a defined region. This tool produces two types of friction. One type reduces the velocity of any particle intersecting or crossing the defined region, and one reduces or dampens spin and rotation. Let's continue with the project that we started in Chapter 18 and add some more advanced particle physics tools as well as some image processing tools to create a fireworks VFX.

Use the **Add Tool ➤ Particles ➤ pFriction** context menu to add a friction algorithm to the particle system simulation's flow node editor, as is shown in red on the right in Figure 19-1. We will need to wire this in between the pSpawn2 explosion setting and the pDirectionalForce gravity setting, as air friction will affect particles more immediately than gravity will. A friction node is shown wired into place in Figure 19-2, circled in red.

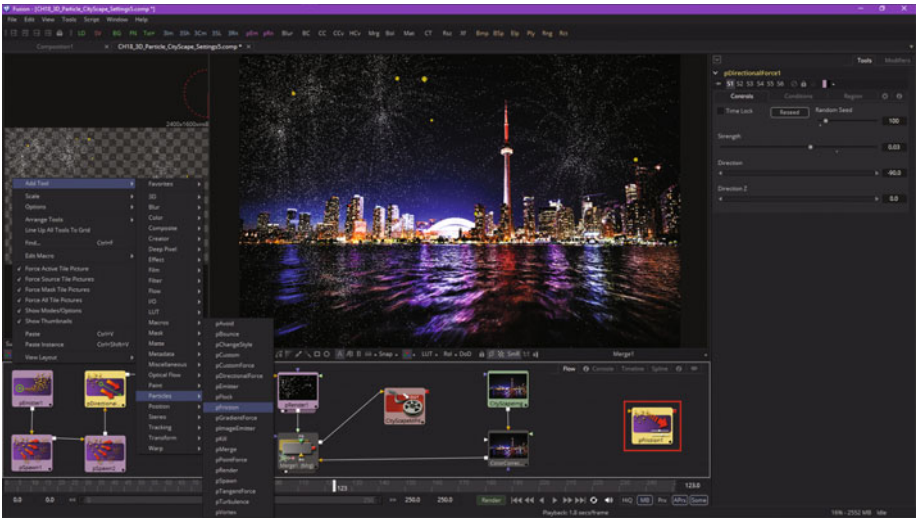

Figure 19-1. *Add a particle friction node to the particle systems project VFX pipeline*

Randomize the algorithm by selecting a **random seed** value, such as the **13** value I have used, as shown in Figure 19-2.

Figure 19-2. Conform the particle friction algorithm using particle controls and sets

The Random Seed slider and **Reseed** button are available whenever the tool relies on a random result. Two tools with the same random seed value may produce the same result, so do make sure to make all your seed values different, even if it is just by an increment of one, for each of your VFX project nodes. You could also click on the Reseed button to have Fusion 8 select the random seed value, or adjust this slider manually and select your own random seeds.

The **Velocity Friction** slider value sets a friction force that is applied linearly to your particle's linear velocity. The larger this value is, the greater the friction is that will slow down the particle. A **Spin Friction** slider value represents your friction force applied to a particle's rotation, or spin. The larger the value, the greater the friction, thus slowing down the rotation of the particle. This can be seen on particles such as lines or stars, for instance. I set Velocity Friction to **0.1**, or 10 percent, and Spin Friction to **0.05**, or 5 percent, for the initial starting values, which are guesses as to the visual effect I am trying to achieve.

Select **Affect Only Set 2** in the **Sets** tab so that the air friction only applies to the fireworks explosion. I also changed the particle style type to **Line**, as shown in Figure 19-3, and I tweaked the settings to increase **Size** and **Size Velocity** settings to **0.5**. I also increased **Size Variance** and **Size Z Scale** to 20 percent (**0.2**).

Figure 19-3. Change pSpawn2 particle type to Line for more realistic fireworks effect

Now that the explosion effect looks more like a firework, we can add some more refinement using the turbulence algorithm.

Turbulence: Adding Chaos to the Particle System

The Fusion **pTurbulence** tool algorithm creates frequency-based chaos for the position of each particle in the particle system. This causes particle motion to become chaotic (unpredictable) and random, much like the motion of water in a raging river versus water in a still lake. The controls for this tool affect the strength and density of the turbulence along each of the x-, y-, and z-axes. Use an **Add Tool ➤ Particles ➤ pTurbulence** context-menu sequence, as shown in Figure 19-4, and add the turbulence algorithm node to your flow node editor.

Figure 19-4. Add a particle turbulence node to your particle project

As you will see in Figure 19-5, I used a random seed of **29** and set the x-, y-, and z-dimension turbulence **Strength** factors to 5 percent, or 0.05. Having Strength be specifiable on a per-axis basis allows many more cool visual effects to be created. We will be using this for air turbulence, which exists on all three axes.

Figure 19-5. Set up the turbulence algorithm using particle controls and sets

The Strength setting tells the turbulence algorithm what percentage of chaotic motion to introduce in a particle system. There is also a **Strength Over Life** spline graph, which should be used to control the amount of turbulence applied to a particle according to its age. For example, a fireworks particle should originally have very little turbulence applied at the start of its life, and as it ages, the turbulence should increase as the particles die out.

I used the default **Density** setting of 10. You should use this control to adjust the turbulence density. High values will produce finer variation in the turbulence produced. Also notice that I have selected both **Set 1** and **Set 2** in the Sets tab, with the **"Affect Specified Sets"** option selected in the drop-down menu.

The next upgrade I'm going to make to the pSpawn explode algorithm is to use a **Line** particle style type in order to better simulate a firework effect, and then add a color gradient to give us the hot white flash, and then a yellow through orange to red and finally dark purple (dying) spark color.

After setting the Style drop-down to **Line**, as shown at the top right in Figure 19-6, I increased the strength of the settings so that **Size** was **1.0**, **Size Variance** was a third (0.33), **Size to Velocity** was **75 percent** (0.75), and **Size Z Scale** was **0.25**.

Figure 19-6. Refine Line particle type settings and add a particle life color gradient

This makes an explode effect look like a firework blast, and now all we have to do is to add colors and then refine the effect using other Fusion 8 tools to increase the VFX project's realism, which is always the desired end result in VFX pipeline design work.

Open the **Color** section of the **Style** tab and set a yellow color, if you want a firework to be all one color. To get more realism, use the **Color over Life** (Gradient) controls. This will allow you to set a white flash and then have a spark cool from yellow to orange to red to purple, as is shown in Figure 19-6.

To set the gradient color "stops," you click inside the gradient bar widget where you want the color (stop) to change, then use the **Color** dialog, shown on the left side of Figure 19-6, to set the color stop value.

Next, let's add some **directional motion** to your firework blasts, just as you see in real-world fireworks displays, where some of the fireworks that explode seem to be going toward one direction rather than exploding evenly. This is due to upper atmosphere wind currents.

Vortex: Adding Radial Motion to a Particle System

The Fusion **pVortex** tool will apply a rotational force to each particle, causing them to be drawn toward the source of the vortex, much like you see in a whirlpool, or with water going down a drain (unless you're on the Equator, in which case there is no spin from the Coriolis effect). We can use this to make some firework explosions more even, and others more directional, while also showing you another impressive particle physics algorithm that is available in Fusion.

Use the **Add Tool ➤ Particles ➤ pVortex** context menu, and add a pVortex node tile to the project, as seen in Figure 19-7.

Figure 19-7. Add a pVortex node tile to your particle physics simulation node processing pipeline

In addition to the common particle controls, the pVortex tool also has controls used to set up the vortex physics effect within the particle system's VFX simulation.

The **Strength** slider will set the strength for the vortex force that should be applied to each particle. The **Power** slider will set the degree to which vortex force strength decreases as it passes through space. In the **Translation** (particle movement) section, the **X**, **Y**, and **Z Offset** sliders set the amount by which the vortex algorithm will offset each vortex-affected particle.

The **Size** slider can be used to set a size for the vortex force by using a round vortex widget, which can be seen in Figure 19-8. **Angle X** and **Y** sliders set the amount of rotational force that will be applied by the vortex algorithm along the x- and y-axis. I used a **Random Seed** of **37** and a **Strength** of 25 percent, or **0.25**.

Figure 19-8. *Set up and configure your particle vortex node's initial settings*

I placed this vortex algorithm processing after friction and turbulence, and before the gravity node tool, as you can see in red in Figure 19-8. I set **Size** to cover **90 percent** (**0.9**) of the screen, as shown using the green and red widget. You can drag the vortex to position using the center red portion of the widget, which will set your Translate section's X, Y, and Z Offset values.

I used the **Sets** tab to select **Set 2** (firework explosion) to be affected by the vortex algorithm by using the **Affect Selected Sets** drop-down menu setting, as seen in red on the bottom right in Figure 19-8.

I was not getting enough directionality from my explosions inside of the vortex widget, so I increased **Angle X** to **20.0** and **Angle Y** to **30.0**. This is seen in red in Figure 19-9, and as you can see in the pRender preview and in the merge view, the particles inside the vortex are directional, and the ones outside it are not. This gives a higher degree of wind current simulation realism to the VFX fireworks, and now we're ready to add more nodes.

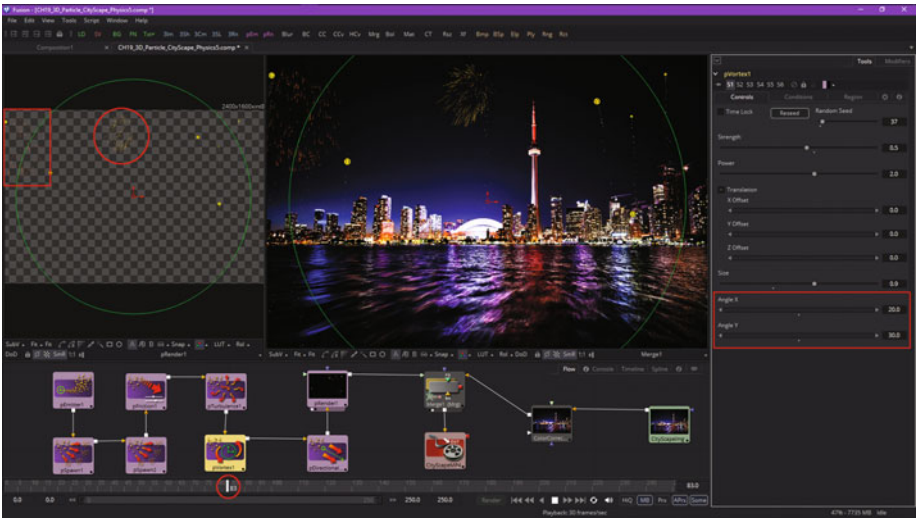

Figure 19-9. *Adjust and refine the final particle vortex node tile settings*

As you can see, what can be done using particle systems, along with physics, in Fusion 8 is nothing short of amazing. But, why stop there? Let's combine this power with Fusion's **modifier algorithms** and image processing algorithms, along with precise masking capabilities, and take this to another level of realism entirely! After all, that is the charter of the VFX artisan—to provide their clientele with something they didn't have to film, that appears to their viewers as if it were indeed filmed!

Shake: Using Modifiers with Particle Systems

In this chapter, I also wanted to make certain that you are using the other tools that Fusion offers to enhance your particle system simulations. The modifiers are especially useful for modifying the particle effects parameters algorithmically, and digital image processing tools help **post-process** the particle system downstream from a pRender tool (tile). The **Shake** and **Perturb** modifiers are especially useful for varying data values, and you can use these to make these fireworks vary in color, just like they do in real life!

A Shake modifier can be used for random position, speed, or color values (or other algorithm variables, if needed). This will allow you to create animated, randomized, numeric inputs.

A resulting shake can be entirely chaotic, or the motion may be smoothed or dampened, to create a gentler, more “organic” look and feel.

To add a Shake modifier to the pSpawn2 velocity, to vary the size of the explosions, select **pSpawn2**, as shown as Step 1 in Figure 19-10, and right-click on the **Velocity** slider, seen as Step 2. Select **Modify With ➤ Shake** from the context menu that appears. To set your Shake modifier parameters go into the **Modifiers** tab, which as you know you can find at the top of the control panel on the far right. I added this to Figure 19-10 as Step 3, and as you can see I chose a different Random Seed, and set a **Smoothness** of 2.0. To get a more random effect I unlocked X and Y, and I set a **0.125 Minimum X** and **0.111 Minimum Y**, and a **Maximum X** of **0.333** and **Maximum Y** of **0.256**. Firework blasts will vary in size, as different velocities cause different blast diameters!

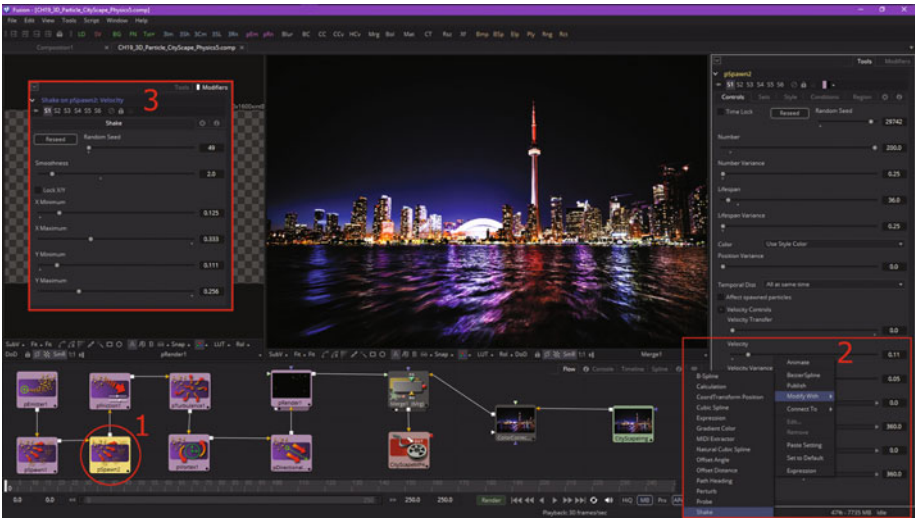

Figure 19-10. Add the Shake modifier to the pSpawn2 velocity data value parameter

Next, let’s go back and refine some of these settings to fine-tune your fireworks effect, such as shortening the life of the particle trails so they don’t fall back down after a blast.

Refining a Particle System: Debugging Algorithms

Since I have been working on the firework blast algorithm for a while, I decided to go back and finish the particle spark trail going up to the blast, or **pSpawn1**. I increased the number of particles to **9** to get more sparks on the way up and then saw a fallback effect that was not realistic, so I shortened the life, using a lifespan of **40**. This uncovered a bug in my visual node programming structures for this particle system, as is shown in Figure 19-11. So now I have to find out why this is happening!

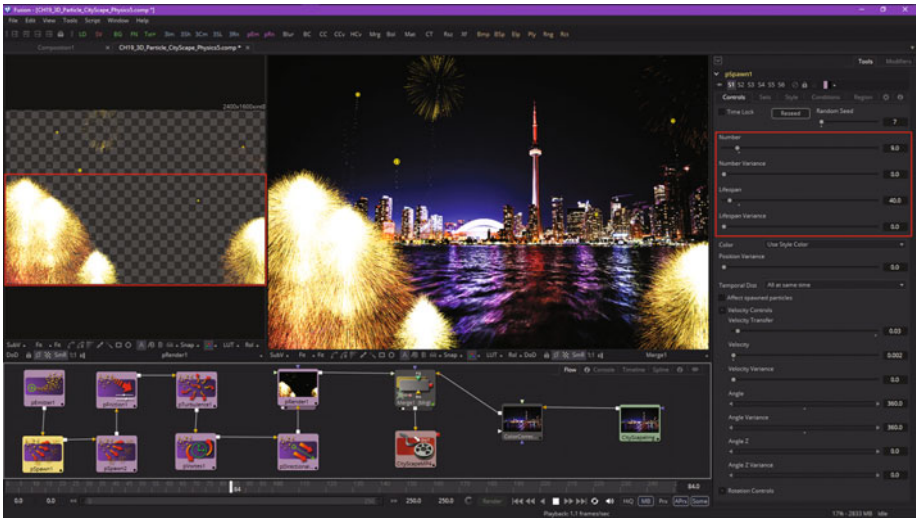

***Figure 19-11.** Lowering the pSpawn1 lifespan revealed an incredibly beautiful particle simulation settings bug*

My debugging process involved looking through all of the nodes to see what the settings were doing. I discovered that in the **pSpawn1** node, which created the spark trails that show these fireworks being shot into the sky, I had **Set 1** configured both as the set for pSpawn1 to use to label itself, and as the conditional set for it to affect, causing the potential for an "infinite loop" of sorts, relative to this algorithm. The obvious solution, since we have already used **Set 2** for the blast, is to use **Set 3** for the trail, and use **Set 1** for the emitter (the firework mortar), as is seen circled in red on the left side of Figure 19-12. Now, pSpawn1 particles are labeled Set 3, but are referencing Set 1, so they're no longer referencing themselves!

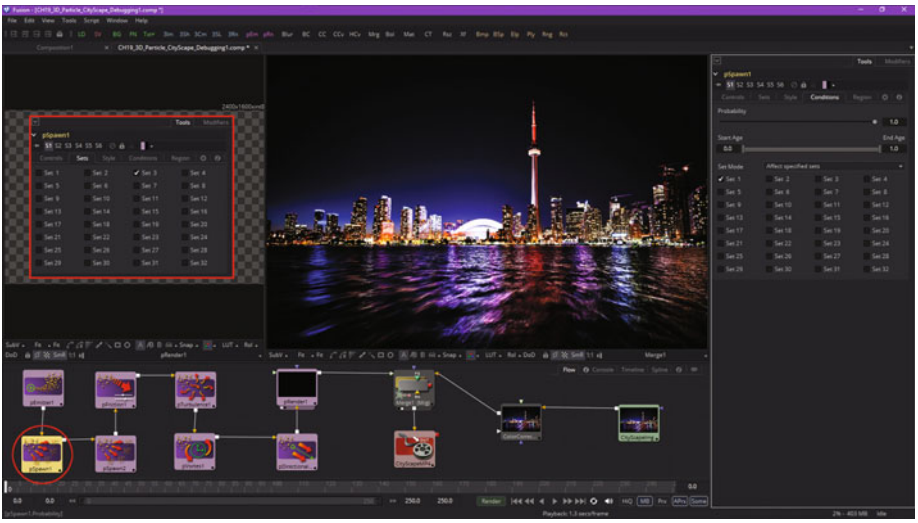

Figure 19-12. Eliminate the circuitous set reference in pSpawn1 to eliminate the problem

Sometimes I wonder how many of the super-cool VFX that I see are discovered by mistake during parameter refinement! Next, let's use the Perturb modifier to make the color for your firework blasts different for each blast, just like they are in real life! I am trying to cover as many different Fusion 8 modifiers and particle tools as I can, so we can cover at least all of the core ones you'll be using in the future.

Perturb: Use Modifiers with Particle Color Gradient

Since we're using Shake for the particle blast-size variations, let's use **Perturb** for the particle blast-color variations, so we can learn more about that modifier algorithm and its parameters. You should add this to your **particle gradient**, by right-clicking on the control, and selecting **Perturb**, as seen in Figure 19-13. Let's first learn about the Perturb algorithm, and then we will set-up the Perturb modifier control panel with your desired parameters.

Figure 19-13. Right-click the pSpawn2 gradient color control and select the Perturb modifier

The Perturb modifier is used to create "jitter." This is smoothly-varying randomized values across multiple input types, such as the color stops and color channels which are used in a color gradient.

The Perturb algorithm is based on a **Perlin noise function**. Perlin noise will be used to add jitter, shake, or wobble to any animatable control. You can Perturb a control even if a control is already animating! The result of a Perturb modifier is often similar to the result for a Shake modifier. Perturb has different control parameters, which might be more suitable for your needs.

Unlike some random modifiers, the Perturb modifier can be applied to a **polyline**, a **shape**, a **grid mesh**, or a **color gradient**, which is why it is on the right-click menu, and why we're using it here for our fireworks blast random color variation purposes.

I show the **Modifiers** tab circled in red in Figure 19-14, where I expanded the **Perturb1** section under the Shake1 section. I set a **Random Seed** of **51** and a **Strength** value of **2.0**, with the **Wobble** variable set to **1.5** and the **Speed** variable set to **1.75**.

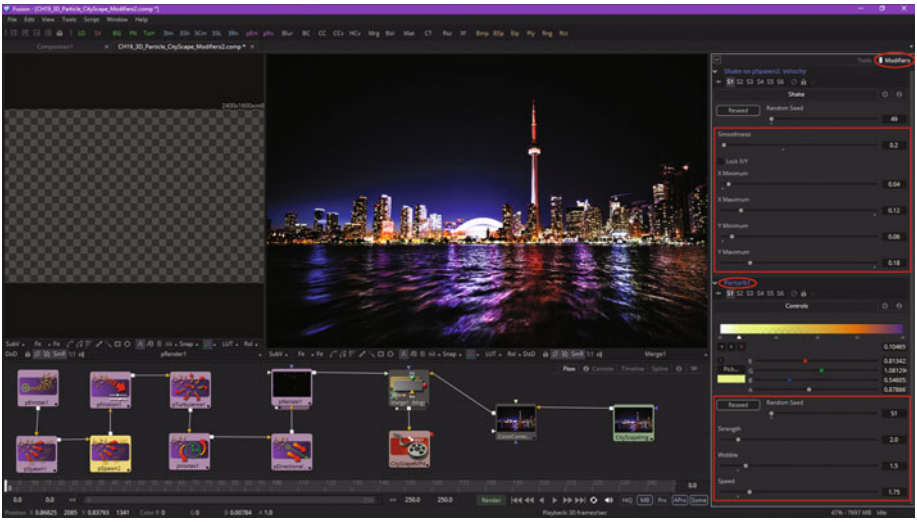

Figure 19-14. Configure the Perturb modifier's setting values to randomly change fireworks color

Finally, I updated the **Shake** modifier parameters as well to define a larger range of blast sizes for the firework blast. This is shown in a red square at the top right of Figure 19-14.

I used a **Minimum X** setting of 4 percent (**0.04**) and a **Maximum X** setting of 12 percent (**0.12**) since I unchecked the **"Lock X/Y"** checkbox. This gave me the ability to add even more variety to the chaos for the blast velocity, which would translate to different firework blast radius results for each successive firework launch, just like you would experience in real life.

I used a **Minimum Y** setting of 6 percent (**0.06**) and a **Maximum Y** setting of 18 percent (**0.18**), with a Random Seed value of **49**. I used a **Smoothness** setting of **0.2** or 20% to get drastic variances in fireworks blast size!

With eight fully-loaded particle system node tiles on the left side of the flow node editor, as you can see in Figure 19-14, I wanted to make sure that there were at least eight image processing node tiles on the other side of the flow node processing pipeline! So, let's add some post-processing to the pRender node, using the digital image processing techniques you learned about in the first half of this book, which will enhance your visual effects project even more, when used with particle system nodes. I wanted to make sure to have a professional VFX processing pipeline with some complexity in place before this book ended!

Imaging Effects: Post-Processing Particle Systems

Let's fatten up the firework explosion line particles using a Glow node and then use a Brightness-Contrast node to increase the color saturation (or to decrease it, to increase realism). Use the **Add Tool ➤ Blur ➤ Glow** context menu to add a glow node, and then wire it between your pRender node and Merge1 node, as shown in Figure 19-15 circled in red. Set the **Filter** option button to **Gaussian**, and deselect your **Alpha** channel processing option, as shown in red in Figure 19-15. I set a **Glow Size** of **1.1** and **Glow** to **0.9**. I set **Blend** to two-thirds (0.67) and used the **Threshold Apply Mode** with a **0 Low** setting and **75** percent (**0.75**) **High** setting. As you can see in Merge1 View2, the firework blasts are fattened up beautifully by using this image processing step, and are more realistic than ever. Next, we'll add a **brightness-contrast** node.

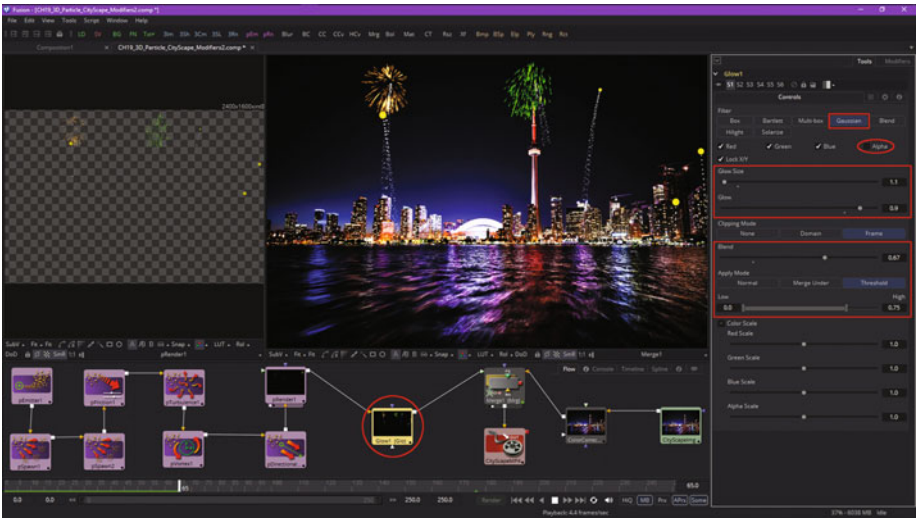

Figure 19-15. Add a Glow node, and configure it as a Gaussian glow effect

Next, use the **Add Tool ➤ Color ➤ Brightness / Contrast** context menu, and add a **BrightnessContrast1** node, then wire it up between the Glow1 node and Merge1 node, as seen in Figure 19-16. Set Saturation to **0.5** to get a reduction of 50% percent in color saturation (which increases the realism), or you can also use a value greater than 1.0, say **1.333**, to increase your color saturation. Add a Transform node tile to your pipeline and pull a second data pipe out of the Brightness-Contrast node tile and wire it into the Transform node input, as shown highlighted in red in Figure 19-16.

Figure 19-16. Add a Transform node, and add a second data pipe into it from the Brightness-Contrast node

Next, let's take a look at using the Transform node tool in Fusion, and use it to create reflections on the water for the fireworks.

Reflections: Transforming Your Particle System

Since you can pull more than one wire (pipe or data path) out of a node output, we have created a transform node by using the **Add Tool ➤ Transform ➤ Transform** context-menu sequence, and then pulled a second wire (data pipe) out of the BrightnessContrast1 node, and connected it to the Transform1 node, as is shown in Figure 19-16.

Since we want it to be a reflection, and therefore upside down, you want to select the **"Flip Vertically"** checkbox, as shown circled in red in Figure 19-16. Leave the other options set at their defaults; you will tweak these later once the primary reflection visual effects are operational. This could take some more node wiring, which we are going to do next using a second Merge node, and some other image processing in order for the reflection itself to give it more of a realistic (blurred and distorted) result. We'll wire up the remainder of the digital image processing node pipeline next.

Add a **soft glow** node after your Transform node, and then add a **Merge2** node and merge your Merge1 node and the SoftGlow1 node, piping the result into your **Image Saver** node, as shown in Figure 19-17. Set the Merge2 **View2** and preview the reflection effect using the **Play** button in the Fusion 8 time ruler.

Figure 19-17. Add the SoftGlow1 and Merge2 nodes, and wire them together using data pipes

This **SoftGlow1** algorithm is adding too much light (glow) to your reflection, so let's learn how to change a node tile's algorithm using the **Replace** function in Fusion. Right-click on your SoftGlow1 tile, as is seen in Figure 19-18, and select the **Replace Tool ➤ Blur ➤ Blur** context-menu sequence. Note that all similar settings should be transferred (set automatically) from the previous tool, as was the **0.5 Blend** setting that I had set.

Figure 19-18. Replace the Soft Glow node tool with the Blur node tool

Next, let's use what we learned about mattes and masking to make sure that the particle system's reflection only happens inside of the water portion of your source image.

Particle Mattes: Masking Your Particle System

Select your **Merge2** node and set the **center** to **1200,560** so that the reflections emanate from the same location as the Line type (emitter) does. I have circled the Merge2's centering widget in Figure 19-19 so you can see it is tangent to the Line emitter. Click the **Add a Rectangle Mask** icon in the view for Merge2 and add a **Rectangle1** matte, which will automatically appear in the flow node editor and be wired into the top of Merge2. You can see this in Figure 19-20 if you wish to look ahead. Now all you have to do is set the matte size and center location using the rectangle mask widget's centering (drag to position) widget and size (highlight side and drag to to size) features. Make sure the mask widget covers the surface of the water perfectly, such that your reflections will only appear in the water.

Figure 19-19. Set the Merge2's center to 1200,560 and add a Rectangle mask node tile

Figure 19-20 shows the positioned rectangle mask and the slider for setting the level of alpha channel blending for this rectangle. You can also set the **center**, **width**, and **height** using numeric input widgets, seen circled in red on your right-hand side. The **Rectangle1** tile shows an alpha channel mask representation.

Figure 19-20. *Set the Rectangle1 mask widget so that it will cover the water areas*

If you slide the **Levels** slider, you'll see the node tile mask representation dim from white (fully transparent) to gray (dimmed reflections). This is one of the controls you will want to tweak when your entire pipeline is set-up, and you are in the settings-refinement stage, which will be reviewed during a final section in the chapter on the "tweaking" process.

The Rectangle mask will constrain the mirror reflection, which looks wrong, that you saw in Figure 19-17. As you'll see in Figure 19-21, the reflection should now work correctly. Let's also mask the sky portion of the image using the ChromaKey node tool, so that the fireworks appear behind the office buildings, instead of in front of them.

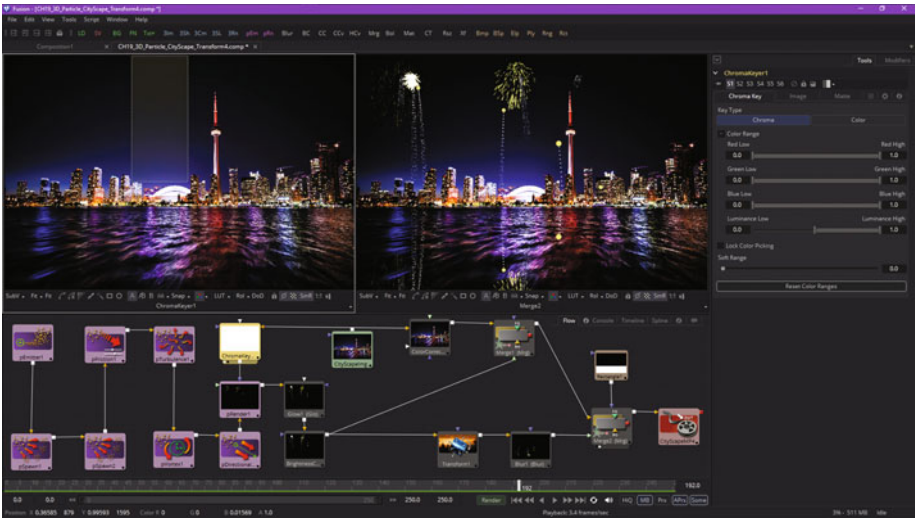

Figure 19-21. Add a ChromaKey matte node and select a range of blue values

Now, let's create a more advanced matte so that the flare and blast effects look like they are coming from behind Toronto buildings rather than from in front of them. We could use the **ChromaKey** tool we learned about during Chapter 12 to remove a dark blue sky and light blue horizon, and then flip the mask, so that the fireworks only appear within the area of the night sky, and nowhere else.

To do this, we'll have to rearrange our flow node wiring, so that the background image loader can be accessed by the Merge2 node, as well as by a ChromaKey node, as is shown in Figure 19-21.

Select the **Chroma** button in the **ChromaKey1** control panel, and drag out a color-selection marquee in the center of the image, so that you get the widest range of blue color sampling. If any shades of blue sky remain, subsequently drag a marquee around them as well.

The resulting mask, which used a small amount of mask creation effort, is an impressive demonstration of Fusion 8's power, as can be seen in Figure 19-22 in View1, which I hae set as the ChromaKey1 node's view dot setting. The next step is to go into the **Matte** tab so that I can invert the matte.

Figure 19-22. Select any and all blue horizon values to build the night sky matte (mask)

Once you invert your matte, you will be left with some of the water that was the same blue color reflected from the sky, so you'll need to add a **rectangular garbage matte** to finish the job. Click the **Add a Rectangle Mask** button to add a **Rectangle2** node, as seen in Figure 19-23, and then position it over the water.

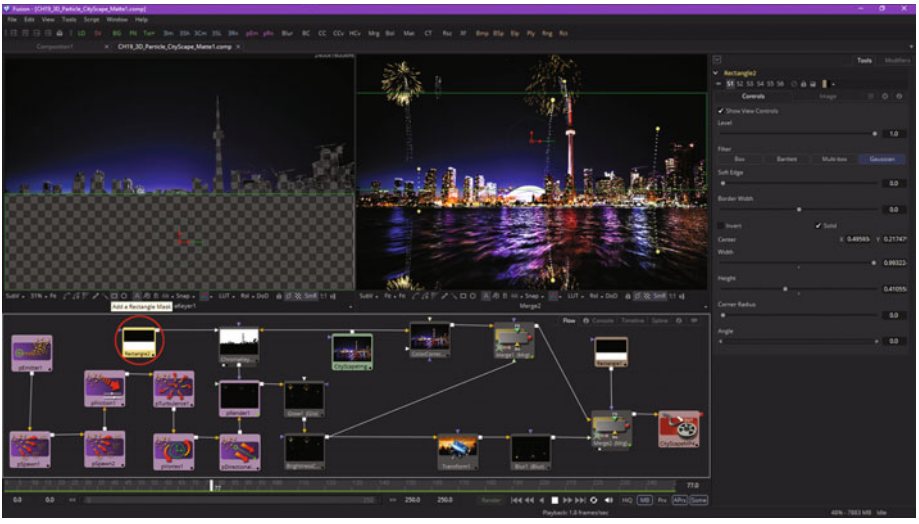

Figure 19-23. Add a rectangle garbage matte, then configure it

The result is a quickly created professional matte as is seen in Figure 19-24, and you are ready to refine parameters in your Flow Node VFX pipeline, to start enhancing the realism for your VFX project. Most industry VFX artists call the refinement process “tweaking”

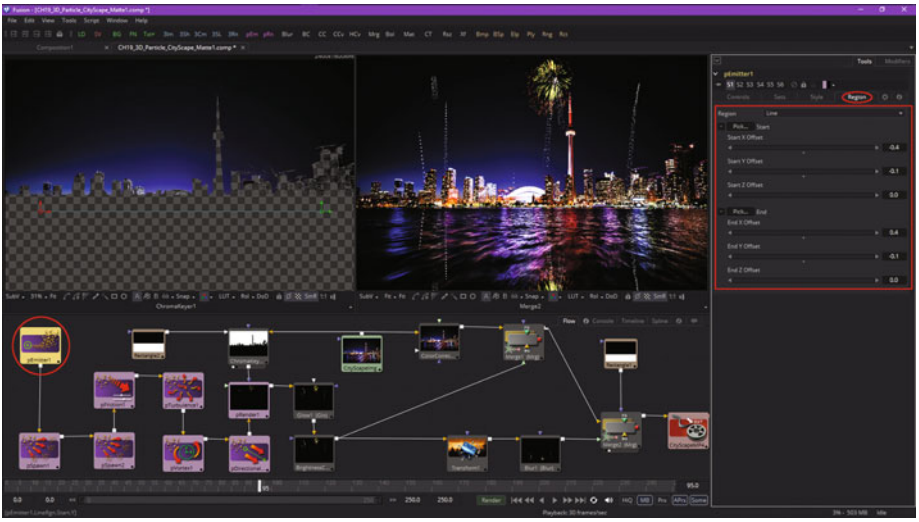

Figure 19-24. Adjust a Line emitter to keep fireworks on screen

Tweaking the Particle System: Reality Refinement

There are a few things that we need to do to refine and finish off our particle system fireworks simulation, such as removing the happy face Brush emitter and turning that back into a **Point** emitter, as well as shortening the **Line** particle style type so that the array of fireworks launches more toward the middle of the image, so that blasts remain fully visible well within the frame of the resulting video.

You can see this Line emitter, and its new length, in your ChromaKey View1 pane on the left in Figure 19-24, as well as the new settings for a uniform 10 percent below center (**-0.1**) **Y-height offset**, and 40 percent from center, or 10 percent in from the sides, **X-width offset**.

The VFX project now has **20** powerful node tiles accessing dozens of powerful keying, masking, imaging, particle, physics, pixel-transformation, rendering, compression (Quicktime codec), and animation algorithms, and at this point you have hundreds of parameters you can tweak or adjust to refine the VFX project pipeline and control this image, which you have now turned into a celebration that never really took place.

Next, let's get rid of those happy face Brush-style type emitters that show us where the emitter is and instead use just a single Point-style type so all you see is a trail of sparks, which is more akin to what you would see in real-life.

I also need to adjust the color of these sparks, which I decided to make yellow-orange for the **Emitter Point** type, using the **Style** tab **Color Controls ➤ Color** setting of Red 100%, Green 80%, and Blue 65%. As you can see in Figure 19-25, I've then set the pSpawn1 algorithm to use this color value for all particles in the firework launching trail by selecting the **"Use Color from Parent Particle Color"** option from the drop-down.

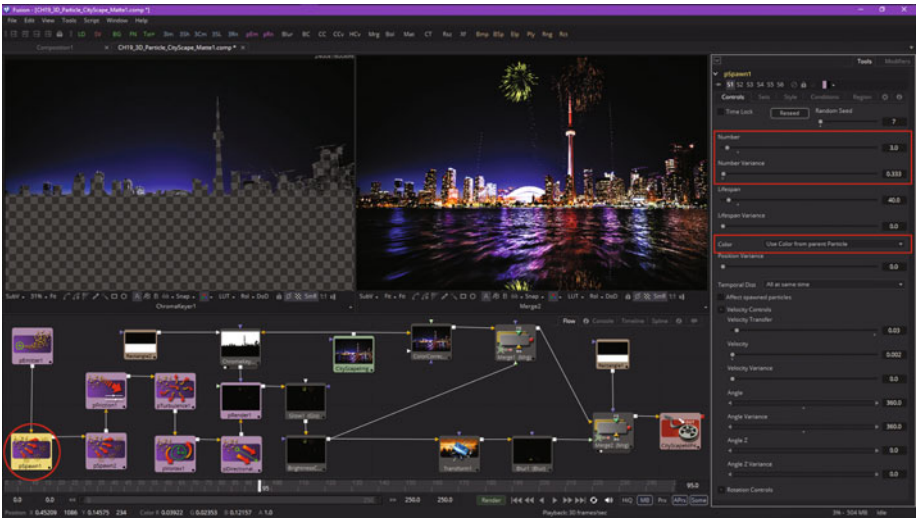

Figure 19-25. *Reduce your spark trail size, and set Use Color from Parent Particle Color*

As you can see, I have also reduced the number of sparks visible significantly to only **3**, to get a more realistic effect, as spark trails are usually minimal. I set a **Number Variance** of one-third, or **0.333** (33.3 percent) so that the **Number** field fluxuates between 2 and 4. One-third of three is one, so the variance may therefore vary from one less (two) to one more (four) than your Number setting of 3. These tweaks can be seen in red on the right-hand side of Figure 19-26.

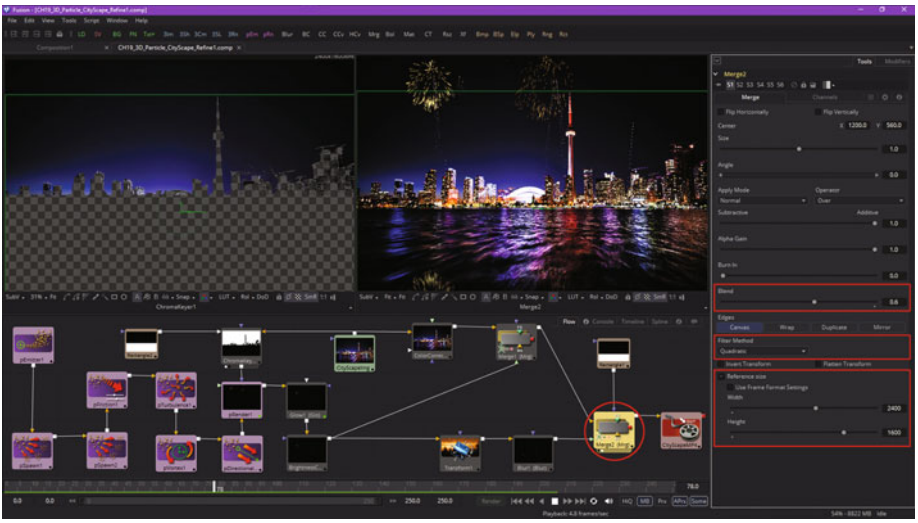

Figure 19-26. Adjust your spark trail values, blending modes and filter algorithms

Once you finish tweaking the particle system node tiles on the left side of the flow node editor (in purple), you will start to refine the image compositing effects and blending mode settings on the right side of your flow node editor (in gray).

You have seen how I set up the flow node editor tiles so that it's easy to visualize an algorithmic flow through the VFX processing pipeline. This is important to streamline your settings refinement process once you get to that stage of your VFX pipeline creation work process.

The Merge2 control panel has an **Apply Mode** area, where 2D image compositing blending modes and related settings are found for mixing the firework reflections, inverted using a transform node, into the original image background used in a Merge1 node. The output of the Merge1 node combines a particle system render result with the image loader source image and is piped into the Merge2 background input. The inverted and blurred particles are the foreground, and are composited together using a blend mode; in this case, a **100** percent **Additive** with **60** percent **Blend**, as shown in Figure 19-26. I used the quadratic filter method (algorithm) used in higher quality compositing and scaling operations. Continue the tweaking process in this VFX pipeline until the VFX results are perfect! Make sure to save your .comp file every so often, so that you do not lose any of your settings tweaks as well.

Summary

In this nineteenth chapter, we took a look at particle physics and the primary physics algorithms provided in Fusion. We also looked at how modifiers, mattes, and imaging algorithms can be used with particle systems to achieve a more realistic visual effect. I wanted to end the Fusion feature chapters with a pro-level VFX project that showed how even complex projects can be created in Fusion, along with the type of continual refinement that they require to achieve an acceptable level of realism.

In Chapter 20, you will learn about VFX content **publishing platforms** and consumer electronics hardware devices.

Chapter 20

Advanced Interactive VFX: i3D Content Publishing

Now that you have a good measure of exposure to fundamental visual effects concepts, techniques, and principles, it is time to take a look at how to publish visual effects content. We will see how to do this via popular open source content distribution platforms such as HTML5, Java and JavaFX, Kindle, Lumberyard, and Android. I am going to organize this chapter based upon consumer electronics hardware device genres.

Consumer electronics devices can support robust games or VFX applications. For instance, **e-Book e-Readers** such as Amazon's Kindle Fire HD use Kindle KF8 format; **smartwatches** use Android Wear SDK under the Android Studio 2.2, using the Android OS 7.0 API; **iTV sets** use the **Android TV** SDK in Android Studio 2.2, and use the Android 7.0 API; **automobile dashboards** use **Android Auto** SDK with Android Studio IDE, with the Android 7.0 API; **tablets** and **smartphones** use Android SDK in Android Studio 2.2 IDE with the Android 7.0 API; **laptops** and **netbooks** use Java with JavaFX; and each of these hardware devices also supports all of the open industry publishing standards, such as PDF, HTML5, and EPUB3.

Even more important, all of these platforms and devices support the same OpenGL 3 i3D (interactive 3D) platform that Fusion 8 supports, and some of them support OpenGL 4 and have GPU hardware as well as HD, and in many cases UHD (4K), displays. This makes them the perfect candidates for delivering your VFX content projects.

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0_20

We will look at how to publish using electronic hardware device types, with the software development platforms these devices support, such as Amazon Kindle, Fire OS, Lumberyard, Google Android Studio 2.2 (Android 7), Oracle's Java 7, Java 8, Java 9, and JavaFX, Adobe's PDF, and the open source WebGL2, SVG, XML, HTML5, EPUB3, CSS3, WebGL, JSON, and JavaScript. Virtually all of these support advanced MPEG-4 H.264 AVC or WebM VP8 and soon will support MPEG-H H.265 HEVC or WebM VP9 and OpenGL real-time rendering in conjunction with 24-bit HD audio. VFX has arrived!

Open Source Formats: PDF, HTML5, EPUB

Let's start with **open content publishing formats** that have been defined by industry groups that support digital video, such as **EPUB3** or **HTML5**, or that have been **open sourced**, such as Adobe **Portable Document Format** (**PDF**) and Amazon's **Lumberyard** AAA game engine. I'm starting with these open formats because they'll be usable across every type of device, including Android OS and iOS, and in some cases on new HTML5 OSes and in HTML5 browsers.

Portable Document Format: Visual Effects in a PDF

The Adobe PDF (Portable Document Format) is utilized by Adobe **Acrobat Reader** and is used around the world for publishing rich media documents, which can include **interactive 3D**, **digital video** and **digital audio** content, as well as digital imagery, digital illustration, or digital painting. I have new media fundamentals titles on all of these areas at www.Apress.com. Acrobat Reader is free, and its **PDF** data format has been open sourced! The **Adobe Acrobat Professional** series of publishing tools are still paid software packages, and are well worth the money if you need to publish via this widely accepted, rich media document publishing format. This PDF format supports **many** digital video formats, including **MPEG-4 AVC H.264** and **Windows Media Video (.WMV)**, as well as Shockwave Flash (.SWF), Flash Video (.FLV), Microsoft AudioVideo Interleaved (.AVI), RealMedia (.RAM), 3GPP Movies (.3GP), Advanced Streaming Format (.ASF), and **QuickTime Movie** (.MOV), which we've used during this book to output VFX content.

Digital audio is also supported in PDF publishing and includes similar formats, such as **MPEG-4 Audio** (.M4A), **Windows Media Audio** (.WMA), Windows Wave (.WAV), MPEG-3 Audio (.MP3) or Audio Interchange File Format (.AIFF), which is used in Apple products.

It used to be the PDF format was only used for creating text-based business documents. However, it has now been adopted as an e-Book format, so it needs to support new media elements. As a matter of fact, you may be reading this book using this PDF file format right now. Other popular e-Book formats include EPUB (EPUB3) and Kindle 8 (MOBI), which we will be covering later on during this chapter after we cover the more widely used open standards of PDF and HTML5.

Another significant advantage of PDF document publishing is that it offers **Digital Rights Management (DRM)** support. This is why I covered this format first. DRM allows you to **copy protect** (or lock) your document if you want to sell it for money. It is important to note here that HTML5, which we're going to cover in the next section of this chapter, is going to feature a DRM model at some point in the future, since HTML5 is being widely utilized as an operating system and for HTML apps.

Adobe also has PDF server products that incorporate DRM features. These allow publishers to market their PDF content in a more secure fashion, allowing better monetization opportunities for digital video content projects and published IP assets.

Another thing to note about PDF, EPUB3, and HTML5 apps is that they are "encapsulated" and require **no server** to function. I call this a 100 percent "client-side" approach. This approach allows you to sell your digital products much like they were physical products. Purchase the product and it's yours, just like a real product, and you can give it away, if you wish, but cannot copy it due to more and more advanced DRM.

Let's look at HTML5 next.

HyperText Markup Language: HTML5 Digital Video

You've already taken a look at how to use MPEG-4 AVC H.264 and QuickTime MOV format during this book. HTML5 supports the MPEG-4 digital video data format as well as WebM. Using plugins, HTML5 could also support other digital video formats, like QuickTime Movie or Windows Media Video. HTML5 digital audio support is more extensive than its digital video data format support, and can also be extended (using plugins) to add additional digital audio data formats. If you're also interested in digital video, you can read *Digital Video Editing Fundamentals* (Apress, 2016).

It used to be that HTML5 was only used for creating website designs. Recently, web browser manufacturers decided to utilize their web browser software (code) to create **HTML5 OS** versions for use on consumer electronics devices. They did this due to the success of Google's Android, Samsung Bada, Apple iOS, and Research In Motion (RIM) Blackberry.

Putting this browser code, with **app launch icon** support, on top of the **Linux OS Kernel**, produced a **Chrome OS** (Motorola), **Firefox OS** (Panasonic iTV), and **Opera OS** (Sony Bravia iTV Sets).

There's also **Tizen OS** (Samsung), which is managed by **The Linux Foundation** for the creator of the Linux OS, Linus Torvald. Tizen also utilizes HTML5 code, as does the **Jolla Sailfish OS**.

HTML5 is simple to implement under any platform, thanks to the open source **WebKit API**, which is also a part of Android Studio 2.2 (currently at Android OS 7.0, and soon Android 8.0).

HTML5 application and website publishing is therefore an excellent way to deliver digital video content using all of the embedded mobile OS, desktop OS, and web browser platforms. With HTML5, you can even make your digital video assets interactive!

This is why DRM is in the future of HTML5. HTML5 will be freely available to you forever for implementing visual effects assets with MPEG-4, WebM, PNG, JPEG, GIF, SVG, HTML5, CSS3, XML, JSON, AJAX, AJAJ, XSLT, Java 9, JavaFX, JavaScript, and similar open source languages, data formats, and publishing technology.

Many of the other formats (or platforms) we are going to be covering during this chapter are also supported under HTML5, so things are not as "clear cut" as you might think regarding a "where and how do I publish" decision. For instance, the **Kindle** reader runs "underneath" HTML5 browsers or OSes, and the **JavaFX** game engine found in Java 7, Java 8, and Java 9 is also supported under HTML5 browsers, and soon will be supported in HTML5 OSes.

Going another direction, the **EPUB3** format supports HTML5 inside of its data format, so your HTML5 code could not only be parsed by HTML5 OSes and browsers but also by EPUB3! **Kindle** and **Android** also support "parsing," or rendering, your HTML5 content.

Next, let's take a closer look at the open source EPUB3 publishing standard used for e-Books—and soon, for much more.

Electronic Publishing: EPUB Digital Video Support

The **EPUB** specification is a distribution and interchange format standard for digital publications and documents. EPUB3, the third major release of the open EPUB standard, consists of four specifications, each defining an important component of an overall EPUB document. **EPUB Publications 3** defines publication-level semantics as well as conformance requirements for EPUB3 documents. **EPUB Content Documents 3** defines XHTML,

SVG, and CSS3 profiles for use in the context of your EPUB3 publications. **EPUB Open Container Format 3**, or OCF3, defines a file format as well as a processing model for encapsulating sets of related resource assets in one single ZIP file format (EPUB container). **EPUB Media Overlays 3** defines a format and a processing model for the data synchronization of text with digital media assets.

EPUB3 has been widely adopted as a format for digital books, also popularly known as e-Books. This version significantly increases the EPUB format's capabilities so that it can support a range of new media publication requirements. These include complex layout, new media and interactivity, and international typography (fonts) support.

The hope is that EPUB3 will be utilized for a broader range of content, including e-Books, magazines, and educational, professional, artistic, as well as scientific publications.

EPUB3 supports digital video data embedded in documents using **MPEG-4 AVC H.264** and **ON2 VP8 WebM**, which is now owned by Google and has been open sourced. It also supports your digital image and digital painting assets using **PNG**, **JPEG**, and **GIF**, and digital audio is supported using **MP3** and **MPEG-4 AAC** data format as well. Digital illustration is also supported using the open source **Scalable Vector Graphics**, or **SVG**, vector data format.

EPUB3 should inherit the same function and feature sets that these data formats can provide your digital video in HTML5 as well as in Kindle, Java, JavaFX, Android 6.0, or other WebKit-capable content publishing and distribution platforms.

Another impressive new media feature in EPUB3 is called **media overlay document**. The media overlay document provides you with the capability of synchronizing your digital VFX creations with vector elements inside of the publishing content document.

Amazon Lumberyard: Royalty-Free AAA 3D Engine

Amazon recently acquired the Crytek AAA game engine code and created a **Lumberyard** AAA game engine, then made it **free** for VFX developers—and **royalty free** as well. You do have to use **Amazon Web Services**, which are best in class, and which you would most likely be using anyway, so this is not a big issue. Figure 20-1 shows the incredible quality level an Amazon Lumberyard project offers. This matches up perfectly with Fusion's quality level, so the two products will work together perfectly in your 2D and 3D visual effects digital content production pipeline.

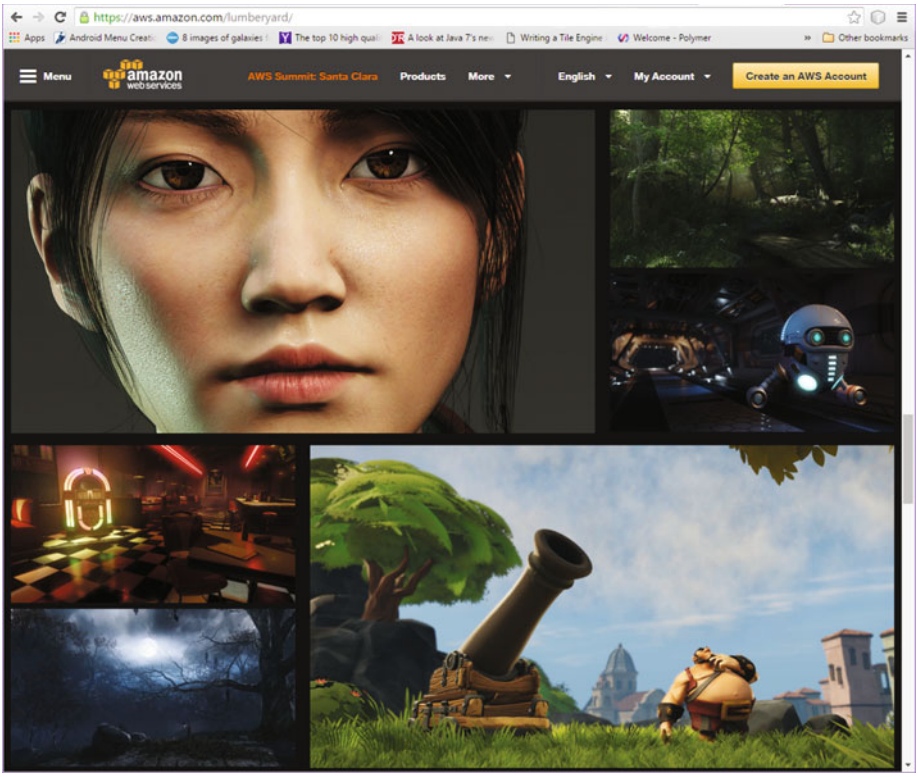

Figure 20-1. The Amazon Web Services Lumberyard web page

If you are interested in producing interactive 3D (i3D) content using Lumberyard and related 3D products, such as Daz Studio Pro and Allegorithmic Substance Painter.

Amazon Lumberyard supports the same .FBX i3D file format that we used in this book, and so using Fusion with these tools should be possible, if not probable, for i3D project pipelines and workflows if you are interested in this area of new media. You can find more information at `http://aws.amazon.com/Lumberyard`.

Open Device: iTV, e-Reader, Tablet, Phone

The next set of **devices** that I'm going to cover are open source and therefore are **free for commercial-use platforms**. Platforms are much more extensive than data file formats are because they execute, or "run," the data

file format on your device. Some of these, such as Android, don't run across every hardware device because they are competitors for other "closed" platforms, such as Apple iOS, RIM Blackberry, or Windows Mobile. These platforms are not industry specifications, like HTML5, but are instead owned by major industry hardware and software manufacturers.

For instance, Oracle owns Java 9 and JavaFX, Google owns Android, and Amazon owns Kindle, Fire OS, and Lumberyard. Let's cover these platforms based on the genres, or types, of consumer electronics devices on which these platforms run, starting with e-Book e-Readers, since the three formats we just covered are all widely used to deliver e-Books. You will see all of the various e-Book formats supported on the `http://www.Apress.com` website where you purchase your educational titles, some of which myself and my technical editor, Chad Darby, wrote, and which span topics from computer programming to new media content development and beyond. Visit this site soon and peruse all of these book titles; you might find a necessary title that up until now you had missed!

e-Book e-Readers: Kindle Fire, Android, Java, or PDF

The **e-Book e-Reader** hardware device is actually an Android tablet, which is why I added Android into the title for this section of the chapter. The world's most popular e-Book e-Reader, Kindle Fire, runs Android OS, as do the Sony e-Book e-Reader and the Barnes and Noble NOOK. Even the Apple iPad runs Kindle, EPUB3, and PDF e-Book titles, as do Blackberry tablets and Microsoft Surface tablets. The reason I added Java in the title for this section is that Kindle has Java capabilities for interactive e-Books, and Android uses Java as well. Since e-Book e-Readers will also read PDF files, I added PDF into this title as well.

Since most e-Book e-Readers are actually Android tablets or iPads, there are a plethora of platforms for delivering visual effects using MPEG-4 H.264 and OpenGL 3.

This means you could deliver visual effects content with Android applications, HTML5 applications, Java applications, HTML5 websites, Kindle Apps, NOOK Apps, or 3D new media PDF documents. This will give you lots of flexibility in publishing digital video content for e-Book e-Readers using the MPEG-4 AVC H.264 format or OpenGL ES 3.1. E-Readers are one of the best-selling new consumer electronics devices, especially during the holidays, along with the brand-new HD and UHD iTV set products.

iTV Sets: Android TV, Java, JavaScript, and HTML5

The **iTV set**, or interactive television set, is the most recent consumer electronics device to hit the marketplace, and iTV set devices are expected to explode in sales during 2017, 2018, and 2019. This is the reason Google has developed a specialized version of the Android SDK (Software Development Kit) for iTV sets called the **Android TV API** (Application Programming Interface).

There are **HTML5 OS iTV set** products as well from Samsung (Tizen OS), Panasonic (Firefox OS), and Sony (Opera OS), so the iTV set consumer electronic device is much like an e-Reader device in that it will allow you to create and deliver digital VFX content with **Java** or **JavaFX** (Android, iOS, HTML5, Windows), **HTML5** markup, **CSS,** or **JavaScript** (iOS, Android, HTML5, Windows).

It's also important to realize that with iTV set devices viewers are going to be paying closer attention to content streams, including visual effects, digital audio, interactive 3D, digital illustration, digital imagery, and digital paintings.

This level of close attention to your content is not always the case with other devices, such as smartphones, tablets, smartwatches, or automobile dashboards (at least, let's hope not).

If you wanted to deliver the digital video content asset across each of the iTV set device platforms, you should utilize HTML5. Android and iOS support HTML5, but HTML5 OS and websites do not currently support Android 5, 6, 7, or iOS applications.

The other side of that decision is that Apple and Google Play have more advanced app stores; therefore, if you're going to monetize your digital video and digital audio content, you should consider developing apps using Java (Android) or JavaFX (iOS) rather than using JavaScript or ECMAscript under HTML5 OSes or HTML5 browsers. HTML5 OSes do fully support MPEG-4 and WebM for digital video as well as for digital audio. HTML5 also supports OpenGL ES 3 via WebGL2, so it is catching up with Windows 10, Android 7, Linux, iOS, and all the popular game consoles, since every device these days has a GPU.

Smartwatches: Android WEAR 2, Java, and HTML5

The **smartwatch** is the next most recent consumer electronics device genre to hit the market. Smartwatch devices are also expected to explode in sales during 2017 and 2018, primarily because there are hundreds of manufacturers making them. This is because the densely populated watch industry is now moving to release smartwatch products so that they do not lose market share to consumer electronics manufacturers, such as

LGE, Sony, Motorola, and Samsung, who already have several smartwatch products each. One of the first custom Android APIs that Google ever developed was **Android WEAR** along with its **Watch Faces API**. Android 7 (Android Studio 2.2) will support the new **WEAR 2 API**.

Visual effects may be a popular feature for these smartwatch devices to support, because raster and vector formats can be optimized from a data-footprint standpoint, as I have often said in my writings.

Additionally, MPEG and OpenGL support is built into Android 7 Wear 2 hardware devices. What this means is that a smartwatch product is like a visual effects timepiece for the user's wrist and can provide professional digital video, 3D, or digital audio asset playback results with overlaid smartwatch functionality.

This is significant for visual effects content as well as digital audio content production, and for digital video application development professionals.

Another important feature of smartwatches is that you'll be able to combine visual effects assets with other interactive functional attributes, such as time, date, weather, fashion, and health features popular with smartwatches, like fitness and physical health monitoring (heart, pulse, etc.).

Once smartwatch screen resolutions go up from 480 pixels to 640, 800, or 960 pixels, even more functionality will become available to developers. The Huawei smartwatch already features a 400 x 400 pixel screen. LG's Urbane 2 is going to leapfrog this with a 480 pixel screen! High-resolution smartwatches should be appearing during 2017 (or 2018), given smartphones already have 4K-pixel screens that only span five inches; an 800-pixel smartwatch screen is certainly possible, as the technology exists already.

VFX software applications such as Fusion 8 are perfectly suited to producing smartwatch faces and applications, as they have the advanced feature set needed to produce the radial, or round, artwork assets required to fit the largely round OLED displays used in some of the most popular smartwatches from Motorola and LGE. Pixel density for these screens is 350 dots per inch, which is a higher level of quality than most laser printers, so your VFX artwork should appear razor sharp, and would be photorealistic as well.

Smartphone and Tablet: Android, Java, and HTML5

Smartphones and **tablets** have been around the longest, as has the hybrid between the two, commonly referred to as a **phablet**. The Android OS is used for all of these device types as well as personal computers that run the Android OS. There are currently billions of smartphones, as well as billions of tablets, and almost one hundred major consumer electronics manufacturers have made products for the open source Android operating system platform.

For this reason, these devices represent a significant opportunity for a visual effects artisan to create digital video and i3D content and applications based upon both of these i3D and VFX genres. This is because there are not as many digital video VFX applications as there are utilities, gaming, digital audio, and digital imaging or visual social media applications.

All popular smartphones and tablets include support for the MPEG-4 and WebM video formats, SVG command data, and SVG data file formats. Modern-day devices can render vector data in 2D or 3D into raster image data, which fits user device screens with pixel-for-pixel precision and can be overlaid on videos using algorithms similar to those utilized in Fusion.

Present-day i3D content publishing platform capabilities will therefore encourage very high-quality visual effects using HD digital audio, thus increasing i3D content consumption. This will allow digital videos, digital images, and digital illustrations to be combined, not only together but also with other new media genres like digital painting, digital audio, speech synthesis, and i3D. This means that you can create a "never before experienced" UI, or user interface, design and amazing user experiences with your digital video editing projects and special effects pipelines.

One indicator that consumer electronics devices have the GPU, CPU cores, and system memory to process the type of VFX we've been learning about in this book is that the Amazon Lumberyard AAA game engine actually delivers i3D applications for both iOS and Android mobile (phones, e-Readers, and tablets) platforms.

Game Console: Lumberyard, Android, Java, HTML

Since Android, Java, JavaFX, and HTML5 now support **OpenGL ES 3.2**, a plethora of advanced game-console products have appeared that are affordably priced between $60 and $180. This is yet another opportunity waiting to happen for the digital visual effects artisan, which you will be once you practice what you have learned in this book. The game consoles run Windows, Linux, or Android, and therefore support Lumberyard, Java, JavaFX, and HTML5 applications—and even e-Books, for that matter. There are over a dozen affordable game consoles currently on the market, and the type of VFX content created in Fusion is a perfect fit!

Some major industry brands are producing game controllers with Android 7 computers inside; for instance, an nVidia Shield or GameStick. Other major manufacturers, such as Amazon, manufacture a game console–iTV set hybrid product, such as the Amazon Fire TV. Others such as OUYA and GamePop make STB (set-top box) products that game controllers (and an iTV set) will plug into. Some, such as OUYA and Razer ForgeTV, come with both the STB and the game controller for a complete gaming package. Android 7 also features OpenGL ES 3.2 and the new Khronos Vulkan API as well.

Since all of these support Android, you can utilize digital video using MPEG-4 or WebM for digital video–based VFX projects, as well as vectors using SVG XML formats and SVG commands.

Visual effects project assets and special effects can be utilized inside the OpenGL ES 3.2 rendering engine, which renders many of today's most popular i3D games. The visual effects asset can also be used for advanced animated 2D texture-mapping effects.

Texture maps are applied as "skins" to 3D mesh geometry, and thus 3D can take your career to all new levels as you can now work as a texture-mapping producer for 3D filmmakers, i3D game makers, and 3D television producers.

Texture mapping can be done by using open source i3D software packages, such as Blender 3D. You can download **Blender 2.78** for free for Windows, Linux, or Mac at `http://www.blender.org` today and start to get up to speed on bridging digital video with 3D!

Future Devices: Robots, VR, and Home Appliances

The future of Android SDKs will surely bring more custom APIs. I expect to see **Android VR** for virtual reality goggles, as well as **Android HOME** for home appliances or home control units, and maybe even an **Android ROBOT** SDK for Android-based robots. I have already seen many of these product types in the marketplace, so it's up to Google to provide custom APIs in these product genres, all of which can be great visual effects as well as digital audio platforms. The platforms will provide all-new opportunities for visual effects designers, digital video editors, and i3D artisans, as well as for all multimedia producers and application developers who are digital video editors, 3D modelers and animators, or digital video effects artisans. The future is indeed bright for 2D and 3D visual effects artisans!

Digital video and digital effects will be important components in all of these emerging device genres. I'd expect that at least two of these genres, home appliances or VR, will showcase interactive visual effects as a way to increase user experience levels, because UHD home theaters will have the full attention of a viewer, comfortably seated upon their couch. Interactive 3D should be another area of increasing popularity.

Paid Software Platforms: iOS or Windows

This last section will cover formats that are not open source—that is, they involve paid software, and in the case of Apple, paid hardware, that will be required to develop for these platforms. Some of these require the company who owns the platform to approve (allow) your software before it can be sold

in the application store. It is important to note that you will be able to get around this approval process by developing using HTML5 for these platforms, or by using JavaFX. Therefore, you could still deliver content for your clients without having to invest thousands in hardware (for iOS) and software (Windows Visual C++ or C# software development packages). There is also a "code once, deliver everywhere" advantage to using platforms such as Amazon Lumberyard and Kindle, Oracle Java and JavaFX, and HTML5, as all of these can either run directly on, or output files for, multiple OSes (Windows, Linux, iOS, Android, etc.) and browsers spanning all genres of consumer electronics devices, most of which we have covered earlier in this chapter.

Apple iPhone and iPad: Supported Media Formats

Apple Macintosh and iOS support MPEG-4 H.264 AVC, as do each of the other popular platforms and devices in the world. Apple also supports its own proprietary **QuickTime Movie** format, as you might have expected. This is also used in Fusion 8, but it does not have the highest level of quality, as you will see in many of the Fusion 8 discussion forums. Apple OSes also support i3D using OpenGL ES 3, and therefore will also render interactive 3D content. Apple also has a proprietary 3D engine called Metal, but content optimized for this only runs on Apple hardware, not on the other one hundred manufacturers' products.

Apple applets support the **MP4** and **MOV** digital video data formats. The MPEG-4 format is also currently being used in Java, JavaFX, Android 7.0, Kindle 8, HTML5, CSS3, EPUB3, JavaScript, PDF, XML, JSON, Windows Mobile, RIM Blackberry, and many others.

Windows Phone: Supported Digital Media Formats

As a proprietary format, Windows and WinPhone do not directly support WebM and SVG, as all of the other platforms and devices in the world do. WebM support was added in Internet Explorer 9, and there is an extension you can get for Microsoft Explorer to render SVG file thumbnails. As Microsoft and Apple represent an increasingly smaller operating system market share percentage as time goes on, SVG file format support issues will become less of a problem for digital painters, digital illustrators, audio or video editors, visual effect designers, and i3D artisans.

Additionally, open source platforms are far easier to obtain and to qualify, and to post applications and content to. Open source platforms continue to be adopted more rapidly by second-tier as well as third-tier consumer electronic device manufacturers due to lower costs and the freedom of intended usage. This is the reason open source has taken over software.

Open platforms are also far more widespread in third-world countries, because they are used on the affordable devices and hardware platforms owned there. A great example of this is the Mozilla Firefox OS platform, where the user's hardware device plus OS will typically cost less than $25 US (or that country's currency equivalent).

This gives i3D VFX developers a far more widespread user base to work with and ensures that more people have access to their work. The trend toward open platforms benefits everyone.

Summary

In this final chapter, we took a look at multimedia and digital **visual effects publishing** concepts, principles, platforms, and file formats. You'll use these to publish, sell, and distribute the digital visual effects new media assets that you create.

You looked at several different data formats, publishing platforms, and hardware devices that will be available to you for developing digital visual effects and interactive 3D new media content, and you looked at the fusion of these in Fusion 8.

I hope you have enjoyed this journey through digital visual effects, digital images, masking, compositing, tracking, spline editing, particle systems, particle physics, and digital publishing concepts and related VFX pipeline work processes.

Now that you have a fundamental knowledge of digital visual effects that you can build on in the future for your new media design, multimedia development, and 3D content publishing endeavors, you can create the groundbreaking raster plus vector–based 3D application or 3D game that captivates users in the rapidly growing digital new media consumer product marketplace.

Be sure to keep your eye out for my other books covering Android Studio, Java or JavaFX, HTML5, Lumberyard, or other new media genres such as digital image compositing, interactive 3D, digital illustration, digital audio editing, digital painting, and digital video editing fundamentals, all currently available at `http://www.Apress.com`, if you want to expand your knowledge base regarding multimedia production for the popular open platforms.

Index

A

B

W. Jackson, *VFX Fundamentals*, DOI 10.1007/978-1-4842-2131-0

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X, Y, Z