

Audio Visualization Using ThMAD

Realtime Graphics Rendering
for Ubuntu Linux

—
Peter Späth

Apress®

www.allitebooks.com

Audio Visualization Using ThMAD

Realtime Graphics Rendering for
Ubuntu Linux



Peter Späth

Apress®

Audio Visualization Using ThMAD: Realtime Graphics Rendering for Ubuntu Linux

Peter Späth
Leipzig, Germany

ISBN-13 (pbk): 978-1-4842-3167-8

ISBN-13 (electronic): 978-1-4842-3168-5

<https://doi.org/10.1007/978-1-4842-3168-5>

Library of Congress Control Number: 2017960214

Copyright © 2017 by Peter Späth

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image designed by Freepik

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Natalie Pao
Development Editor: James Markham
Technical Reviewer: Massimo Nardone
Coordinating Editor: Jessica Vakili
Copy Editor: Kezia Endsley
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 <http://www.apress.com/rights-permissions>.

Apress titles 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 Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-3167-8. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

Contents

| | |
|---|-------------|
| About the Author | xi |
| About the Technical Reviewer | xiii |
| Introduction | xv |
| ■ Chapter 1: Sound Input..... | 1 |
| Preparing for Sound Input | 1 |
| Understanding Sound Structure | 2 |
| Time-Elongation Representation | 3 |
| Frequency-Power Representation | 3 |
| Input Data Taken from the Sound Card | 4 |
| Summary | 5 |
| ■ Chapter 2: Visualization Basics..... | 7 |
| Toolchain | 7 |
| Installing ThMAD..... | 7 |
| Connect PulseAudio Sound to ThMAD | 11 |
| Recording a Video..... | 15 |
| Making a DVD from Your Recording..... | 21 |
| Basic Samples..... | 21 |
| Basic 2D Sample | 21 |
| Basic 3D Sample | 29 |
| Summary | 35 |

- Chapter 3: Program Operation..... 37**
 - ThMAD Artiste Operation 37
 - Starting and Using Different Modes 37
 - Stopping ThMAD Artiste..... 42
 - Starting with Errors 42
 - ThMAD Player Operation 43
 - Starting and Using Different Modes 43
 - Stopping ThMAD Player 47
 - Creating and Installing Faders..... 47
 - Summary 49
- Chapter 4: 3D Concepts 51**
 - Coordinate Systems 51
 - Space Mapping 54
 - Spatial Operations: Translation, Rotation, and Scaling 57
 - Exposure to Light 58
 - Eye and Camera 61
 - ThMAD Meshes 64
 - ThMAD Particle Systems 64
 - Summary 65
- Chapter 5: Stories: Basic Level..... 67**
 - More 3D Rendering Pipelines 67
 - Transformations..... 67
 - Wireframes 79
 - The Ocean Module..... 86
 - Texture Mapping..... 90
 - Automatic Texture Coordinates..... 92
 - Floating Textures I..... 99
 - Floating Textures II..... 106

| | |
|--|------------|
| Blobs, Blobs, Blobs | 108 |
| Basic Blobs | 109 |
| Perlin Noise Blobs..... | 113 |
| Summary | 116 |
| ■ Chapter 6: Stories: Advanced Level..... | 117 |
| Backfeeding Textures | 117 |
| Blurring in Two Dimensions | 117 |
| Self-Similarity..... | 129 |
| Particle Systems | 145 |
| Waterfall | 146 |
| Image Bit Particles | 154 |
| Center Clamped Particle Systems | 159 |
| Ribbon Particles..... | 161 |
| Glowing Objects | 163 |
| Summary | 169 |
| ■ Chapter 7: ThMAD GUI Reference | 171 |
| ThMAD Artiste GUI | 171 |
| Starting and Stopping the GUI | 171 |
| The ThMAD Desktop and Its Parts | 171 |
| Window Modes | 174 |
| Fullwindow Mode | 175 |
| Performance Mode | 176 |
| Fullscreen Mode | 176 |
| The Main Menu..... | 177 |
| Module Choosers..... | 181 |
| The Module List | 181 |
| The Graphical Module Chooser..... | 183 |

- The Assistant..... 184**
- The Object Inspector 186**
- Saving and Loading States..... 186**
- Modules..... 188**
 - Module Types..... 188
 - Placing and Deleting Modules 189
 - Connecting Modules 190
 - Cloning Modules 190
 - Module Anchors: Parameters and Connectors..... 190
 - Drawing Connections Between Anchors..... 192
 - Enumeration Input as Module Parameter 194
 - Float Input as Module Parameter 194
 - Float3 Input as Module Parameter 195
 - Float4 Input as Module Parameter 197
 - Quaternion Input as Module Parameter..... 198
 - String Input as Module Parameter..... 198
 - Resource as Module Parameter 199
 - Sequence Input as Module Parameter 200
 - Exporting States 203
 - Macros..... 203
- Notes 206**
- ThMAD Player 206**
 - Starting and Stopping the GUI 206
 - Player GUI Operations..... 206
- Summary..... 207**

| | |
|--|------------|
| Chapter 8: ThMAD Module Reference | 209 |
| Screen | 211 |
| screen0 | 212 |
| Bitmaps | 212 |
| Filters | 212 |
| Generators | 219 |
| Loaders | 222 |
| Modifiers | 223 |
| Dummies | 223 |
| Math Modules | 225 |
| Accumulators | 225 |
| Arithmetic | 226 |
| Array | 232 |
| Color | 233 |
| Converters | 234 |
| Dummies | 236 |
| Interpolation | 237 |
| Limiters | 239 |
| Oscillators | 240 |
| Mesh | 247 |
| Dummies | 247 |
| Generators | 248 |
| Importers | 249 |
| Modifiers: Color | 251 |
| Modifiers: Converters | 251 |
| Modifiers: Deformers | 252 |
| Modifiers: Helpers | 255 |
| Modifiers: Pickers | 255 |

| | |
|------------------------------------|------------|
| Modifiers: Transforms | 256 |
| Particles | 258 |
| Segmesh | 259 |
| Solid | 259 |
| Texture | 265 |
| Vertices | 266 |
| Xtra | 269 |
| Particlesystems | 269 |
| Fractals | 269 |
| Generators | 271 |
| Modifier | 275 |
| Renderers | 278 |
| Basic | 279 |
| Mesh | 283 |
| OpenGL Modifiers | 288 |
| Oscilloscopes | 300 |
| Particlesystems | 301 |
| Shaders | 306 |
| Text | 308 |
| Xtra | 309 |
| Selectors | 310 |
| Sound | 319 |
| input_visualization_listener | 319 |
| midi → aka_apc40_controller | 321 |
| ogg_sample_* | 321 |
| raw_sample_* | 322 |
| Strings | 323 |
| System | 323 |

| | |
|---------------------|------------|
| Texture..... | 329 |
| Buffers..... | 329 |
| Dummies | 334 |
| Effects..... | 334 |
| Loaders..... | 335 |
| Modifiers..... | 337 |
| OpenGL | 341 |
| Particles..... | 343 |
| Macros | 344 |
| Summary..... | 344 |
| Index..... | 345 |

About the Author

Dr. Peter Späth has worked as an IT consultant with heavy focus on Java related development for over 15 years. Recently, Peter has decided to focus on his work as an author and working in a self-paced manner on software.

About the Technical Reviewer



Massimo Nardone has more than 22 years of experience in security, web/mobile development, cloud and IT architecture. His true IT passions are security and Android.

He has been programming and teaching people how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years.

He holds a Master's of Science degree in computing science from the University of Salerno, Italy.

He has worked as a project manager, software engineer, research engineer, chief security architect, information security manager, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years.

His technical skills include security, Android, Cloud, Java, MySQL, Drupal, Cobol, Perl, Web and Mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, and more.

He currently works as the Chief Information Security Office (CISO) for Cargotec Oyj.

He worked as a visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (in the PKI, SIP, SAML, and Proxy areas).

Massimo has reviewed more than 40 IT books for different publishing companies and he is the coauthor of *Pro Android Games* (Apress, 2015).

This book is dedicated to Antti Jalonen and his family, who are always there when he needs them.

Introduction

Sound visualization is about capturing the sound coming from outside, using a microphone or the line-in jack of your sound card, or coming from inside if a sound file is played on your computer. The software suite ThMAD allows for graphically building a sound pipeline, yielding real-time graphics to produce beautiful and interesting output from that sound. The introductory chapter depicts such a system, describes the targeted audience, and gives you some hints about text conventions and the preferred way of reading this book.

About Sound Visualization and ThMAD

You can buy a CD or listen to a radio station, or listen to music provided by a stream or other means. If you like the music it will give you an emotional and/or intellectual feeling, similar to how a good movie makes you feel. Concerning movies, there is an obvious marriage of musical sound and vision—movies have sound tracks and the combined pleasure is higher the more the sound track matches the story on-screen. Making a movie usually means you first have the scenes and speech assembled, and then you add the sound track. This is of course a high art if you want to make a good movie and a good combination of the two senses—sight and sound. If it is really good, one will amplify the other in either way.

As for audio visualization, you have a combination of visual and aural input to your senses. But it happens to be done differently compared to movies. The order is different and the coupling happens in an apparently more automated way. For audio visualization, the sound comes first and, based on it, the visual part of the joint artwork is generated by some program. As for the second assumption, I intentionally say apparently automated, because once you have a rendering program, you can start the program and enjoy the visualization of your music while you sit on a chair and do nothing. There is a third difference: the soundtrack of a movie is based on the plot, and it is by no means possible to change the plot but use the same soundtrack. For audio visualization the sound input can be different for the same visualization and you still get an interesting net result.

In order to get to a nice visualization and to have it become really good, you have to construct a rendering pipeline in some computer language, which could be an art itself. This has to be done manually. The good thing about it is that you don't need to be a master artist, nor a master software developer, from the beginning, since getting into it is not overly complicated and you'll be able to get your feet wet using the right program.

The Thinking Machine Audio Dreams (ThMAD, pronounced Thee-Mad) software suite is such a program. Once you learn the basics, you can start with some easy working rendering setups, and if you have enough time to learn, experiment, and build up sophisticated rendering pipelines, you will end up with magnificent audio visualizations.

ThMAD basically consists of a program named Artiste for generating a rendering pipeline, and a program named Player that you can use to run the pipeline in a visualization performance. While working inside Artiste, you see the realtime result of what you are doing, promptly and appropriately reacting to incoming sound according to changes you made, which gives a tremendous boost to your productivity. ThMAD and the surrounding system are depicted in Figure 1.

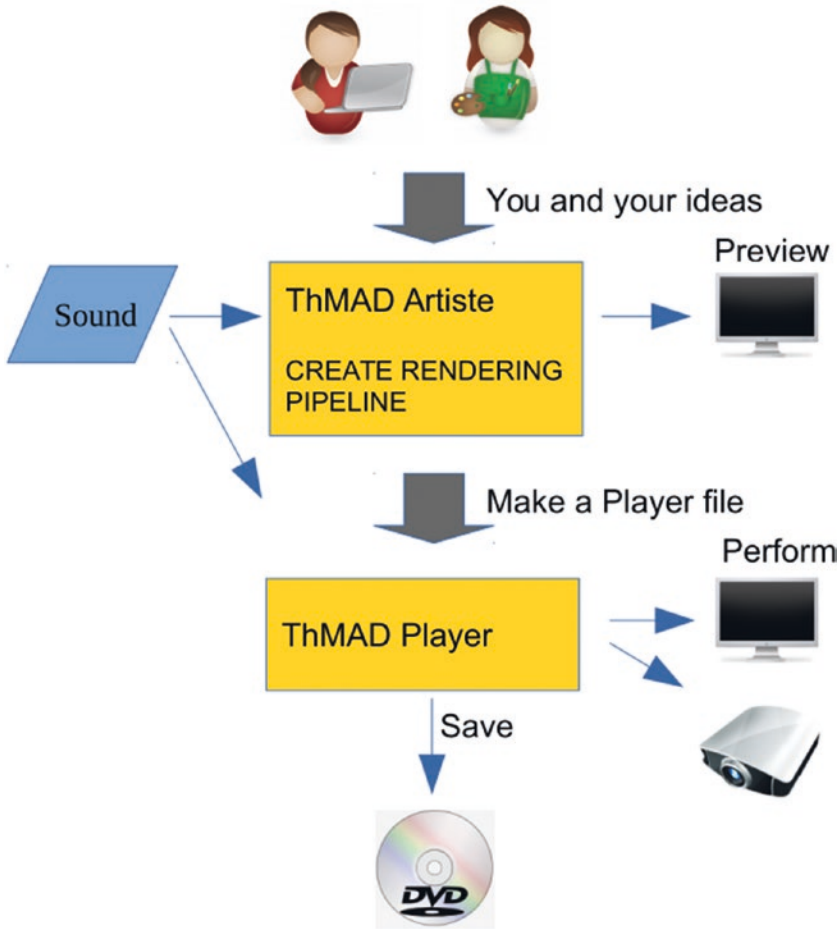


Figure 1. ThMAD and the surrounding system

The ThMAD software is based on the open source GNU GPL licensed (<https://www.gnu.org/licenses/gpl-3.0.txt>) software VSXu from Vovoid Media Technologies®. The fork has been taken from version 0.5.0. A considerable amount of work has been done on it since then, but some code has been introduced to allow for VSXu states to be loaded and interpreted correctly.

ThMAD is GNU GPL licensed, so the source is open to everyone. You cannot sell it or use it for a product you sell according to the GNU GPL. But you can use it for private or professional pleasure or work on it and give it or your own work away (only!) as a GNU GPL licensed software itself. You can find the source here: <https://sourceforge.net/projects/thmad/>.

■ **Note** VSXu is a registered or unregistered trademark of Vovoid Media Technologies AB, Sweden. Dr. Peter Späth is independent of Vovoid Media.

ThMAD is for Linux, more precisely Ubuntu. You can however run Ubuntu inside a virtual machine under Windows or whatever and use it there for development and experiments. But don't expect such a setup to give you high performance. For serious applications, it is strongly suggested you get an Ubuntu box. It is not hard nowadays to buy a PC or laptop with preinstalled Ubuntu, or install Ubuntu yourself. The latter might be tricky sometimes and under certain circumstances, you should expect to read a lot of documentation and blogs before you get it running smoothly. Of course, the Ubuntu homepage will help, and you can start with a trial version running from a bootable CD. Linux distributions other than Ubuntu will work as well, but you have to compile them yourself from the source. As far as this book is concerned, ThMAD was tested only with Ubuntu.

Under the hood the software OpenGL is used, and it's a widely adopted industry standard for sophisticated graphics rendering, including manufacturing and computer games. You will be able to use a wide range of graphics cards to see ThMAD running. Be cautious with cheap hardware though, especially for older graphics cards and older or overly downsized systems. You cannot expect each and every rendering pipeline to run smoothly, and some of the more advanced ones may even fail. Do yourself a favor and spend some extra money on at least a medium-grade computer and decent graphics hardware.

This book is based on ThMAD version 1.0.0. The associated OpenGL version as of the time of writing is 3.0.

The Book's Targeted Audience

This book is for artists with some IT background, or developers with artistic inclinations. Development experience is not required, but is surely helpful for some advanced features like shader programming. It is however not necessary to use ThMAD at all, and this book helps you with samples you can use as a working basis if the shader source code is involved.

This book is not a development guide, so hints about the actual implementation are given rather infrequently and only as anchor points for people who might feel an inclination to get into the development.

In order to fully understand ThMAD, and OpenGL based software in general, you need to know basics about 2D and 3D graphics and coordinate systems. However, most geometric concepts are explained to some fair extent.

Installation

The installation process of ThMAD is discussed in Chapter 2, “Visualization Basics”.

Conventions Used in This Text

Working with ThMAD involves using its modules, which are organized in a tree-like structure.

Modules are usually named `maths` → `converters` → `4float_to_float4`. Or in short with fixed width font, `4float_to_float4` if the module position inside the module tree is clear from the context.

State is the common notion for a rendering pipeline while constructing it. Finished states are also called *visuals*. References to sample states, including associated code provided with the installation, as well as informational hints in general, are highlighted like so:

■ **Note** This sample is as a source available under `A-3.2.1_Visualization_basics_basic_samples_basic_2d_sample` inside the `TheArtOfAudioVisualization` folder.

With the “Note” label replaced with “Tip” for informational hints. By folder in this context I mean a folder as showing up in the module lister or browser. Important notes and pitfalls are marked as follows:

■ **Caution** Due to the backfeeding it might easily happen... ..

... ..

Code and script snippets, as well as terminal input and output, usually show up in monospace font like here:

```
apt-get install libc6 libfreetype6 libgcc1 libpulse0 libstdc++6 libglfw3
```

Very small code snippets appear directly inside the text. If a longer line does not fit into a line, a trailing `␣` at each line of code signifies that while actually writing it, the `␣` must be removed and the subsequent line break must be discarded. For example,

```
echo "cmd [...] rectangle ␣  
abc [...]"
```

should be entered as

```
echo "cmd [...] rectangle abc [...]"
```


At many places, an asterisk `*` is used as a wildcard to denote any string. This frequently happens to refer to all the files inside a folder, or to file name patterns. Upon first startup, ThMAD Artiste creates a data folder for all your states and visualizations at

```
/home/[USER]/.local/share/thmad
```

and a symbolic link at

```
/home/[USER]/thmad
```

This points to the aforementioned If you are referring to the data folder inside this book, the link location is used.

How to Read This Book

This book can be read sequentially. Chapters 1 to 4 serve as an introduction, with Chapter 2, “Visualization Basics,” perhaps being the most important, since it describes the basic system setup and two important simple visualization examples.

Chapters 5, “Stories—Basic Level” and 6, “Stories—Advanced Level” contain a collection of independent stories or tutorials that you can work through in any order.

Chapters 7 and 8 are references that you can consult whenever you have doubts or questions while working through the stories, and of course you can use them to deepen your knowledge about modules and to get ideas for your own visualizations.

CHAPTER 1



Sound Input

In this chapter, we deepen the knowledge of how the computer can be prepared to capture incoming sound or produced sound, how the sound is represented internally, and how the data arrives at ThMAD. We distinguish between the obvious air pressure elongation versus time representation, and the power versus frequency representation, or *spectrum*.

Preparing for Sound Input

After you purchase a PC or laptop with Ubuntu Linux, or after you have Ubuntu Linux installed on your PC, you have basically two options—you can use external audio sources or you can let the sound play from the computer. As for the latter, you could use the CD player of your PC, you could play some files from a USB stick or your hard disk, or you could stream audio files via the Internet or some other means.

Note that other Linux distributions might work as well. Give it a try—chances are good that you'll find similar programs, tools, and settings to accomplish the same thing.

If you want to use external audio sources, you need a microphone or a sound card with a line-in to connect to. Especially for laptops, the built-in microphones are not of the highest quality, but they might be enough for your purposes. You actually don't want to accurately reproduce the sound, but react to it, and for this aim, having perfectly linear input curves is not too important. On the other hand, if you don't want to lose important impulses from the basses, which can happen with cheap microphones, getting yourself a decent microphone might help you avoid surprises. Also, bear in mind that audio visualizations might be brittle to the structure of the incoming sound under certain circumstances.

Usually you want to avoid that and the overall outcome should be interesting for any kind of music input. This is easy enough to check with different recordings. But if, for example, the basses never make their way through the audio hardware to a suitable extent, because your microphone misses the basses, your rendering pipeline might lack reactivity to an important part of the incoming sound. If instead for external sound input you just connect some audio source to the line-in jack of your computer, you are automatically on the safe side.

If you want to play CDs using your computer's CD player, or play audio files or streamed audio contents, e.g., using your browser, chances are good you don't have to do anything but start suitable programs or let the operating system do it for you automatically. For larger sound file collections, a program for administering them might be handy. RhythmBox, which is preinstalled on Ubuntu, is a good option.

The current version 1.0.0 of ThMAD primarily depends on PulseAudio, which is an audio routing server that handles all sound streams inside your computer. It knows everything that's captured or recorded, and everything that's played. Ubuntu Linux comes with PulseAudio preinstalled and automatically started; for other Linux distributions you may have to install it first.

ThMAD can also connect to ALSA, which is a low-level technology that talks to the sound hardware, and it can connect to JACK, which is a sound server that music professionals usually prefer. It is, however, considerably more difficult to use those options compared to PulseAudio, so we will as a sort of standard case use PulseAudio in the text.

For a graphical description of the standard PulseAudio sound chain, see Figure 1-1.

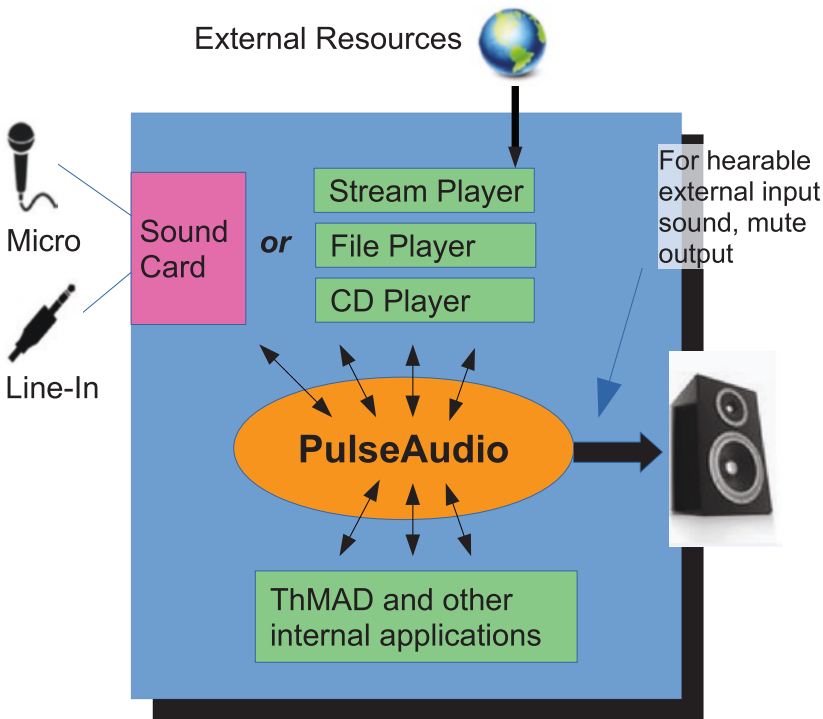


Figure 1-1. The PulseAudio sound server inside Ubuntu

Understanding Sound Structure

Sound is about air pressure oscillations that are received by your ears. From a mathematical or physical point of view, there are different representations for sound—the time-elongation (or time-pressure) representation and the frequency-power representation. Both of these are discussed in the following sections.

Time-Elongation Representation

On a diagram with the x-axis denoting the time and the y-axis denoting the pressure, the time-elongation representation might, for a sine wave, look like Figure 1-2.

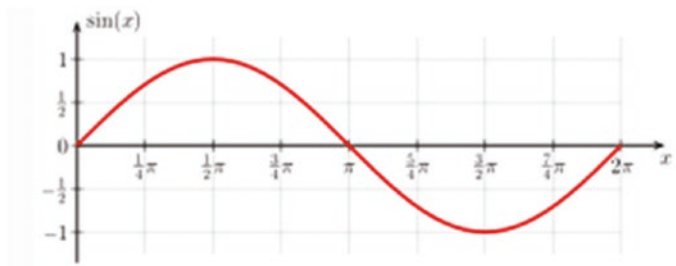


Figure 1-2. A sine wave

In computer systems, we need a digital representation for this sine wave. The idea is as follows: we divide the time into small time steps, say 44,100 steps per second, and for each time step, we write down the current air pressure, or y-value, and save it inside an array. This is sometimes called *analog-to-digital conversion* (ADC). Note that 44,100 is a widely adopted industry standard, for example, it's used with music CDs. Because we have two ears and like stereo, we do that conversion twice, for the left ear and the right ear. By that means, we end up with 88,200 numbers, which digitally represent one second of stereo sound. For the pressure or y-value representation, we use integer values (-32,768 up to 32,767), with the lower value representing a negative pressure offset, so maybe 0.997 bars, and the higher value representing a positive pressure offset, say 1.003 bars.

According to a scaling we can freely define, these could be mapped to 0.997 bars \rightarrow $y=-1000$ and 1.003 bars \rightarrow $y=+1000$. All the other numbers are mean pressures between these values. Of course we could use number ranges other than -32,768 ... 32,767, but the range we chose here is internally represented by exactly two bytes of data, and computers like that very much. It is also a trade-off: fewer different values means less resolution and poorer quality, and more different values means higher storage need.

Of course, in reality music is stored in lots of different formats, including MP3, Ogg Vorbis and others, mainly for reasons of saving space. The 88,200 numbers per second add up quite rapidly. But in case of letting an application like ThMAD access PulseAudio data, it will receive the data in an uncompressed and untransformed, raw format. This is nice, since then ThMAD doesn't need extra logic to handle different sound formats.

Frequency-Power Representation

A practically less obvious representation of sound consists of writing down the frequency distribution at each instant of time.

Consider the time range [10s;10.1s] when listening to some music. Instead of reporting the air pressure amplitudes at each instant, e.g., 10.000s, 10.001s, 10.002s,

we report the frequency mixture of the tones that arrive in our ears during some time range [10s;10.1s]. $A \cdot 100\text{Hz} + b \cdot 200\text{Hz} + c \cdot 300\text{Hz} + \dots$, where x Hz means a sine oscillation frequency of x per second, and the a, b, c, \dots are weights or power coefficients. The lower the number, the smaller the contribution and the higher, the larger the contribution.

Doing this in a mathematically concise way is called *Fourier Transformation*, and it turns out that it is a perfectly equivalent way of describing sound. In fact, if we have sound in a pressure versus time representation, we can apply a Fourier Transformation to transfer this into a power versus frequency representation without losing any information. That means the process is reversible and we have something like an *Inverse Fourier Transformation* to go back the other way. We don't show the mathematical details here; you can find a lot about that in other books and on the Internet.

Why we are mentioning Fourier Transformation here is that it turns out to be important for our audio visualization aim. Because instead of letting our visualization react on the elongation, which we *could* do but which bears the danger of losing things since changes happen so fast, we could also react to the powers of frequencies. Just think of reacting to bass beats in one way and reacting to the treble melody in a totally different way. Because beats happen at a much less frequent rate, maybe twice a second or something like that, the influence of the sound on the visualization is much more perceptible compared to the fast sound pressure oscillations.

To use such a representation inside ThMAD, the input from PulseAudio undergoes a Fourier Transformation and the frequency related powers we get from that can serve as an input to the visualization setup. We'll show you the details about that later.

Input Data Taken from the Sound Card

In any case, what the application will first receive from the sound driver, which is the software counterpart of the sound card, is an array of size N of elongation data. For example, if at an instant t_0 , we request 512×2 samples from the sound driver at $44,100$ Hz, with the "2" multiplier because we want to capture two-channel stereo data, this means we have $512/44,100 = 0.0116$ seconds of data from $t_0 - 0.0116$ to t_0 . This implies that, if we don't want to lose any data, we need at least 0.0116 seconds before we ask for the next chunk of data from the sound driver, and so on.

While for the first sound representation, pressure versus time, we are done by just providing the input array acquired from the sound driver to the visualization pipeline, for the power versus frequency representation we pass this array through the Fourier Transformation. We get an array with powers for the frequencies $1 \cdot f_0, 2 \cdot f_0, 3 \cdot f_0, \dots$, where for the case of a $44,100$ Hz sampling frequency and $512 \cdot 2$ acquired samples, f_0 calculates to $f_0 = 44,100 / 512 = 86.13$ Hz. For some Fourier Transformation algorithm intrinsics, the power for the highest frequency will not be for $44,100$ Hz, but only half of it— $22,100$ Hz. So, for an input of size 512 , the Fourier Transformation will yield 256 power values.

Summary

You learned how sound data arrives at your system and how that sound is represented internally. You learned that ThMAD primarily depends on Ubuntu's internally running standard sound server, called *PulseAudio*.

In the next chapter, you will investigate the toolchain necessary to construct audio visualizations. You will also learn how to install ThMAD on your system and what needs to be configured to have everything running smoothly. The chapter starts with two basic examples to be run inside ThMAD.

CHAPTER 2



Visualization Basics

This chapter is about the toolchain used for an audio visualization project and presents two basic audio visualization samples using ThMAD.

If you want to become an artist, one of the first things you'll have to do is learn how to use the tools to accomplish an oeuvre. You don't start with the most complicated setups though, apart maybe from whetting your appetite, you begin with simple things. Then you improve your proficiency step by step, maybe learn about a more and more complicated example of other artists' work, and in the end hopefully you can accomplish your own ideas no matter how complicated they are. Well, some people prefer to mix the stages and learn from the inside with more complicated setups. It is up to you. But learning the tools is inevitable in any case.

Toolchain

The toolchain tells us which program is needed for the complete work and presentation setup, starting from input, which is sound in this case, to output, which is graphics. The latter might show up in the monitor, or in the beamer, or inside a video file if sound and graphics are merged.

Actually, you don't need too much. Ubuntu Linux provides you with quite a lot. To play something or capture something from the outside, everything is already there. And PulseAudio, the sound server running inside Ubuntu, already knows about it and provides the sound data to programs that need it, like ThMAD.

What is left is a connection from PulseAudio to ThMAD, ThMAD itself, which is providing the output to a monitor or beamer, and a program to record and produce a video if you like.

Installing ThMAD

After you downloaded ThMAD from <https://sourceforge.net/projects/thmad/> as a Debian package that Ubuntu understands (Debian is the mother distribution of Ubuntu) with suffix `.deb`, you have to make sure the dependencies are fulfilled.

A future version might do this automatically, but for now you do it manually. ThMAD depends on the following packages, where entries marked with a ■ are most probably already installed on your Ubuntu Linux system from the beginning:

- libglfw3 (≥ 3.1)
- ■libc6 (≥ 2.17)
- ■libfreetype6 (≥ 2.2.1)
- ■libgcc1 (≥ 1:4.1.1)
- ■libgl1-mesa-glx (≥ 11.2.0) (or libgl1)
- ■libglew1.13 (≥ 1.13.0)
- ■libglu1-mesa (≥ 9.0.0) (or libglu1)
- ■libjpeg8 (≥ 8c)
- ■libpng12-0 (≥ 1.2.13-4)
- ■libpulse0 (≥ 0.99.1)
- ■libstdc++6 (≥ 5.2)

To install these packages, log in as root inside a terminal. Press Ctrl+Alt+T and then enter `sudo su` at the terminal. You'll be asked to enter your password. Do so then enter the following:

```
apt-get install libc6 \
libfreetype6 libgcc1 \
libgl1-mesa-glx libglew1.13 \
libglu1-mesa libjpeg8 libpng12-0 \
libpulse0 libstdc++6 libglfw3
```

where the `\` means the following newline (you press Enter or Return) gets ignored. If you want to enter the command in a single line, just ignore the backslashes.

Don't worry if the output says you already have . . . installed; it will not hurt if you try to install packages again, instead the command will simply ignore packages you already have on your system.

To install ThMAD itself, say you have downloaded it via browser and it ended up in folder Downloads in your home directory, you will install it, still as root, via

```
dpkg -i /home/[USER]/Downloads/thmad_1.0.0_amd64.deb
```

or any other version you get. [USER] should be replaced with your Linux username. All files will end up in `/opt/thmad`. After that, log off as root by pressing Ctrl+D. This is important for subsequent actions to not mess up your system.

For your convenience, launchers are available; you can place them via these commands on your desktop:

```
cp /opt/thmad/share/applications/thmad-artiste*.desktop\
~/Desktop/
cp /opt/thmad/share/applications/thmad-player*.desktop\
~/Desktop/
```

To see whether everything works, use the launcher for Artiste, or on the terminal, enter this:

```
/opt/thmad/thmad_artiste
```

A window should show up, as shown in Figure 2-1. Congratulations! ThMAD is now running on your system.

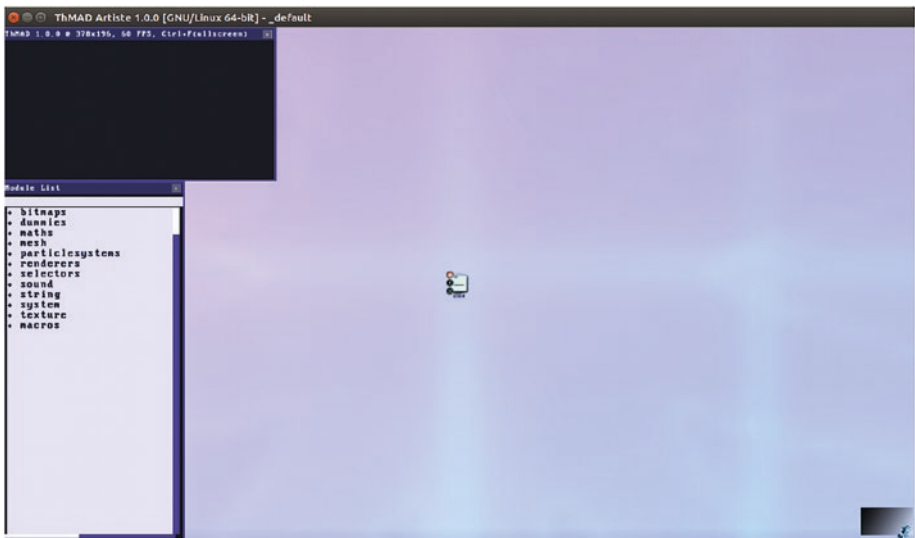


Figure 2-1. The ThMAD Artiste startup window

While the main installation folder can be renamed, the launchers depend on the installation residing in `/opt/thmad`. You could however edit the launchers appropriately, if you think a different installation folder is a better option for you.

As a last preparation step before actually using ThMAD Artiste, you might consider releasing the Alt key from the operating system. The default Ubuntu window manager, Unity, uses the Alt key to start the *Heads Up Display*, HUD, but ThMAD uses it as well for various GUI actions. To disable Ubuntu using the Alt key for HUD, or to change the key binding, go to the Keyboard section of the preferences, go to the Shortcuts tab, then to the Launchers menu. Select the Key to Show the HUD entry and press Backspace to disable it, or choose a new key or key combination to change the binding. See Figure 2-2.

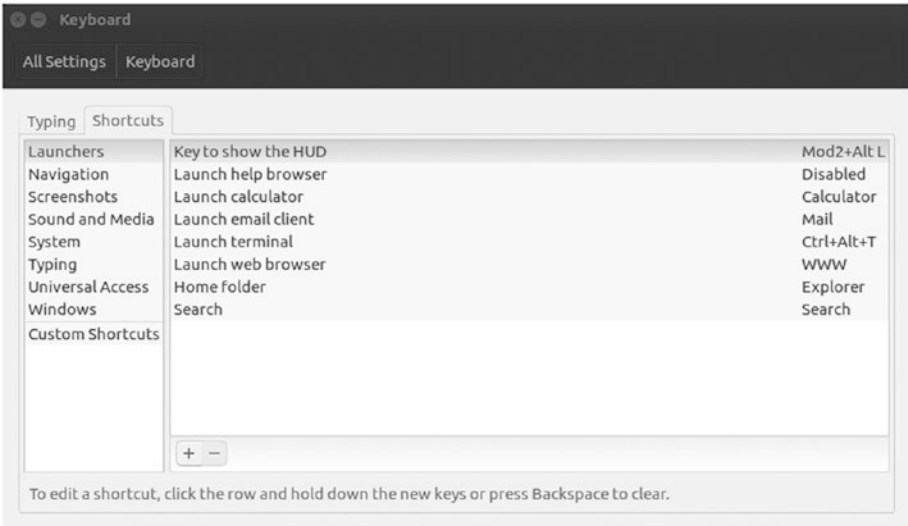


Figure 2-2. Releasing the Alt key in Ubuntu Unity

After clicking on Key to Show the HUD and pressing Backspace, it will be disabled. Or you can enter a different key or key combination to change the binding.

As an internal detail, for those who are interested in it, ThMAD version 1.x.x depends on OpenGL version 3.x.x. To find out which OpenGL version your system has, you can install the package mesa-utils:

```
sudo apt-get install mesa-utils
```

and then enter:

```
glxinfo | grep 'version'
```

This will give you something like

```
server glx version string: 1.4
client glx version string: 1.4
GLX version: 1.4
Max core profile version: 3.3
Max compat profile version: 3.0
Max GLES1 profile version: 1.1
Max GLES[23] profile version: 3.0
```

OpenGL core profile version string: 3.3 (Core Profile) Mesa 12.0.6

OpenGL core profile shading language version string: 3.30

OpenGL version string: 3.0 Mesa 12.0.6

OpenGL shading language version string: 1.30

OpenGL ES profile version string: OpenGL ES 3.0 Mesa 12.0.6

OpenGL ES profile shading language version string: OpenGL ES GLSL ES 3.00

The OpenGL core profile version string line points you to the OpenGL version in use.

Connect PulseAudio Sound to ThMAD

In order to connect ThMAD to Ubuntu's sound server PulseAudio, a couple of things are worth mentioning.

To see how PulseAudio connects things, the application `pavucontrol` comes in handy. It needs to be installed and, in order to do that from a terminal, open a terminal window via `Ctrl+Alt+T` and enter

```
sudo apt install pavucontrol
```

You will be prompted for your password. You can instead use the installer launcher



and enter `pavucontrol`. After the installation, you can run it from the starter. Press the Windows key, then enter `pavucontrol` and click on the launcher icon.

PulseAudio internally handles the following objects, reflected inside the `pavucontrol` program:

- *Playback.* Applications sending sound data toward PulseAudio are playback objects. Without running a music-playing application, you still will have one entry, called System Sounds, which belongs to the operating system. With a music-player at work, say RhythmBox (it is installed with a standard Ubuntu installation), it will look like Figure 2-3.

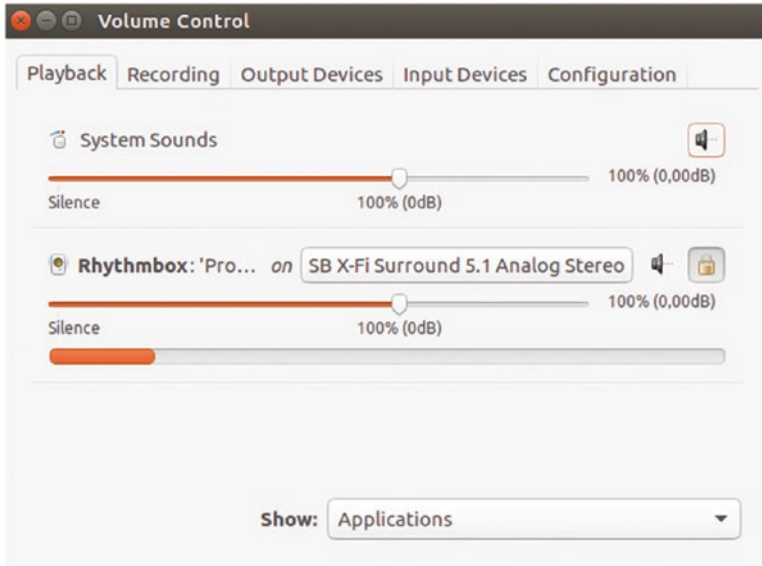


Figure 2-3. PulseAudio playback objects

You can see RhythmBox is currently sending something to my sound card called SB X-Fi by looking at the orange meter bar below it.

- *Recording.* Applications reading sound data will be listed here. ThMAD will later show up here.
- *Output Devices.* The sound cards installed on your system show up here, more precisely their playback channels. For me it lists, among others, SB X-Fi Surround 5.1 Analog Stereo.
- *Input Devices.* The sound cards installed on your system shows up here, more precisely their recording channels.

Input channels are microphones, but also *monitors*, and they plug to the soundcard's output, say speakers or earphones. For me it lists, among others, Monitor of SB X-Fi Surround 5.1 Analog Stereo. See Figure 2-4. This is important, because that is where ThMAD connects.

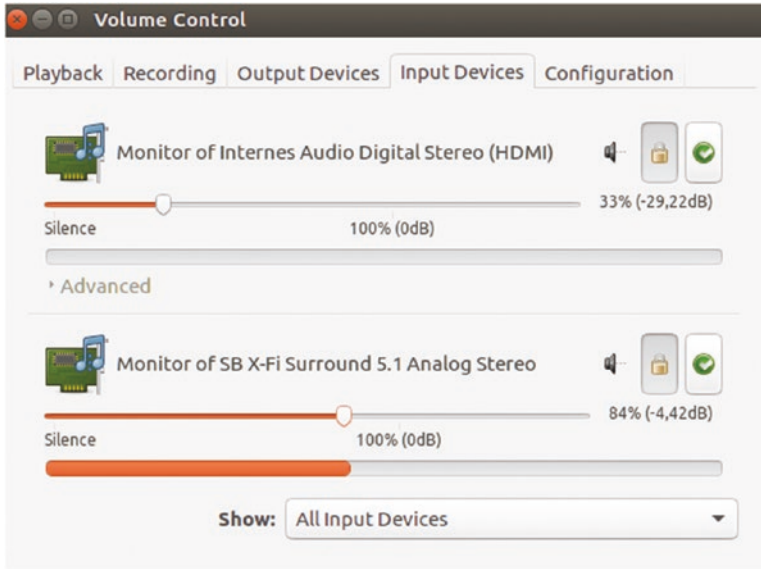


Figure 2-4. PulseAudio input device objects

You can see in Figure 2-4 that PulseAudio is currently providing sound data to the monitor *Monitor of SB X-Fi* by looking at the orange meter bar below it.

To actually do the connection, start ThMAD Artiste and drag the menu item `input_visualization_listener` from the Sound section from inside the left modules menu to the canvas. To make it appear, click on Sound inside the left menu to open the submenu. The module `input_visualization_listener` will then appear and you can drag it to the canvas. See Figure 2-5. You can use the mouse wheel to zoom the canvas in or out in order to make the module symbols appear bigger.

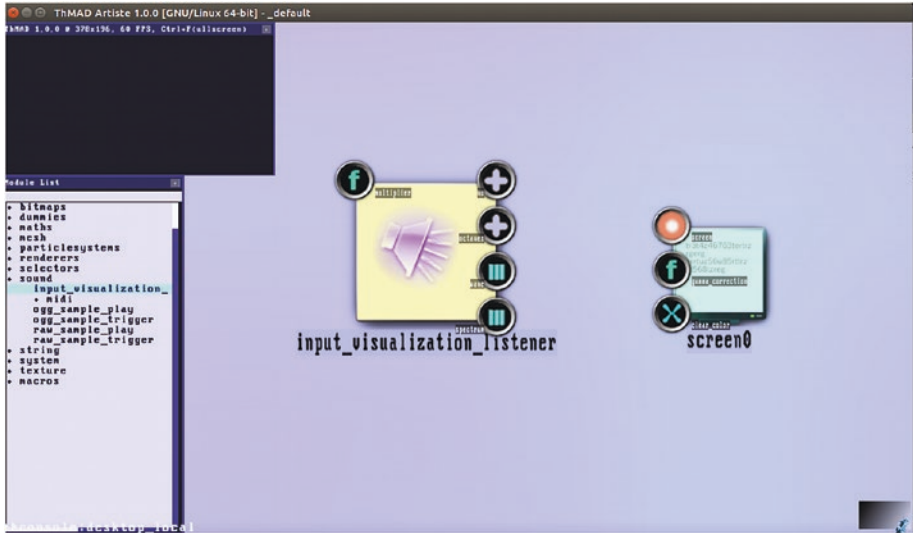


Figure 2-5. Open the ThMAD sound module

Then open pavucontrol if it's not already opened. Go to the *Recording* section and connect thmad to a monitor of your audible sound card output channel, as shown in Figure 2-6.

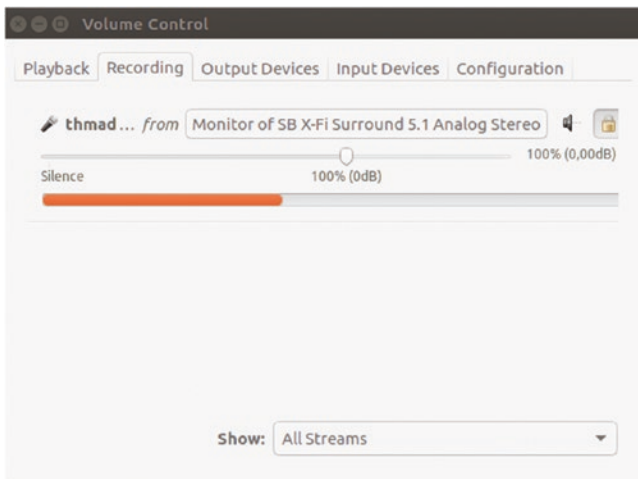


Figure 2-6. Connecting ThMAD to a PulseAudio monitor


You can see its monitor called *Monitor of SB X-Fi* sends some sound data to ThMAD by looking at the orange meter bar below it.

The connection will be saved automatically, so you don't have to do all that again later. Just play some music and start ThMAD, either Artiste or Player, and it will connect automatically to the same sound source you provided in the configuration steps.

Recording a Video

Especially after you made an exceptionally beautiful visualization, you maybe want to create a video file from that. It is currently not possible to directly create a video file from ThMAD, but Ubuntu helps you with that. So the idea is to start ThMAD, play a music file, and let Linux create a video.

To start, install the SimpleScreenRecorder program using the Ubuntu Software

Center . You can also start it from the terminal. (Remember, press Ctrl+Alt+T to open a terminal and enter `sudo su` to become root.) Then use this command:

```
apt-get install \
simplescreenrecorder
```

Do not confuse the *simple* in the name with “poor”. Actually this tool is quite powerful and dependable—it creates high-resolution video files and includes the sound from PulseAudio or ALSA or JACK.

Start ThMAD, open your visualization, resize the window, maybe to the maximum available size of your monitor, then switch to fullwindow mode via Ctrl+F. You can possibly remove the status line via Alt+T. Then start the SimpleScreenRecorder program by pressing the Windows key and then entering `simplescreen`. Its initial page is shown in Figure 2-7.



Figure 2-7. SimpleScreenRecorder page 1

Click on Continue. The next page shows the input parameters for the part of the screen to be captured; see Figure 2-8.

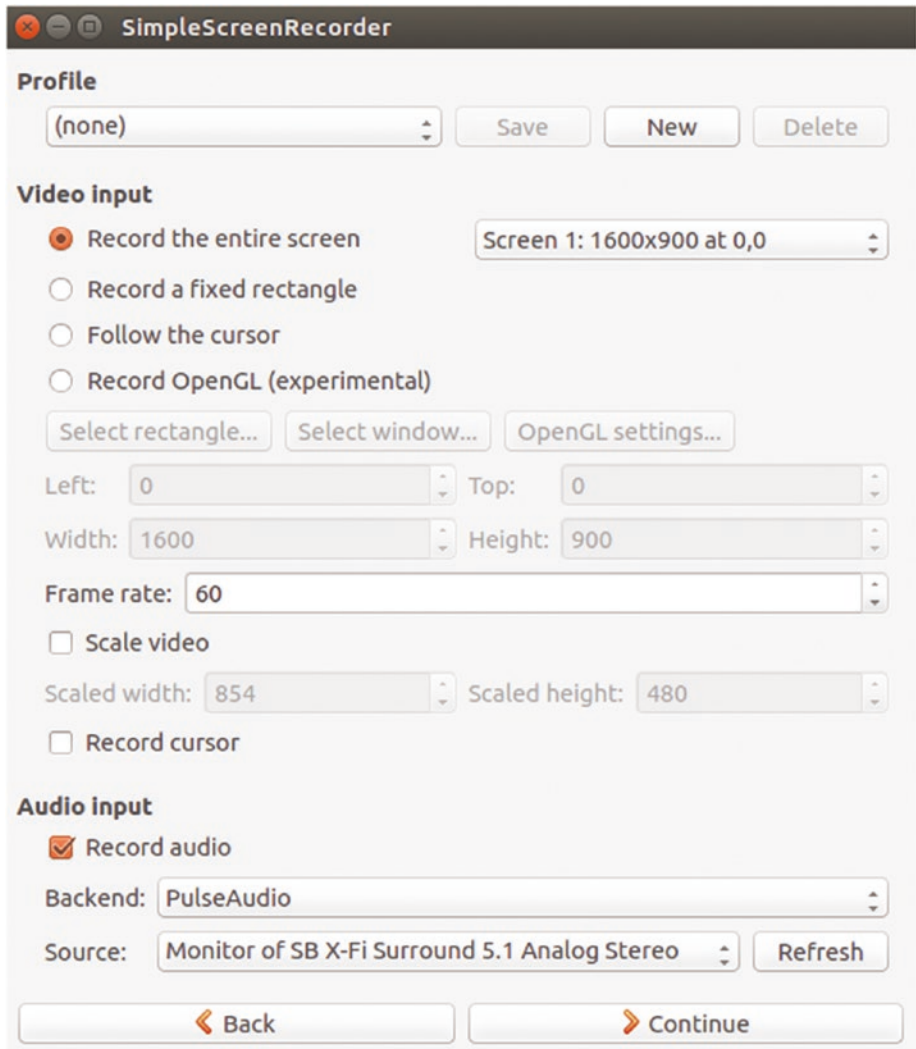


Figure 2-8. SimpleScreenRecorder page II

Select Record a Fixed Rectangle, then click the Select Window... button and on the Select Rectangle... button. Then click on the top-left position of the inner part of the ThMAD window without releasing the mouse button, and drag to the bottom-right position, releasing the mouse button. Clicking somewhere on the ThMAD window will eventually record the title bar and borders of the window. If you want, you can also choose the mode. Do not move or resize the ThMAD window after this step; otherwise, the wrong part of the screen will be recorded. On the same page, the frame rate can be specified. Usually the default value of 60 will do quite well, but you can change it to 100 for better quality or to 20 for a smaller video size.

Make sure the Record Audio checkbox is checked, select PulseAudio as the Backend, and choose the appropriate sound monitor.

Click on Continue again. On the next page (see Figure 2-9), enter the file name for the video file. All other values can be set to the value shown, but of course you can experiment with other values:

- Change the Container type to a different video format.
Matroska will do well for Linux, for other operating systems other values may work better. If you hover the mouse over the box, more info is displayed.
- Change the Codec type to a different video compression type.
H.264 is a good choice here. Again, hovering the mouse over the box shows more info.
- For the Audio Codec, select Vorbis, which is the recommended value. For information about the other values, hover over the box. The default rate, 128 Kbps, is a reasonable rate, but you can choose a different value.

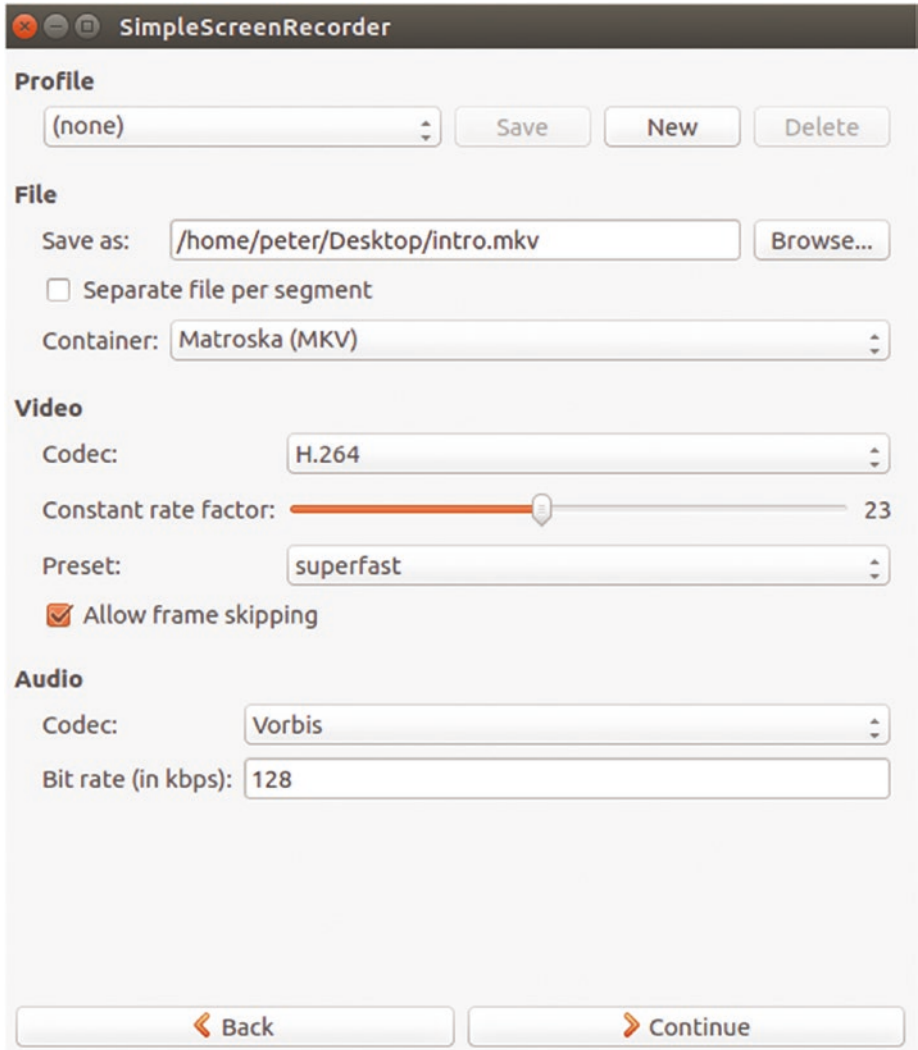


Figure 2-9. SimpleScreenRecorder page III

Click on Continue to show the last page, as shown in Figure 2-10. If you enable the Recording hotkey as shown, you can start recording via some keyboard sequence. Shown is Ctrl+R, but you can try others of course.

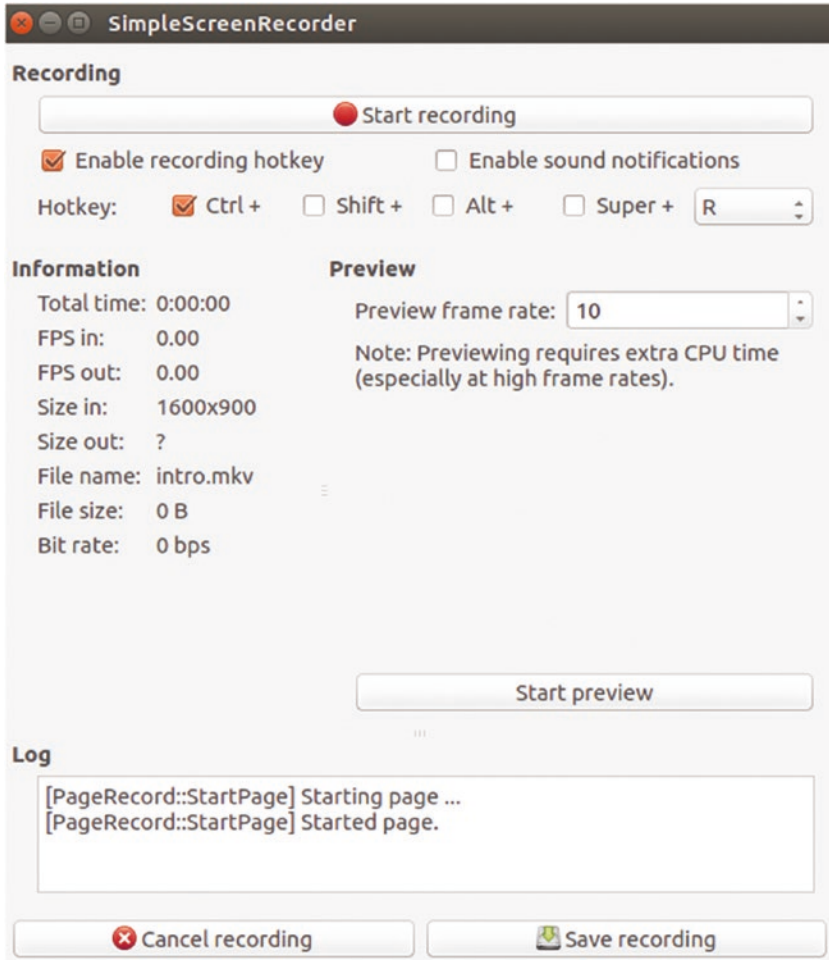


Figure 2-10. SimpleScreenRecorder page IV

Start the recording by clicking Start Recording or pressing the hotkey. Immediately afterward, start your music player. The video is now being recorded. Make sure the FPS In and FPS Out values are similar and close to the frame rate you specified. Otherwise, your system might be too weak and you should be more conservative with the performance related values chosen inside SimpleScreenRecorder. This especially comprises the recorded window size and the frame rate.

When your performance has ended, click on Save Recording, and you are done.

If, as a final step, you want to make a DVD from that, the programs discussed in the next subsection can help you do so.

Making a DVD from Your Recording

If you want to create a DVD from a sound file, you have several options. Open the Ubuntu

Software Center  and enter `dvd` to get a list.

The one that worked well for me is called *DeVeDe*. If you want to try that, install it and start it from the launcher by pressing the Windows key and entering `devede`. From the startup screen, choose VideoDVD. Add the MKV file you generated in this chapter or add several if you have more than one and want to put them all on one DVD. Then click on the Forward button. There choose a folder where intermediate data will be written to—what you enter there doesn't matter, but it should not exist before. Click on OK. After the intermediate files have been created, you have a chance to directly burn them on a DVD. Needless to say, you need a DVD burner on your computer for that option.

The program allows for tailored menus as well. This is not described here in detail, but you can click the Help button. The help shows up in the startup screen and gives you a fairly good introduction into the advanced features of *DeVeDe*.

Another one is called *DVD Styler*. It has more options compared to *DeVeDe* if you want a nice looking DVD menu. It also has a good introduction and a good manual included in the Help menu.

Basic Samples

In this section, we present two basic samples, one in 2D and the other in for 3D.

Basic 2D Sample

As a first sample, we want to describe a setup with a rendering pipeline consisting of a simple 2D rectangle reacting to some sound input from a music player.

■ **Note** This sample is as a source available under `A-2.2.1_Visualization_basics_basic_samples_basic_2d_sample` inside the `TheArtOfAudioVisualization` folder. You can load it via the Open... option if you right-click on an empty spot of the canvas.

First start ThMAD Artiste. If you installed it as described in this chapter, you can double-click on the launcher icon on your desktop, or you can open a terminal via `Ctrl+Alt+T` and enter `/opt/thmad/thmad_artiste`. In case you use the terminal, you'll see some diagnostic output especially showing the sound driver context. This might help under certain circumstances.

The ThMAD Artiste canvas will now show up, and if you started it the first time and without previous work, it will just show a very tiny module icon in the center. The size of any GUI elements, including text fonts in ThMAD, is relative, so you might first want to maximize the window to improve readability. Next you can zoom into the scene and pan

the view to move the viewport. For zooming, you can press W and R on the keyboard or use the scroll wheel of your mouse. For panning, use the arrow keys or keys S, D, F and E, or click into an empty part of the canvas and drag the mouse by clicking and moving it. Your screen will now look like Figure 2-11.

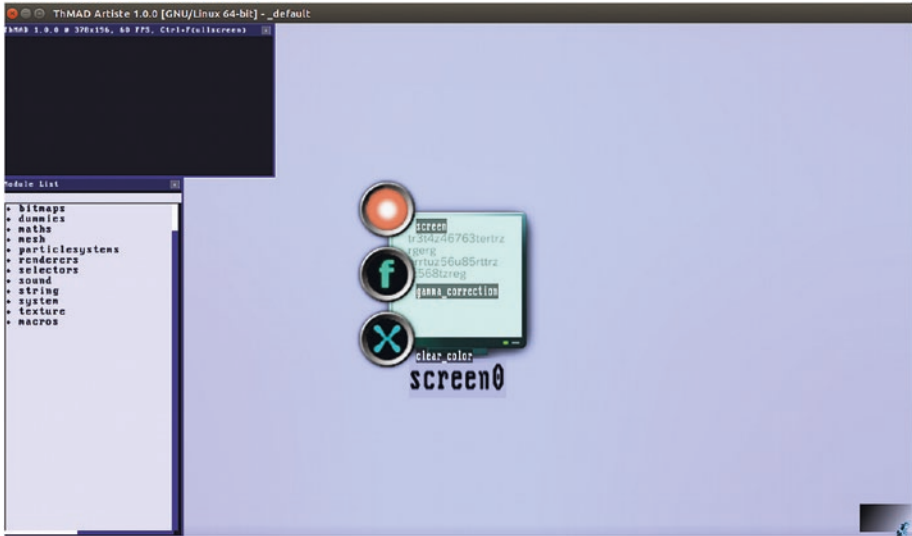


Figure 2-11. *ThMAD Artiste start screen, zoomed in*

In the bottom-right corner, the minimized assistant is shown. Press the Tab key to cycle through different sizes. While the assistant shows some information about how using the GUI, a more complete description of the GUI is found in Chapter 7 “Artiste GUI Reference”. The assistant has different modes and shows different contents. You can change mode or contents by right-clicking on the assistant and selecting one of the entries that appears; see Figures 2-12 and 2-13.



Figure 2-12. Assistant modes

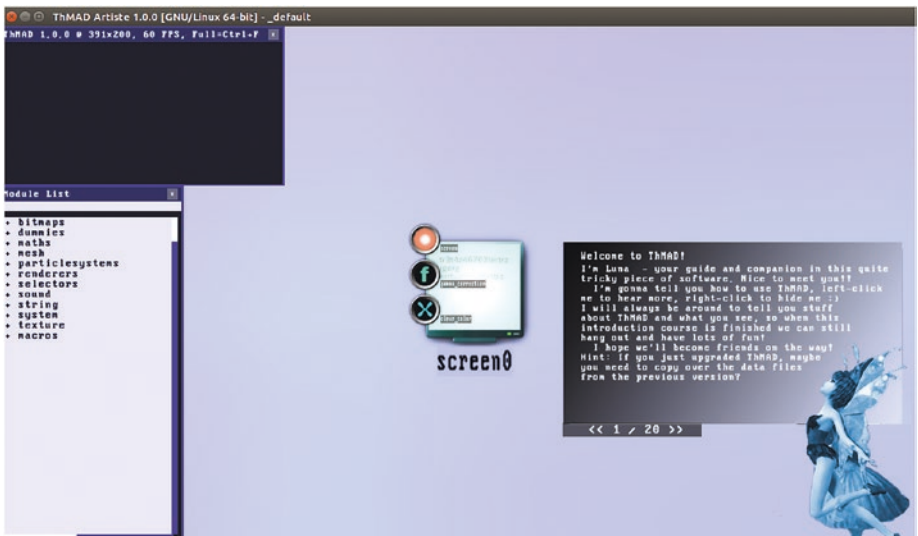


Figure 2-13. The assistant

If you choose Automatic Mode, the assistant will afterwards show context related information. Whenever you click on a module icon in the canvas, some information about it will be shown inside the assistant. Clicking on Courses enables you to read one of a couple of courses that help you improve your proficiency in using the GUI. A key and mouse usage reference is shown when you choose Keys/Mouse Reference.

For the rest of this book, we have the assistant minimized. Remember, you can do that by pressing the Tab key a couple of times.

Now we want to draw something. To accomplish that, click on Renderers in the module menu to the left, and then click on Basic. From the items showing up then, click on `colored_rectangle` and drag it to the canvas, next to the `screen0` module. See Figure 2-14.

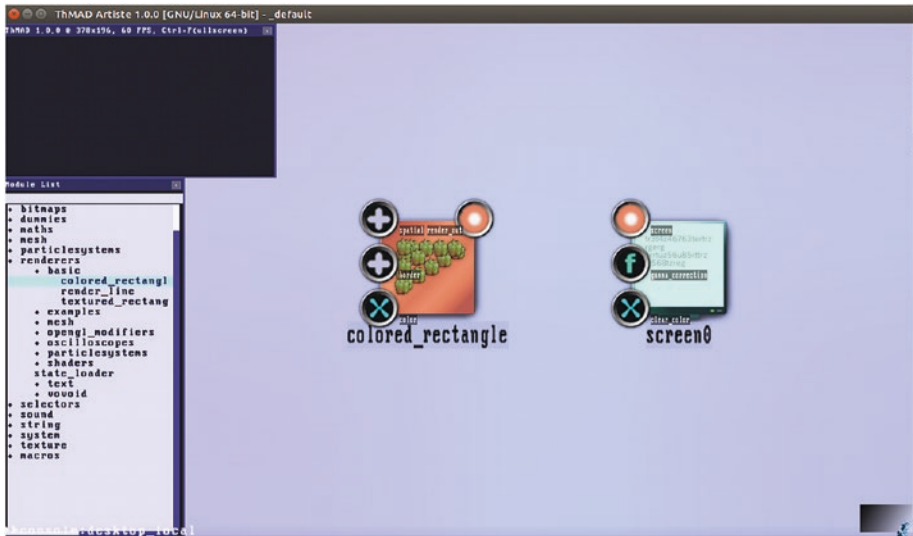


Figure 2-14. Placing a rectangle on the canvas

In order to see something in the black preview window, the modules still need to be connected.

Nothing happens yet. You first need to connect the output from `colored_rectangle` to the input of `screen0`. See the anchors at the borders of the modules? As a general rule, equally shaped and colored anchors can be connected, and this is what we will do here. One anchor of `colored_rectangle` has the label “render_out” and one anchor of `screen0` has the label “screen” and they look alike. To connect them, click on either of them and drag it to the other. The connection will then be shown on the canvas, and the preview window in the top-left will show the rectangle. See Figure 2-15.

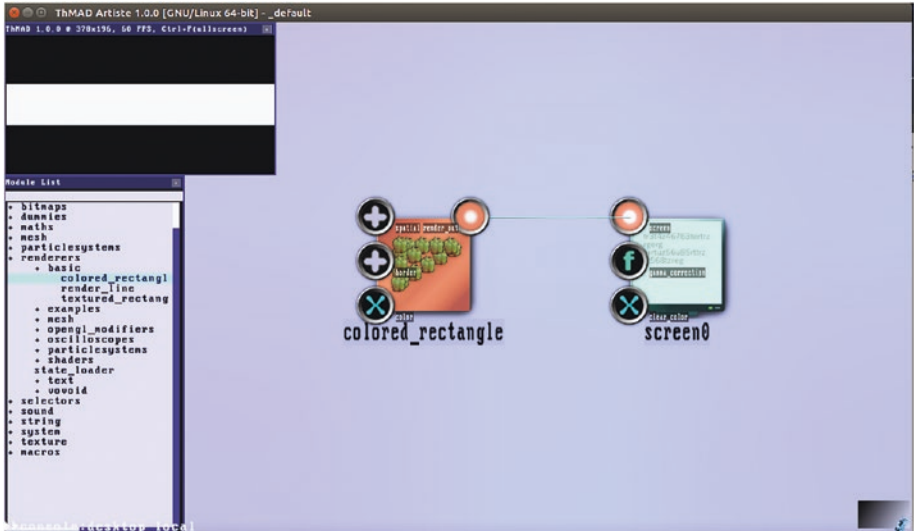


Figure 2-15. Connecting and drawing the rectangle

You might wonder how the drawing size and position in ThMAD are determined. With 3D sketches it is more complicated, but we will come back to that later. For our current 2D sketch, the drawing canvas covers a coordinate area left bottom $(-1; -1)$ to right top $(+1; +1)$ inside the x-y plane, and the position of the rectangle is chosen such that its center matches the center of the canvas $(0; 0)$. Also its size, when first placed on the canvas, is 2.0×0.6 . Different shapes follow different conventions, but usually the shape center will by default be at $(0; 0)$.

The drawing output has been shown in a small window on the top-left corner of the canvas, but you can change the view mode. Press `Ctrl+F` to maximize the preview window, and afterwards press `Alt+F` to create an overlay if you like. This overlay fully mixes graphical output and the modules canvas, including all GUI input functionalities. This is an extremely powerful feature, which is why it is called *performance mode*. Pressing either `Alt+F` or `Ctrl+F` again will change back to the previous view mode.

Next, we want use sound to dynamically change the orientation of this rectangle. For this aim, we place the `input_visualization_listener` module on the canvas. You find it in the Sound section of the left module. Click on Sound and you can see it. Or, you can enter the name of the module in the search field of the window. Click onto the empty space right under the title “Module List” and then enter `input_visualization_listener` or a suitable part of it. You will realize that just the first three letters will already filter the complete list to the one we are looking for. Clear the search line if you want to disable the filter. Clearing can be done by using the usual Backspace or Delete keys, or by pressing `Ctrl+Del` once. Or, as a third means to locate a module, double-click on an empty part of the canvas to show the graphical module browser (see Figure 2-16). There you’ll find it as well. Click at an empty point and drag the mouse to navigate in that browser. Hover over an entry to read info about it, and click and drag on an entry to place it on the canvas. Right-click on an empty point to close the browser.

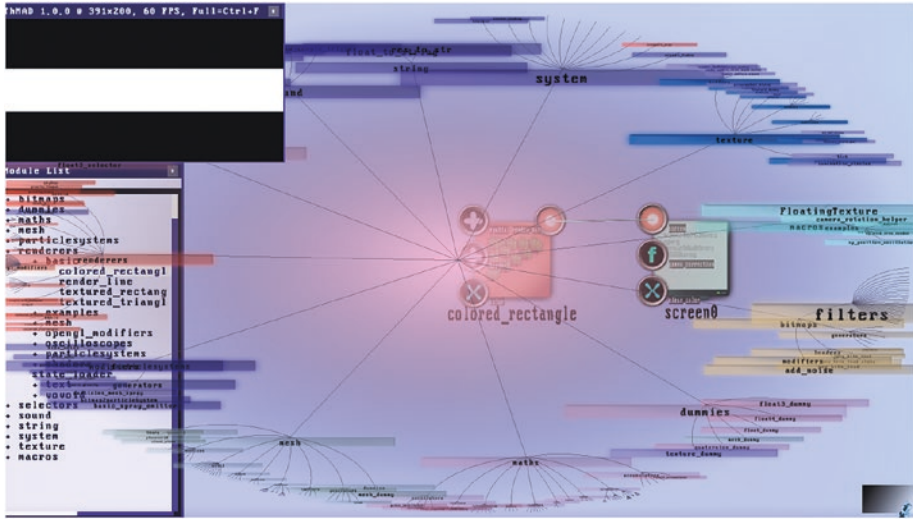


Figure 2-16. The module browser

After you dragged the module `input_visualization_listener` on the canvas, you want to connect the current sound volume to the drawing angle of the rectangle. See the `vu` anchor on top right of the `input_visualization_listener`, and the `spatial` anchor on the top-left of the `colored_rectangle` module? Both are so-called *complex* anchors, depicted by a +. They have sub-anchors. You cannot connect complex anchors, only non-complex anchors or sub-anchors. To unveil a module’s sub-anchors, click on them; see Figure 2-17. Now connect either of the `vu` sub-anchors of the `input_visualization_listener` module to the `angle` sub-anchor of the `colored_rectangle` module.

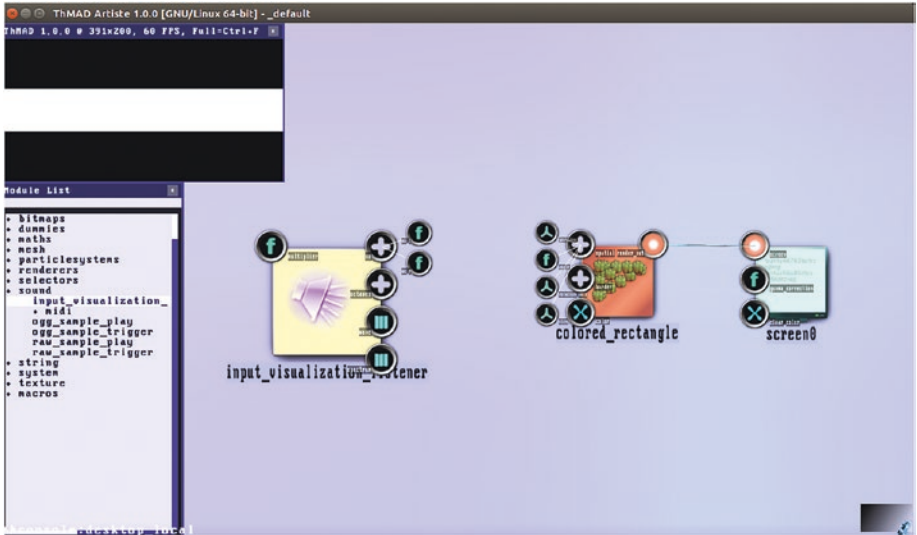


Figure 2-17. Sub-anchors opened

If sound is playing and everything is set up the correct way, you will see the rectangle wobble around to the rhythm of your music; see Figure 2-18.

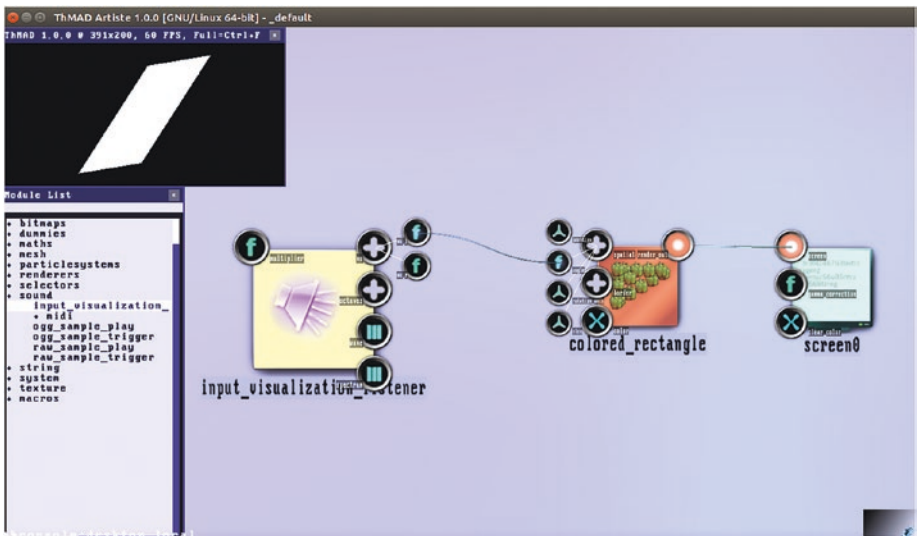




Figure 2-18. Wobbling rectangle

Congratulations, you just made your first audio visualization! To save it, just move to the cursor to an empty place on the canvas and press Ctrl+S. The screen will flash shortly to tell you the pipeline has been saved.

If you don't see the rectangle wobbling around, check whether the PulseAudio sound server is running and that it is sending data to ThMAD. If it is not running, you won't hear any music playing under PulseAudio anyway, but in addition to that, the sound icon of the Ubuntu desktop should look like this: , i.e. not grayed out or crossed out like that: .

To see whether PulseAudio sends data to ThMAD, cross-check with 1.2. As a last check, start ThMAD Artiste from a terminal by pressing Ctrl+Alt+T. Then enter `/opt/thmad/thmad_artiste`. Inside the terminal, you should see a diagnostic output like this:

```
-----
rtaudio_record.h audioprobe()
Audio Type = Linux PulseAudio
Available APIs:
  Linux ALSA
  Linux PulseAudio

Current API: Linux PulseAudio

Found 1 device(s) ...

Device Name = PulseAudio
Probe Status = Successful
Output Channels = 2
Input Channels = 2
Duplex Channels = 2
This is the default output device.
This is the default input device.
Natively supported data formats:
  16-bit int
  32-bit int
  32-bit float
Supported sample rates = 8000 16000
22050 32000 44100 48000 96000
```

Especially important in that list are the following entries:

- Found 1 device(s) (or more than one). If this is zero, it means ThMAD cannot attach to the sound server.
- Device Name = PulseAudio. This is the default sound server. Expect to do more work to actually set up ThMAD for non-PulseAudio drivers like ALSA and JACK. At the least you have to add the `-sound_type_alsa` or `-sound_type_jack` option to use ALSA or JACK. If you are not using the default device, you also have to specify `-snd_rtaudio_device=<NUMBER>` where `<NUMBER>` can be determined from the program output in the terminal.

- If 16-bit int is not a supported format, you are probably using a very rare sound card, or the sound driver is broken. If 44,100 is not in the list of supported sample rates, you can ask ThMAD to use a different sample rate. This can only be done while starting ThMAD from a terminal, and all you have to do is add `-snd_sample_rate=48000`, for example.

Basic 3D Sample

ThMAD acts on top of OpenGL, which is a 3D rendering technology used in games and industrial grade software projects and products. Given that information, we'd feel kind of disappointed if ThMAD wasn't capable of providing us with 3D rendering capabilities. The good news is it is.

It is, however, worth telling, that some important aspects need to be taken into account from the start in order to see anything at all in the 3D world of ThMAD:

- In many cases we need to define the surface material of the object.
- We need to deal with object surface parts that are not visible because they are hidden behind other objects.
- In many cases we need light.
- We need a camera to project the 3D scene onto a 2D monitor.

A lot more can be said about 3D rendering, and you can gain more insight by reading Chapter 5 “3D Concepts,” but in order to point out the most basic and most important aspects, we provide a very basic 3D rendering pipeline.

But before that, if you followed the path and constructed the 2D sample sketch, you might have missed a word about naming the file. The sketch in ThMAD is more commonly called *state* and it has to be saved inside. The reason is, when ThMAD starts with its default configuration it will automatically load a *state* named `_default`, and when you press Ctrl+S at any point, the current *state* will be saved again under the name `_default`. Consider `_default` as a temporary working name for *states*. However, if you want to save it properly for later, you should give it a decent name. To do so, start the ThMAD Artiste if it's not already running and showing the *state* you want to save, right-click at some empty spot, and from the pop-up menu, choose Save As.... In the first line of the pop-up that appears, enter a name for your state. Enter something at the other lines, too; they are all mandatory. Then click the OK button.

■ **Note** This sample is available as a source available under `A-2.2.2_Visualization_basics_basic_samples_basic_3d_sample` inside the `TheArtOfAudioVisualization` folder.

Now quit ThMAD Artiste and start it again. In the window title, you'll again see that it loaded the `_default state`. If the `_default state` is not empty, in order to commence with an empty `state`, right-click at an empty spot of the canvas, navigate to New ► and click on Empty Project. All modules except for the `screen0` module should now be gone.

Note that this startup behavior of ThMAD, more precisely automatically and unconditionally loading the `_default state`, can be tweaked in the configuration accessible from the main pop-up menu. There, you can set it to load the last saved state instead of the default. For the 3D pipeline, now commence as follows:

1. 3D objects in OpenGL and ThMAD are defined by points building a *mesh*, which in turn defines the surface of the object. A cube mesh, for example, consists of six faces and eight points. You start with such a cube mesh, which in ThMAD is called `mesh_box`. Inside the module menu, you'll find it by choosing Mesh → Solid → `mesh_box` or under the same coordinates in the module browser. Remember, double-click on the canvas to open the graphical module browser. Place `mesh_box` by dragging it on the canvas.
2. The `mesh_box` defines the 3D object, but it doesn't render it. An appropriate renderer is called `mesh_basic_render` and you can get to it at Renderers → Mesh → `mesh_basic_render`. Place it next to the `mesh_box` module and connect them—anchor the “mesh” of the former to the “mesh_in” of the latter.
3. Next you need to tell ThMAD to take care of hidden surfaces. Place the module Renderers → `opengl_modifiers` → `backface_culling` on the canvas and connect it to the `render_out` anchor of `mesh_basic_render`. Switch the status anchor of `backface_culling` to ENABLED.
4. Now the 3D object needs surface properties: navigate to Renderers → `opengl_modifiers` → `material_param`. Place this on the canvas and connect it to `backface_culling`.
5. To free yourself from eternal darkness, you need light. In ThMAD Artiste, choose Renderers → `opengl_modifiers` → `light_directional`. Place the light next to `backface_culling` and connect them. Open the Properties complex anchor by clicking it, then switch the enabled anchor of `light_directional` to YES.
6. You are almost done. To actually look at the 3D scene, we still need a camera. There are several and we choose this one: Renderers → `opengl_modifiers` → `cameras` → `orbit_camera`. Place it next to the light and connect them.
7. Finally, connect the `orbit_camera` to `screen0`.

At last you can see something; see Figure 2-19. This result is a little boring because we are looking straight toward the x-y-plane and thus only see one face of the cube.

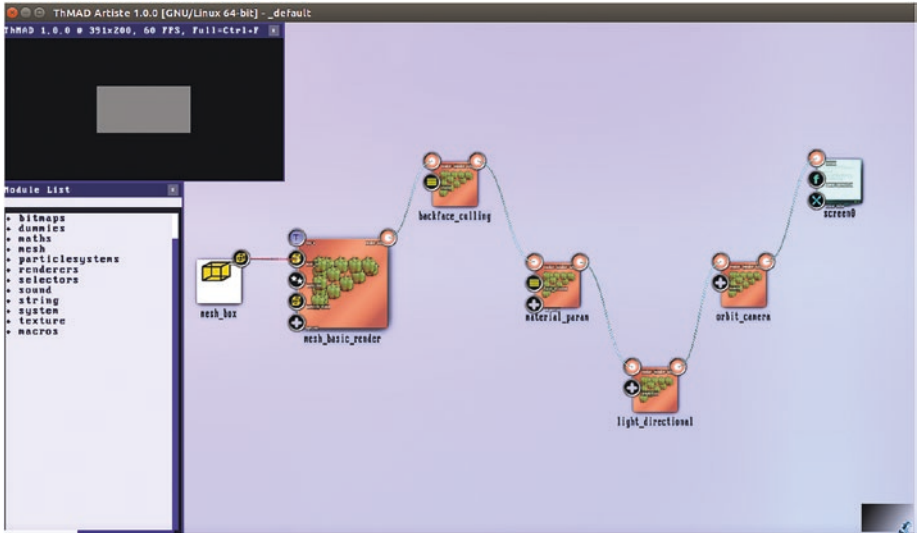


Figure 2-19. First 3D state

To help with that, you first change the color of the light. Again inside the Properties complex anchor of `light_directional`, right-click on the `diffuse_color` anchor and choose the Color control from the pop-up menu. From the control—see Figure 2-20—choose any color you like. Click and drag in the top left area to choose the Hue, and then click in the big quadratic area to choose the Saturation and Value. You can drag there as well to see the immediate changes in the preview window. Double-click on the top-right area to quit the control.

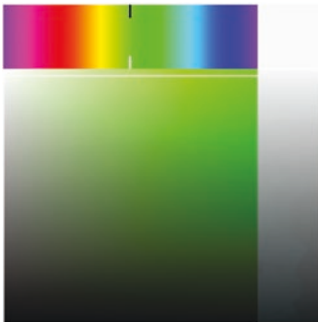


Figure 2-20. Color chooser control

You might have noticed that it does not properly look like a face of a cube, instead rather like one face of a rectangular prism. That comes from your monitor most likely having an aspect ratio other than 1:1. There is an easy way out of it: open the Camera complex anchor of the `orbit_camera` module and switch the Perspective Correct anchor to YES.

It still does not look very 3Dish. To unleash the power of the third dimension and show it as a cube, you rotate the camera position and look at it from a different angle onto the scene. To do so, open the complex anchor named “Camera of the module orbit_camera. Inside you’ll find the anchor called Rotation. Right-click on it and choose Axes View. A colored cross will show up; see Figure 2-21.

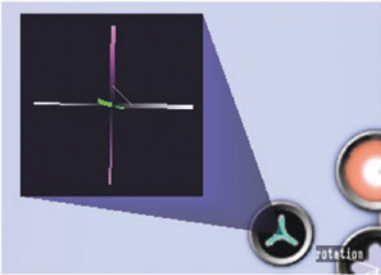


Figure 2-21. Axes view anchor control

Click and drag on it near its center and you’ll see the camera rotate about the center of the scene, which happens to be the center of the cube as well. When you are satisfied, you can close the control by double-clicking on it. Your result might look like Figure 2-22.

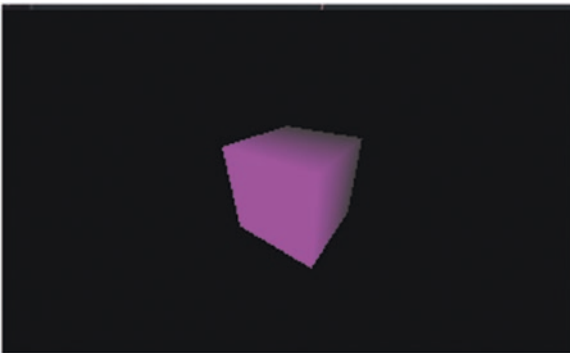


Figure 2-22. Elaborated 3D scene

As a final step, you’ll add responsiveness to sound. Place the sound → input_visualization_listener somewhere near the screen0 module. Do the same with the module maths → converters → 4float_to_float4, which will help us to control the color.

Connect the output result_float4 anchor of 4float_to_float4 to the input anchor clear_color of screen0. Also connect one of the vu_* subanchors of vu of the input_visualization_listener to the third input anchor of 4float_to_f. See Figure 2-23.

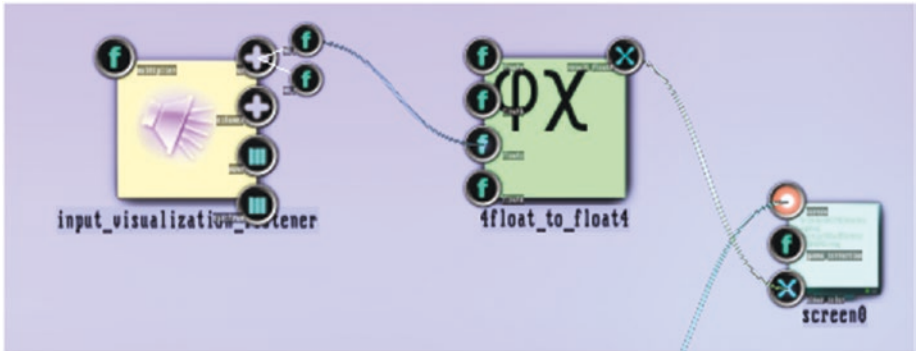


Figure 2-23. Sound input for the 3D sketch

The Multiplier input anchor of `input_visualization_listener` controls the sensibility to the sound. To give a value to it other than its default of 1.0, double-click on it and click and drag on the knob that appears. Then give it a new value, such as 0.3. Close the knob by double-clicking on it.

The four input anchors of `4float_to_float4` demultiplex the screen clearing color, so they represent the RED, GREEN, BLUE, and ALPHA channels of the background color. Set the RED and GREEN anchors to 0.0, again by clicking on them and using the knobs to change their values. Leave the ALPHA set to value to 1.0.

Start the sound input, and you should see the background flickering in blue to the input of the sound. If you don't see that, your sound volume may be too low and you should assign a higher value to the input multiplier of `input_visualization_listener`.

Of course you could do the same for any color value in use, say a light color or a material color. Go ahead and play with those at will. You also can draw several connections from the `vu/vu_*` anchor of `input_visualization_listener` to different input anchors of other modules. This is a powerful feature throughout ThMAD, applicable to a lot of anchors of many modules. The other way around—connecting several output anchors of different modules to one input anchor of a certain module—applies as well depending on the module and anchor type. For example, several renderers of several pipelines acting in parallel can connect to the sole `screen0` input.

Go to an empty spot of the canvas and press `Ctrl+S` to save it as the `_default state`, or right-click on an empty spot of the canvas and choose `Save As...` to give it a name. Remember, the `_default state` is opened automatically when you start ThMAD Artiste the next time. But to keep it for future reference, use `Save As..` and give it a lasting name.

For the rest of the book, we will often use a tabular listing of anchor values to make things as clear as possible. Note that default anchor values are explained only in cases where this will noticeably help in understanding states. Also, anchors connected to other modules will usually be described in the text body, only sometimes in the table.

The tabular presentation of anchor values for the current state is as shown in Tables 2-1, 2-2, 2-3, 2-4 and 2-5.

Table 2-1.

| renderers → opengl_modifiers → backface_culling | | |
|--|---------|-------------------|
| status | ENABLED | Enable the module |

Table 2-2.

| renderers → opengl_modifiers → light_directional | | |
|---|------------------|---|
| properties /light_id | 0 | You only have to change this if you have more than one light. |
| properties /enabled | YES | Enable the module. |
| properties /position | 0; 0; 1 | This means the light is shining from above the x-y plane. |
| properties /ambient_color | 0; 0; 0; 1 | Non-directional light black = OFF. |
| properties /diffuse_color | 0.9; 0.6; 0.8; 1 | Directional light with diffuse reflectance. Choose at will. |
| properties /specular_color | 0; 0; 0; 1 | Directional light with exact reflectance. Does not make too much sense with only plane faces. |

Table 2-3.

| renderers → opengl_modifiers → cameras → orbit_camera | | |
|--|------------------|--|
| camera / perspective_correct | yes | Fix aspect ratio according to your screen. |
| camera / rotation | 0.71; 0.46; 0.53 | For the orbit camera, the position of the camera. It will be looking at the destination. |
| camera / distance | 2.0 | The distance of the camera to the destination. |
| camera / destination | 0; 0; 0 | Where the camera will be looking, or the center of the 3D state. |

Table 2-4.

| maths → converters → 4float_to_float4 | | |
|---|-----|---------------------------------|
| <i>The floatc anchor is connected to some other module.</i> | | |
| floata | 0.0 | Will map to a RED color value |
| floatb | 0.0 | Will map to a GREEN color value |
| floatd | 1.0 | Will map to an ALPHA value |

Table 2-5.

sound → **input_visualization_listener**

| | | |
|------------|-----|-----------------------------|
| multiplier | 0.3 | Depends on the sound level. |
|------------|-----|-----------------------------|

Summary

In this chapter, we learned which tools are needed to run and use ThMAD, and how it is installed and configured. We also saw how to create a DVD from a visualization, and we introduced the first two basic visualization examples using ThMAD.

In the next chapter, we will look more thoroughly at the operational aspects of ThMAD, including how its parts are started and which options can be used to achieve specific needs.

CHAPTER 3



Program Operation

The ThMAD suite primarily consists of two programs—ThMAD *Artiste* and ThMAD *Player*. We describe how these programs can be started and stopped, including all possible options for controlling program operation. We furthermore describe faders, which conduct the transitions between several visualizations inside *Player*.

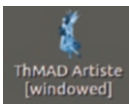
ThMAD Artiste Operation

ThMAD *Artiste* is the program for creating visualization sketches, called *states*, and viewing them in a preview mode.

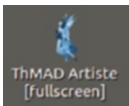
Starting and Using Different Modes

If you followed the installation instructions in Chapter 2, “Visualization Basics,” you will find two launchers on your desktop:

They are actually just links to the following.



Start Artiste in windowed mode



Start Artiste in fullscreen mode

Windowed mode:

```
/opt/thmad/thmad_artiste
```

Fullscreen mode:

```
/opt/thmad/thmad_artiste -f
```

which are going to be described in more detail in this chapter.

Starting from a terminal—you can open one via `Ctrl+Alt+T`—the synopsis is shown in Table 3-1.

Table 3-1. *ThMAD Artiste Options from the Terminal*

/opt/thmad/thmad_artiste [option1 option2 ...]

Options

| | |
|-----------------------|--|
| <none> | Starts Artiste in windowed mode. Shows the canvas for creating sketches, a small preview window, and a module list. |
| -help | Shows help and immediately quits the program. |
| -h | Same as -help. |
| -sm | Prints all detected monitors and immediately quits the program. You can use the output to specify a monitor number for the -m option. |
| -m mon | Uses monitor number mon for the fullscreen mode. Has no effect if not used with the -f option. |
| -f | Starts Artiste in fullscreen mode. It is not possible to switch to the fullscreen mode from inside the program. You can exit this mode by pressing the Escape button. Can be used in conjunction with the -ff and -fn options. |
| -ff | Starts Artiste in fullwindow mode. The graphics output will use the complete window space. You can later switch back to the non-fullwindow standard mode by pressing Ctrl+F. |
| -fn | If in fullwindow mode, suppresses the info text in the header area. |
| -s 1024x860 | Sets the window size in windowed mode. 1024x860 is just an example; see output of the -sm option to list possible values. |
| -p 200x100 | Sets the window position; 200x100 is just an example. |
| -novsync | Experimental. Disables using double buffering. |
| -gl_debug | Experimental. Activates the special OpenGL debugging feature. |
| -port 3267 | Starts a TCP/IP port where commands to control ThMAD from outside may be sent to. 3267 is just an example. The details of the protocol are not part of this book. |
| -sound_type_alsa | Directly use the ALSA API instead of PulseAudio. |
| -sound_type_jack | Use a JACK sound server endpoint to connect. |
| -snd_rtaudio_device=5 | If using ALSA or JACK, specify the sound device to use. Sound devices are listed upon startup, but the audio_visualization_listener module must be present. |

If you used the Artiste desktop launcher or entered `/opt/thmad/thmad_artiste` inside a terminal to start Artiste, it will show up in a window, as shown in Figure 3-1. Within this windowed mode, in the top-left corner you will find a small output sub-window where the graphics will be painted. You can move this preview window around on the canvas by clicking and dragging anywhere from inside it and you can change its size by clicking and dragging from its borders or corners.

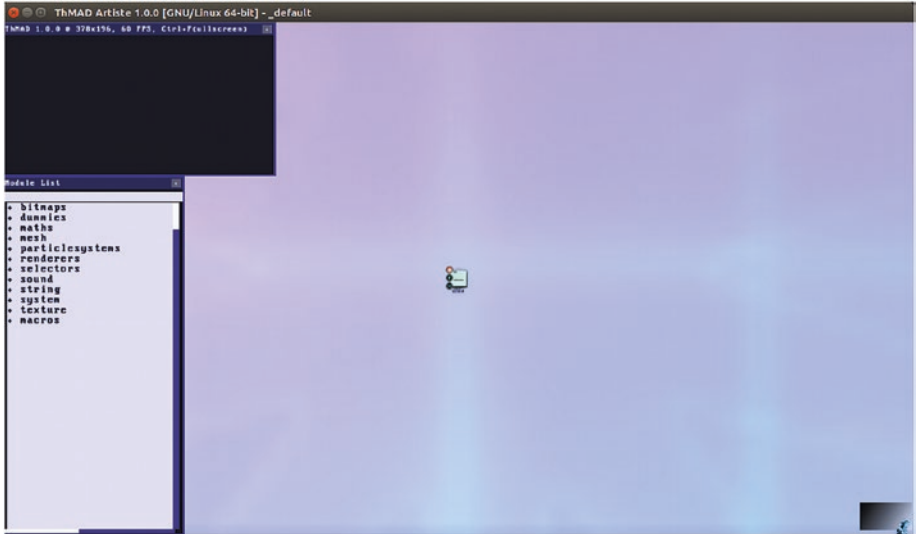


Figure 3-1. ThMAD Artiste in windowed mode

For details about how to use the GUI, have a look at Chapter 7, which contains a ThMAD GUI reference guide.

In Ubuntu, you can take a snapshot image of the window or parts of it by pressing the Print key with or without the Shift key held down. From inside the window, you can change to fullwindow mode by pressing the key combination Ctrl+F. The GUI elements then vanish and the output will use the complete window space. See Figure 3-2.



Figure 3-2. *ThMAD Artiste in fullwindow mode*

By default, fullwindow mode shows some status information in the header area. To disable or enable this status display, you can press Alt+T once or twice.

You can also start Artiste using the `-fn` option to disable the status information from the beginning. To return from the fullwindow mode to the windowed mode, press Ctrl+F again.

There is also a performance mode, which presents an overlay of the *state* creation canvas and the graphics output, as shown in Figure 3-3.

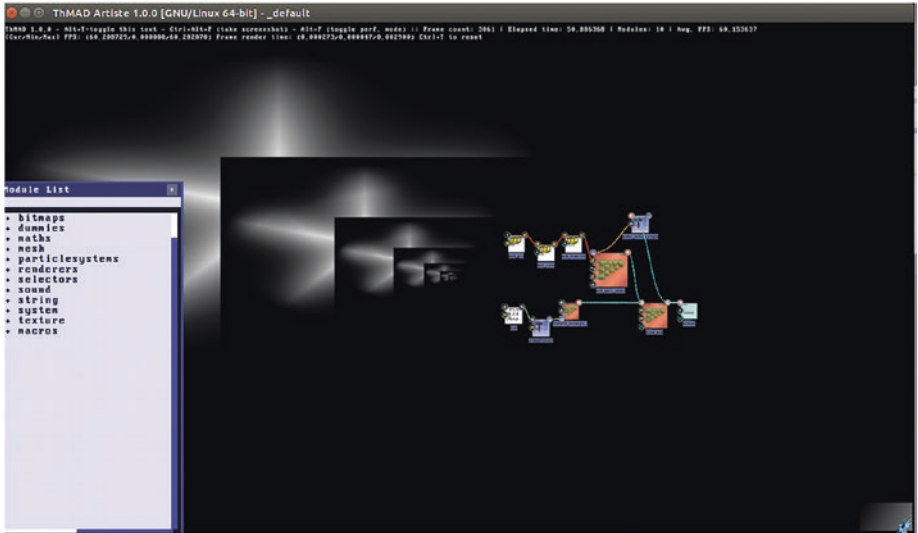


Figure 3-3. *ThMAD Artiste in performance mode*

To enable performance mode, start in fullwindow mode or switch to fullwindow mode and then press `Alt+F`. The performance mode is fully functional speaking of construction. The output of the sketch will in real-time be drawn all over the background while you are editing the rendering pipeline. You exit performance mode by pressing `Alt+F` again. Also inside the performance mode you can toggle the visibility of the header info lines by pressing `Alt+T` once or twice.

Artiste can also be started in fullscreen mode; to do so, use the accordingly named launcher on the desktop or start Artiste from a terminal via this command:

```
/opt/thmad/thmad_artiste -f
```

Include any other options you want to add. In fullscreen mode, all window shortcuts are available.

To leave the program in any mode, press the `Escape` key, or, if available, click on the close button of the window, or use the main pop-up window after you right-click on an empty spot of the canvas.

There is currently no way to let ThMAD render graphics data directly to a video file, but you can use other programs for that aim; see [Chapter 2](#) for more information.

Stopping ThMAD Artiste

ThMAD Artiste can be stopped via these methods:

- From the main pop-up menu you get by right-clicking an empty spot of the canvas, click Exit. ThMAD detects if you have saved changes and if this is the case will ask you whether you really want to exit.
- By pressing the Escape key. Also here, ThMAD will tell you if there are unsaved changes.
- There is a module inside the rendering pipeline called `system` → `shutdown` and you can place it on the canvas and connect it to the screen module `screen0`. As soon as the module's input raises above 1.0, the system will be shut down.

■ **Caution** The shutdown module shows no mercy—all unsaved data will be lost. If, by whatever means, you manage its input to receive a number greater than 1.0 upon the next state startup, you will never be able to open that state again unless you change the state file manually by using a text editor. This is easy however—just remove the line `component_create system;shutdown ...` from the state file.

Note that, referring to the shutdown module, a common feature of ThMAD is that a module not connected to a sub-pipeline eventually reaching the `screen0` module will never run. That is why it has to be connected to, for example, the `screen0` itself to function correctly, even though it does not produce graphical data.

Starting with Errors

If an error message appears telling you that a module could not be loaded, the reason for that might be that your state is from pre-ThMAD times. While considerable effort has been spent on backward-compatibility toward the predecessor of ThMAD, certain old modules may have leaked through, and Artiste will show an empty state in these cases.

There is still hope that you can repair it using system programs. Open the state with a text editor from the following file

```
/home/[USER]/thmad/[VERSION]/data/states
```

and remove the corresponding line starting with

```
component_create [NAME]
```

After that, starting the Artiste GUI will yield all the rest of the *state* to appear. If the module was important, try to find a substitute from the module list or the module browser.

ThMAD Player Operation

Once you're finished with your Artiste state, you can use Artist's export functionality to convert a *state* to a *visual*. This happens when you choose the Compile → Music Visual command from the Artist's main context pop-up menu after you right-click on an empty spot of the canvas.

Starting and Using Different Modes

In default operation mode, the Player will recursively register all visuals it finds inside the user data folder and play them one by one. If you are used to ThMAD's predecessor VSXu, where the Player by default looks in the installation folder, you have to know that difference.

Also, contrary to Artiste's operation, the Player knows about *faders*, which introduce a transition between visuals when it comes to switching from one to another. They are described later in this chapter; here we just mention that Players get created via Artiste and exported via the Compile → Music Visual Fader menu command. They then end up inside the *faders* folder.

The Player will not see your exports automatically, since data spaces for Player and Artiste are kept separate. In order to make exported states available to ThMAD Player, you either have to copy the visuals and faders from these folders:

```
/home/[USER]/thmad/[VERSION]/data/visuals
```

```
/home/[USER]/thmad/[VERSION]/data/faders
```

to these folders:

```
/home/[USER]/thmad/[VERSION]/data/player_visuals
```

```
/home/[USER]/thmad/[VERSION]/data/player_faders
```

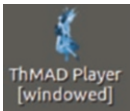
Or you have to copy them to some other place and tell the Player via the startup option where to find them; the `-path` flag is described shortly.

By default only a handful of visuals and faders are made available to the Player upon installation. You can find a lot more visuals and some more faders inside these folders:

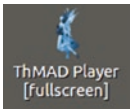
- `/opt/thmad/share/thmad/player_visuals`
- `/opt/thmad/share/thmad/example_visuals`
- `/opt/thmad/share/thmad/player_faders`
- `/opt/thmad/share/thmad/example_faders`

You can copy them into the `player_visuals` and `player_faders` folders. Note that some of those show noticeable output only when there is actually sound input.

If you followed the installation instructions in Chapter 2, you will find two starter icons on your desktop:



Start Player in windowed mode



Start Player in fullscreen mode

They are the links to the following commands.

Windowed mode:

```
/opt/thmad/thmad_player
```

Fullscreen mode:

```
/opt/thmad/thmad_player -f
```

You could also just invoke them from a terminal. If you choose to do so, the full list of options, including the `-f` flag, can be seen in Table 3-2.

Table 3-2. *ThMAD Player Options from the Terminal***`/opt/thmad/thmad_player [option1 option2 ...]`****Options**

| | |
|---------------------------|---|
| <code><none></code> | <code>/home/[USER]/thmad/ [VERSION]/data/ player_visuals</code> Uses all faders found inside <code>/home/[USER]/thmad/ [VERSION]/data/ player_faders</code> . Visuals and faders are played in random order, each running for 30 seconds. Note that <code>/home/[USER]/thmad</code> is a symbolic link to <code>/home/[USER]/.local/ share/thmad</code> . |
| <code>-help</code> | Shows help and immediately quits the program. |
| <code>-h</code> | Same as <code>-help</code> . |
| <code>-path PATH</code> | Does not load the visuals from the local user data path. See the <code><none></code> options. Instead, it loads all visuals from the path <code>PATH/player_visuals</code> and uses all faders found inside <code>PATH/player_faders</code> . |
| <code>-dr</code> | Disables the randomizer. The Player will then not automatically cycle through the available visuals. Still the visual that's chosen will be a random one. |
| <code>-rb 20</code> | If the randomizer is <i>not</i> disabled, at least 20 secs (or you can choose any other number) will transpire before changing to the next visual. If this option is not provided, the value defaults to 30 seconds. |
| <code>-rr 10</code> | Randomizes the randomizer, if not disabled. Visual runtime duration will be chosen randomly between the base number from the <code>-rb</code> option and the <code>-rb</code> number plus the <code>-rr</code> value. In this example, it's between 20 and 30 seconds. If this option is not provided, the value defaults to 0 seconds. |
| <code>-f</code> | Starts in fullscreen mode. |
| <code>-sm</code> | Lists available monitors and monitor modes. |
| <code>-m 2</code> | If in fullscreen mode, uses monitor number 2 (choose at will). |
| <code>-fm</code> | Lists available video modes for fullscreen mode. Depends on the monitor chosen (see the <code>-m</code> option). |
| <code>-p 300x200</code> | If in windowed mode, sets the window position to (300;200). Choose at will. |
| <code>-s 640x480</code> | If in windowed mode, specifies the window size. 640x480 is only an example; choose any size you like. |

(continued)

Table 3-2. (continued)

| /opt/thmad/thmad_player [option1 option2 ...] | |
|--|---|
| | If in fullscreen mode, this may be used to request the resolution. ThMAD then tries to find the best possible match. See the <code>-fm</code> option to see a list of the available video modes. If this is not given and the fullscreen mode and possibly some monitors are requested, the video mode will automatically be chosen based on your current settings. Letting the system choose is the preferable way. ¹ |
| <code>-no</code> | No splash screen and overlay. Means that it will start immediately with the first visual and it will not print a visual's name at its beginning. |
| <code>-lv</code> | List visuals seen by the Player. Depends on the <code>-path</code> option if chosen. |
| <code>-lf</code> | List faders seen by the Player. Depends on the <code>-path</code> option if chosen. |
| <code>-port 3267</code> | Starts a TCP/IP port to which the commands to control ThMAD from outside are sent. 3267 is just an example. The details of the protocol are not part of this book. |
| <code>-sound_type_alsa</code> | Directly use the ALSA API instead of PulseAudio. |
| <code>-sound_type_jack</code> | Use a JACK sound server endpoint to connect. |
| <code>-snd_rtaudio_device=5</code> | If using ALSA or JACK, specify the sound device to use. Sound devices get listed upon startup, but the <code>audio_visualization_listener</code> module must be present. |

¹ If you request a certain resolution in fullscreen mode, it may cause ThMAD program termination and show your desktop in that new resolution. You may have to manually revert the resolution setting or restart your desktop if you want to switch back to the resolution you are accustomed to.

Unlike ThMAD Artiste, in the Player, the visual will immediately cover the whole window or screen, and there is nothing like a context menu for the Player. You can, however, press F1 to get some basic on-screen help.

Stopping ThMAD Player

ThMAD can be stopped using one of these methods:

- By pressing the Escape key with the focus on the ThMAD Player window. In fullscreen mode, no focus is needed.
- If while constructing the state, you placed the Module System → Shutdown on the canvas and connected it to `screen0`, as soon as the module's input raises above 1.0, the system will be shut down.

Creating and Installing Faders

Faders are programs that create smooth transitions whenever the Player switches from one visual to the next. They are created using Artiste and from there are exported as special *fader visuals* and then ready for use by the Player.

The detailed procedure for constructing and then installing your own faders is as follows:

1. Start ThMAD Artiste with a clean canvas containing only the `screen0` module.
2. From the main context menu, right-click an empty spot of the canvas and then choose the New → Transition for ThMAD Player menu entry.
3. A state is presented and started with the Module System → `visual_fader` as its main module; see Figure 3-4. This one is responsible for controlling the fading. Consider its two texture type inputs as blind inputs—they will be served the two transitioning states while the fader is running inside the Player. Anything you might want to connect there will be silently ignored. It is the two outputs which are interesting—they present both transitioning states as textures for the rendering process while the fader is running, eventually, as usual in ThMAD, ending up in the `screen0` module.

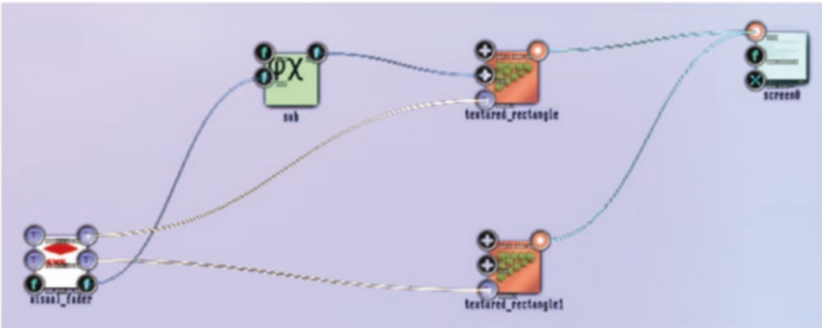


Figure 3-4. A basic fader transition state

The other output anchor, called `fade_pos_out`, gives you the state of the transition. It will read 0.0 when the transition starts and 1.0 when the transition ends. Note that only while constructing the fader in Artiste does the value seem to be oscillating. This is for illustration purposes only—while it is running inside the Player, the transition from 0.0 to 1.0 happens only between two visuals.

4. The `transition_length` input anchor specifies the number of seconds used for the transition.
5. You must save the *state* the first time, if you want to work on it later. Closing Artiste without saving will lead to data loss in this case. The next time you load it, Artiste will handle it like any other state, including the check for possible data loss when closing it.
6. If you want to install the fader for use by the Player, export it from the main context pop-up. Remember you have to right-click an empty spot of the canvas and then choose `Compile` → `Music Visual Fader`. You will then find the fader inside the `/home/[USER]/thmad/[VERSION]/data/faders` folder.

Make the new visual fader accessible to the Player. Either copy it to `/opt/thmad/share/thmad/player_faders` if you want Player to use its default data path, or to this path:

```
/home/[USER]/thmad/  
[VERSION]/data/player_faders
```

For the `-path` option at work, see the section entitled “Starting and Using Different Modes,” earlier this chapter.

Summary

In this chapter, you learned how to invoke ThMAD Artiste and Player, and what options you have when you use the starters from inside a terminal. You saw that Artiste can run inside a window or cover the whole screen. It can also mix input and output on the same screen.

In the next chapter, you will learn about 3D concepts used in ThMAD: coordinate systems, space mapping, lighting, and 3D objects as seen by ThMAD.

CHAPTER 4



3D Concepts

If you want an exhaustive reference to 3D computer graphics concepts, lots of materials are available on the Internet and in bookstores. OpenGL, the technology behind ThMAD, is a huge concept and complete coverage of all its topics exceed the scope of this book. Even so, an introduction with our special view on audio visualization should be worthwhile. In addition some of the peculiarities about how ThMAD handles some graphics sub-pipelines and its own graphics concepts on top of OpenGL are described later in this chapter.

Coordinate Systems

Transformations in three dimensions are about spatial operations, like shifting, scaling, and rotation. But before we start talking about spatial operations, we need to know how 3D objects are represented in a way that a computer program can understand, and this immediately leads to thinking about coordinate systems.

To make things easy, we start with two dimensions. If you look at your computer screen, we *define* the plane you see as the x-y-plane, with the x coordinate of a point on that plane denoting the horizontal distance to a vertical line at the left border, and the y coordinate denoting the vertical distance to a horizontal line at the bottom border. See Figure 4-1.

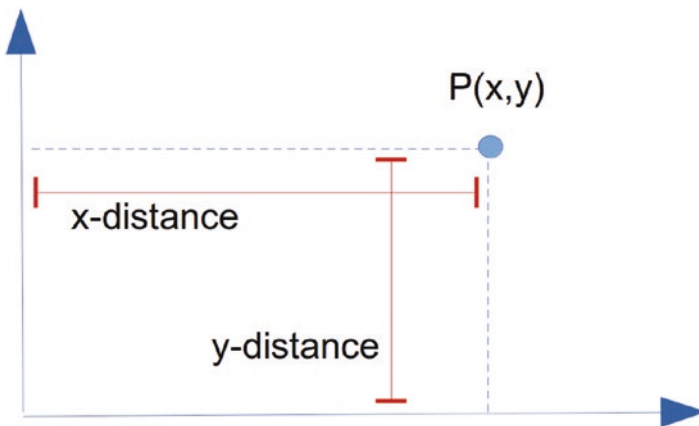


Figure 4-1. X-Y coordinate system

The vertical line as the reference to the x-distance is also called the y-axis, and the horizontal line as a reference to the y- distance is called the x-axis. We said that we use the borders of the screen for the axes, but in fact this is an arbitrary decision; we could have used any line parallel to them. Another common choice that's also quite often used in ThMAD is to use axes at the center of the screen, as shown in Figure 4-3. The crossing point of the axes is called the origin O, and it has the coordinates O(0,0,0), since the distances are zero for both coordinates.

What numbers will have been assigned to the coordinates of point P in Figure 4-3? At the center it is easy; $x=0$ and $y=0$. We need another definition for points away from O, and we could choose x to be 1.0 at the right edge, and y to be 1.0 at the top edge. As a consequence of placing O in the middle of the screen, we then automatically have $x=-1.0$ at the left border and $y=-1.0$ at the bottom border. See Figure 4-2.

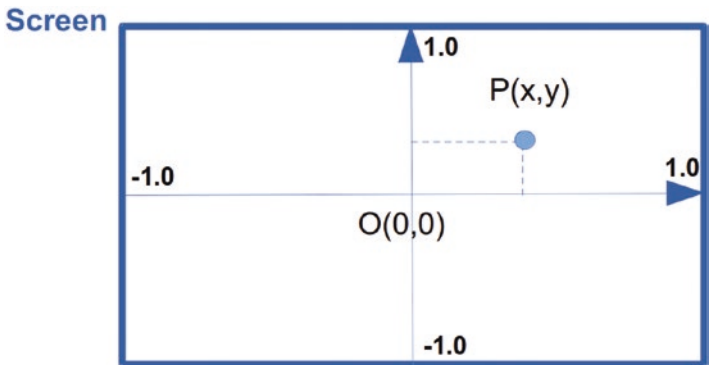


Figure 4-2. X-Y coordinate system, scaled to -1 to +1 for both dimensions
P(x,y) lies at approximately $x = 0.4$ and $y = 0.3$, hence we can now write $P(0.4, 0.3)$.

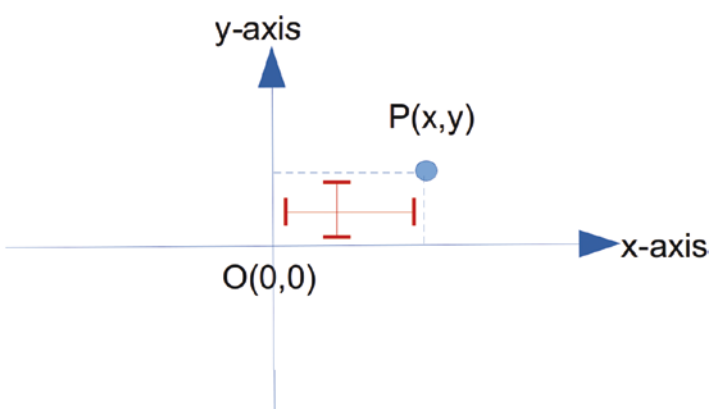


Figure 4-3. X-Y coordinate system, centered at the middle of the screen

We are now equipped with all we need to talk about points in two dimensions. But what about the third dimension and the z-axis? It must be perpendicular to the x-axis and y-axis, so it either points from front-right onto the screen, or from the back.

OpenGL is a so-called right-handed system, meaning that if you let your right-hand index finger follow the order x-axis to y-axis, then your thumb will point to the positive z-direction. Sitting in front of the screen, look at the x-y plane from a positive z- position. See Figure 4-4.

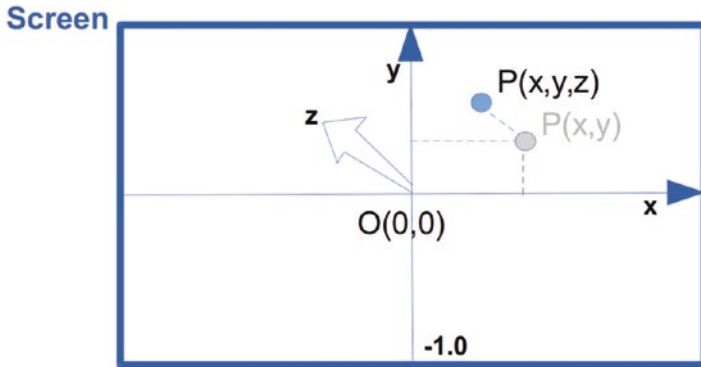


Figure 4-4. X-Y-Z coordinate system

The scaling of the z-axis is not that obvious—there is no z-border on a two-dimensional screen. To get things right, OpenGL does this:

1. Fixes the x-y aspect ratio. This scales the x-axis such that a square as input: $(-a,-a) \rightarrow (a,-a) \rightarrow (a,a) \rightarrow (-a,a)$ will still appear as a real square on the screen, with all sides having the same length. This obviously changes the border coordinate values. With the bottom and top border still at -1 and +1, respectively, say for a 5/4 screen, the left and right borders are now at -1.25 and +1.25.
2. Scales the z-axis in such a way that a cube as input after any kind of rotation in space will still appear as a cube, with all edges having the same length on the screen.

Having said that, an obvious choice for positioning objects is the cube for the front face

$$(-1,-1,1) \rightarrow (1,-1,1) \rightarrow (1,1,1) \rightarrow (-1,1,1)$$

and this for the back face

$$(-1,-1,-1) \rightarrow (1,-1,-1) \rightarrow (1,1,-1) \rightarrow (-1,1,-1)$$

Of course, you can position objects as you wish and use any coordinates you like, but somehow having that “double unit cube” in mind helps avoid surprises, like objects unintentionally and miraculously disappearing out of sight. If you have other bounding boxes for your 3D scene, that is boxes with the smallest possible dimension but still containing all the objects in question, you can use translation and scaling to shift and squeeze everything into the unit space, do anything you like there, and afterward scale and shift it back. See the sections entitled “Space Mapping” and “Spatial Operations: Translation, Rotation, and Scaling” in this chapter.

Space Mapping

Coordinate spaces of any kind, whether they are in two or three dimensions, can be mapped into another space of the same kind in such a way that the inter- and intra-relations of points comprising them are remained one or the other way.

The most important transformation of such a kind is called the *linear transformation*, and it is represented by this calculation rule:

$$\begin{aligned}x' &= a_{11} \cdot x + a_{12} \cdot y + a_{13} \cdot z + a_{14} \\y' &= a_{21} \cdot x + a_{22} \cdot y + a_{23} \cdot z + a_{24} \\z' &= a_{31} \cdot x + a_{32} \cdot y + a_{33} \cdot z + a_{34}\end{aligned}$$

by which any point $P(x,y,z)$ gets mapped to a point $P'(x',y',z')$. Using so-called *homogeneous* coordinates, mathematicians and we, if we want to be concise, extend this concept by a translational part. We eventually use the following matrix notation:

$$\begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

where we replaced a_{14} , a_{24} , and a_{34} with t_1 , t_2 , and t_3 to express their constant translational nature. The figure in the middle, for brevity we call it (a_{ij}, t_i) , is called the *transformation matrix*; the others are called coordinate *vectors*. The dot represents a matrix multiplication. The formalism for the multiplication is as follows:

$$x' = x'_i = a_{ij} \cdot x_j = \sum_j (a_{ij} \cdot x_j) = \mathbf{a} \cdot \mathbf{x}$$

where $x_1 = x$, $x_2 = y$, and $x_3 = z$. The symbol \sum_j follows the usual mathematical rule “sum over all j ”. The nice thing about transformations represented by matrices is that combined transformations—e.g., first scale, then rotate, then shift—can be represented by multiplied matrices:

$$\begin{aligned}\text{Combined}(\mathbf{a}, \mathbf{b}, \mathbf{c}) &= \mathbf{a} \cdot \mathbf{b} \cdot \mathbf{c} = a_{ij} \cdot b_{jk} \cdot c_{kl} \\ &= \sum_j \left(a_{ij} \cdot \sum_k (b_{jk} \cdot c_{kl}) \right)\end{aligned}$$

OpenGL (in the version 3.x used by Ubuntu while writing this book) internally uses several distinct matrix stacks. More current versions of OpenGL somewhat lowered their significance or even removed them in favor of shader language constructs, which are also available in ThMAD. We do not present a shader language introduction in this book; however, you can find some shader using examples in the stories in Chapters 5 and 6.

Within ThMAD you usually do not have direct contact with matrices. For all the typical matrix operations, ThMAD has modules—translation, scaling, and rotation. You can stack them graphically by connecting as many of them as you like. Internally ThMAD of course combines all those operations in the aforementioned matrix multiplication way. It's important for you to know about the concept of transformation combination operations, or stacked transformations because many states use such combined transformations.

From the ThMAD point of view, which is not only practical but also necessary in some use cases, are transformation combinations of the form:

$$\mathbf{t}^{-1} \cdot \mathbf{a} \cdot \mathbf{t}$$

This expresses: (1) First transform according to the matrix t , (2) then transform according to a , (3) then transform according to the inverse of t , as designated by t^{-1} . Looks like theoretical mathematics, does it not? Well, it is practically important. Consider an example.

Consider a box of size 0.1 x 0.1 x 0.1 centered at (0.5; 0.5; 0.5). You want to rotate it around its center by some rotation.

transformation \mathbf{a} . It sounds strange, but there is no module of ThMAD capable of doing that! But there is a module for a rotation around (0;0;0). We cannot use that rotation in this case, because in the end, the center of the box was moved away, and we wanted it to stay at (0.5; 0.5; 0.5). What we can do now to perform the operation is:

1. First shift the box by a translation vector (-0.5; -0.5; -0.5) so that its center is at (0;0;0).
2. Now perform the rotation around (0;0;0).
3. Then shift it back via the translation vector (0.5; 0.5; 0.5).

This exactly is what $\mathbf{t}^{-1} \cdot \mathbf{a} \cdot \mathbf{t}$ expresses. Apart from the mathematical notation or actually calculating coordinates manually, you should get used to the idea of embraced transformations in a graphical sense; see Figure 4-5.

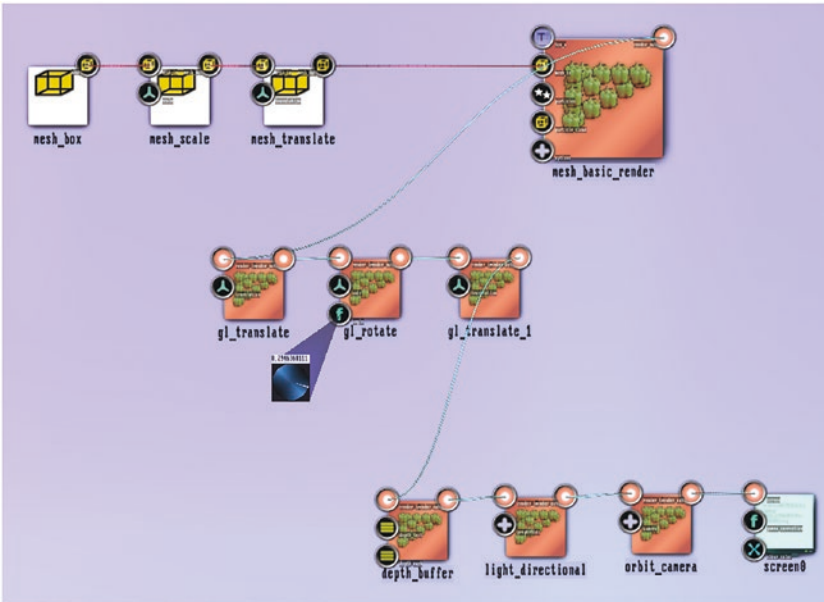


Figure 4-5. Embracing transformations

The top sub-pipeline produces a box away from the origin and paints it. The bottom sub-pipeline just removes hidden surfaces, places a light and a camera, and sends it to the screen. The interesting part is the middle sub-pipeline; it shifts the box back to the origin, then rotates it around the origin, and shifts it back where it was before. The “translation” anchor of the left module reads (-0.5; -0.5; -0.5) and the same anchor for the right module is (0.5; 0.5; 0.5). The “angle” anchor, opened here, can now be used to rotate the box around its center.

■ **Note** This sample is available as a source under A-4.2_Embracing_Transformation inside the TheArtOfAudioVisualization folder.

For those who want to know what t looks like for the following sample (see the section on how to construct translation matrices):

$$t = \begin{pmatrix} 1 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & -0.5 \\ 0 & 0 & 1 & -0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$t^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

By the way, t^{-1} is called the *inverse* of t because the product gives the identity matrix:

$$\begin{aligned} t \cdot t^{-1} &= \begin{pmatrix} 1 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & -0.5 \\ 0 & 0 & 1 & -0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

The latter is called identity matrix because $a \cdot I = I \cdot a = a$ for any matrix or vector a .

Spatial Operations: Translation, Rotation, and Scaling

Now that you're equipped with the notation from the previous section, you can express translation and scaling easily.

Using the matrix presentation we introduced, we get the following for the translation along vector (a,b,c) :

$$t = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

And for the scaling with scaling factors (s_1, s_2, s_3) :

$$t = \begin{pmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where for uniform scaling with a single scaling factor s the components s_1 , s_2 , and s_3 are all the same: $s_1 = s_2 = s_3 = s$.

Rotation is a little bit more complicated—we have to define how to rotate. For example, we can rotate around an axis (u,v,w) from the origin, at some angle φ . The formula in this case reads as follows:

$$\begin{aligned}
 U &= 1 - u^2 \\
 V &= 1 - v^2 & C &= \cos(\varphi), D = 1 - \cos(\varphi) \\
 W &= 1 - w^2 & S &= \sin(\varphi)
 \end{aligned}$$

$$t = \begin{pmatrix} u^2 + U \cdot C & uv \cdot D - w \cdot S & uw \cdot D + v \cdot S & 0 \\ uv \cdot D + w \cdot S & v^2 + V \cdot C & vw \cdot D - u \cdot S & 0 \\ uw \cdot D - v \cdot S & vw \cdot D + u \cdot S & w^2 + W \cdot C & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Another possibility is to first rotate about the x-axis by some angle, then about the y-axis by some angle, then about the z-axis by some angle. The transformation matrices for that can easily be deduced from this formula by respectively setting $(u,v,w) = (1,0,0) / (0,1,0) / (0,0,1)$. ThMAD primarily uses the combined formula to make things concise.

A third possibility to describe rotations consists of using *quaternions*. While the theory behind quaternions is not that easy to understand, it might be enough for the purposes here to say that a quaternion is a number consisting of four parts:

$$q = (q_1, q_2, q_3, w)$$

It is enough to describe a rotation in space, provided a bunch of special quaternion calculation rules are given. That is just another way of saying that, for quaternions, we do not need matrix operations and instead can use that new calculation ruleset, which was introduced especially for them.

In order to learn how to perform transformations like these in ThMAD, have a look at Chapter 5.

Exposure to Light

In the real world you can see things because they are reflecting light that shines on them. So we need light sources to make objects appear. In OpenGL and ThMAD, it is the same, with two exceptions. The first one is that some simple states do not need a light. However, in many cases it does not hurt to add one, and it's good practice to have a light. The second exception is that in OpenGL and ThMAD, you actually never see a light directly, you only see its reflections.

Strictly speaking, this is not a lot different from the real world—if you look at a lamp, you see the light coming from there mostly because it is reflected from the parts of the lamp. And only if you had something like a point light source could you see the light without reflecting objects. This corner case is not embarrassing, and because of that and for simplicity, directly visible light sources are not available in ThMAD or OpenGL. Note that there still is a concept of a surface material emitting light by itself, but this is conceptually connected to material properties, not light sources. Details follow shortly.

In OpenGL and ThMAD, we thus have two kinds of objects that are involved in lighting: lights or light sources and materials.

Lights as seen from the light source perspective come in four flavors, to allow for realistic scenes:

- *Ambient light.* Think of this as non-directional light, which is everywhere. If you are in a dull room with only a little light sneaking through the gaps in the curtains, you still see some kind of dim light which seems to light everything evenly in the room and seems to come from everywhere and nowhere. Only experience tells us that the light originates from the lighted gaps. This is what ambient light is about, and the way ThMAD and OpenGL handle it is to add some constant color value to all surfaces. The truth is a little bit more complicated, however, since material properties are involved into the calculation of ambient color coming from surfaces
- *Diffuse light.* Consider a light source at some point. If the light from there hits a material surface, and from there gets scattered evenly into all possible directions, this is diffuse light. This kind of diffuse reflectance is most commonly related to the material nature of surfaces. Unless you need a surface that perfectly mirrors 100% of any light shining on it, you will have a diffuse light component in your scene. It is therefore found in almost all 3D scenes.
- *Specular light.* If a light beam hits a material surface and is reflected by the same angle away from the surface, we speak of specular light. You want to have a specular light component to make your scenes shinier and more interesting.
- *The screen's clear color.* This will be applied to the whole scene each frame before anything else is rendered. It will thus show up as a background color and may shimmer through objects only if they are fully or in part transparent.

For the different light modes, see Figure 4-6.

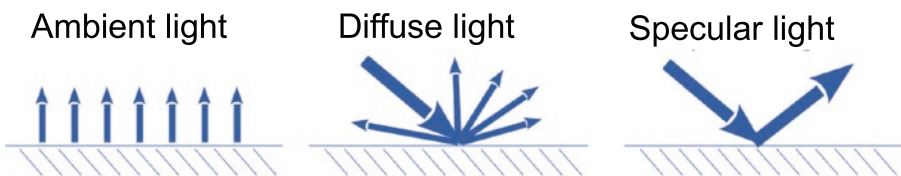


Figure 4-6. Light reflection modes

The material parameters used for the reflected light calculation are:

- The ambient light defined by a light source has its counterpart in an ambient reflection color. The overall ambient part of the final pixel color will be the product of each of the RGB components from the ambient color coming from the light and the ambient color defined as a material parameter.
- As for diffusive light, the pixel color outcome of some material point is calculated by multiplying each color component of the incoming diffusive light with the matching material parameter's diffuse reflectance color component. While for the ambient lighting, the position of lights and surfaces in space does not play a role, for diffuse light the 3D nature of objects unveils.
- While diffuse light reflects in all directions when hitting a surface point, specular light gets reflected as for a mirror, at precisely the same angle on the other side of the normal; see Figure 4-6. The calculation goes as for ambient or diffuse light: each material parameter's RGB component is multiplied by the corresponding RGB component of the light module.
- Especially for specular reflectance, we additionally have the concept of shininess, which describes the amount of fuzziness regarding the viewer's position and the reflected light beam.
- Emissive light. Materials can produce light without the help of a dedicated light switched on. The emissive light is similar to the ambient light contribution, but is not combined with a corresponding light source parameter; it stands for itself and adds to all the other light produced by materials and light sources. Also it does not produce any reflections on other objects.

Sample objects can be seen in Figure 4-7. The way the light comes from the surface parts of course not only depends on light and surface properties, but also on the angle we look at it. This is internally handled by the projection transformation, which is covered in the next section.

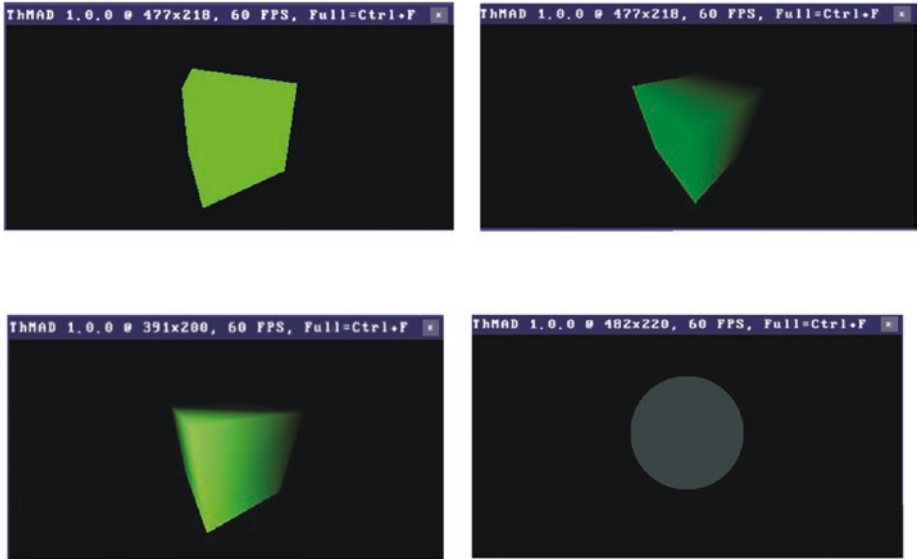


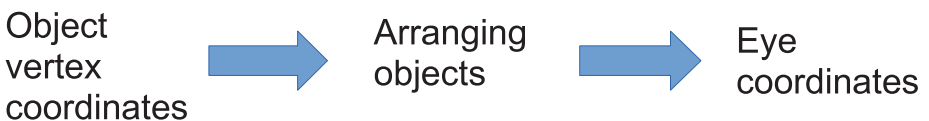
Figure 4-7. Objects with different light reflection modes

In Figure 4-7, the top left is ambient light, the top right is diffuse light, the bottom left is diffuse and specular light, and the bottom right is emissive light showing a 3D sphere.

Eye and Camera

After we place some objects in a scene and add light and material properties, we need something or someone who is actually looking at the scene. And since we have a flat screen as a part of this process, and obviously the latest part the computer program has an influence, this is the equivalent to a rule about projecting the 3D scene onto a flat two-dimensional rectangle, the screen.

The usual OpenGL 3.x methodology for describing this process consists of comparing it to a photography or camera setup. We start with arranging objects in 3D space and compare this to placing objects in a room for to be photographed; see Figure 4-8.



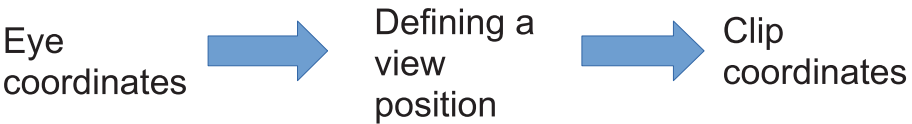
Camera analogy: Placing objects in a room

MODELVIEW MATRIX

Figure 4-8. Modelview transformation

This part of the process ends up with eye coordinates and is governed by the modelview matrix stack. If you do OpenGL programming without using ThMAD, you explicitly tell OpenGL to use the modelview matrix stack. With ThMAD, the program does it for you transparently inside the modules.

Next, the camera is positioned and pointed toward the scene; see Figure 4-9.



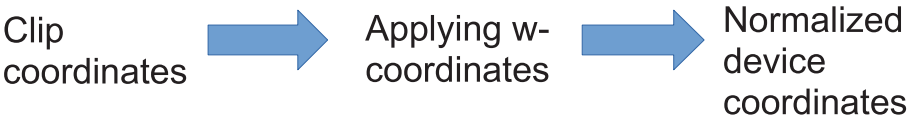
Camera analogy: Position and pointing the camera, choosing a lens

PROJECTION MATRIX

Figure 4-9. Projection transformation

This projection transformation inside OpenGL also allows for the specification of near and far clipping planes, something that doesn't exist in a real-world photography process. Using them gives us the ability to remove objects or object parts that are closer than the near clipping plane or farther away than the far clipping plane. They are part of the projection matrix sub-process because it is easy to do for the hardware at this stage.

After this, the OpenGL system applies the so-called w-coordinates. These are internal numbers and part of the matrix and vector operations. The details are out of scope—it will be enough for our purposes when we say that the hereby used homogeneous coordinates allow for using matrices for translation, which a 3x3 matrix of 3D objects is not capable of. The outcome is part of the two-dimensional screen world and it is called normalized device coordinates. See Figure 4-10.



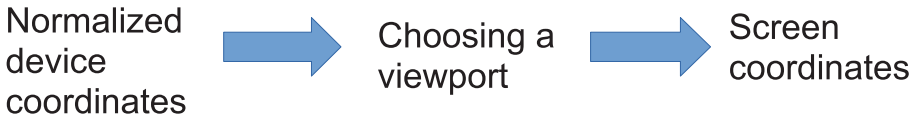
Camera analogy: none.

INTERNAL PROCESS

Figure 4-10. Device-normalization

This is an internal sub-process that has no parameters, hence there is nothing like an associated matrix step.

As a final step, a viewport is chosen and applied, which means we define the actual region we want to see. See Figure 4-11. Also at this step, there are no object-related parameters and hence there is no matrix stack associated with it.



Camera analogy: Choosing the part from the negative for making a photograph

INTERNAL PROCESS

Figure 4-11. Screen coordinates

Note that up to the clip coordinates and the normalized device coordinates, the z coordinate of each vertex is available to the system, and it is needed for determining what needs to be culled from further rendering because it is hidden by other objects. With the camera analogy, physics does that trick for us, but inside OpenGL, those z coordinates, called depth at this stage, are used to determine which vertex points can be ruled out from the rest of the rendering process.

The way of hiding objects is an interesting topic anyway. Lots of energy has been put in the design of graphics hardware, which effectively calculates what is to be taken out of the rendering process because it cannot be seen by the viewer or the camera. To be precise, there are actually two properties that define it:

- *The aforementioned depth buffer.* This is a buffer that traces z -coordinates of pixel coordinates in the rendering step, which actually renders the pixels on the screen. That is to say, the very last rendering step. It works as follows: after the depth buffer is cleared, which usually happens together with the color buffer clearing, any new pixel drawn tells its z -coordinate to the depth buffer, and this leads to one of the following:
 - If the depth buffer is uninitialized at that point, it saves the pixel's z -coordinate in the depth buffer and renders the pixel.
 - If the depth buffer at that point is defined and the z -coordinate of the new point is greater than or equal to the one in the depth buffer, it saves the pixel's z -coordinate in the depth buffer and renders the pixel.
 - If the depth buffer at that point is defined and the z -coordinate of the new point is less than the one in the depth buffer, it discards the new pixel.

- Since we know that greater z-coordinates means being closer to the viewer, this effectively rules out points to be hidden.
- *Backface culling*. This is another way of ruling out pixels, because they are on the back side of an object. Consider for example a cube: each of the six sides consists of two triangles (this is the way quads are handled internally). Each triangle has three vertices. Now if we are careful and define them—if the three vertices from an angular point of view describe a clockwise rotation, they are *defined* to lie on the back side, otherwise on the front side. Because this is just a yes/no decision for each surface part, the graphics hardware can very effectively handle this.

A couple of things need to be taken into account to enable all that hiding magic, but ThMAD together with OpenGL provide two modules tailored for that: a backface culling module and a depth buffer module.

ThMAD Meshes

Meshes in ThMAD are an abstraction of point coordinate sets describing 3D objects.

To define a mesh, certain classes of objects need to be defined. Associated notions are:

- *Vertices*. This is an array of three-dimensional position vectors of the points building the mesh.
- *Vertex normals*. Describe the orientation of surface elements in space. Needed to correctly light objects.
- *Vertex colors*. Vertices often have colors assigned to them.
- *Vertex texture coordinates*. They describe the mapping of vertices onto texture coordinates.
- *Faces*. Precomputed atomic pieces of the surface defining the mesh.
- *Face normals*. These have the same meaning as vertex normals described previously, but associated with the faces. This feature is not used very often.
- *Vertex tangents*. Used by some modules where vectors parallel to surface elements are needed.

ThMAD Particle Systems

Particle systems are about hundreds, thousands, or maybe many more small 3D objects entering a scene, obeying some generation, movement, interaction, and finally decaying rules. You want to use them to simulate fog, waterfalls, candles, bursts, interesting artificial effects, and things like that.

They came into being because of greatly improved graphic hardware capabilities during the last decades. With modern state of the art (or also just medium grade graphics and computation hardware), it is possible to have hundreds of thousands of particles at work. There is no genuine particle system concept inside OpenGL, but of course with so many 3D objects to be handled, the way OpenGL concepts are used is crucial to acceptable performance.

Fortunately, OpenGL provides for some modules that deal with particle systems and internally tune the way OpenGL gets used. The modules are about:

- *Generating particles.* Either given a single point or a set of points
- *Modifying particles and their trajectories.* Some randomization of particle size, precisely controlling all particles sizes, applying wind, letting them act like fluid particles, letting them follow gravity, letting them bounce, and letting them rotate.
- *Controlling particles' life spans.*
- *Controlling the way particles are rendered.*

Albeit experimenting with particle systems is interesting and a lot of fun, there is no chapter dedicated to particle systems. But there is a section in Chapter 6 that provides more insight and presents some examples.

Summary

In this chapter, you learned about 3D coordinate systems, transformation operations like shifting, scaling, and rotating, and about lights and cameras. In addition, you learned about meshes and particle systems inside ThMAD for 3D sketches.

The next chapter contains a series of tutorials or stories that cover a lot of what can be done with ThMAD.

CHAPTER 5



Stories: Basic Level

In this chapter, we dive more into the art of audio visualization by looking at elaborated walk-throughs covering various aspects. We can by no means describe all possible kinds of sketches, but this chapter provides you with techniques that can help you realize your own ideas. There is no particular order for these sections, so you can sequentially work through them or read them per interest—whatever best suits you.

More 3D Rendering Pipelines

ThMAD is built on top of OpenGL, which is an industry standard for 3D graphics rendering. OpenGL is a fundamental technology for your graphics programming needs and you can achieve quite a lot with it. However, contrary to OpenGL, ThMAD operates at the more abstract concepts of instantiating, configuring, and connecting modules. There are advantages and disadvantages to this approach—one advantage is that you don't have to be a programming expert to use ThMAD. A disadvantage is you cannot do everything with ThMAD you can do with OpenGL.

This chapter contains more examples that are explained in-depth. This chapter has as an additional benefit that in the end, you'll understand some parts of OpenGL, since the same or at least similar concepts and names are used for objects and parameters.

Transformations

Transformations here involve translation, rotation, and scaling.

■ **Note** Samples from this subsection are available as source under A-5.1.1_* inside the TheArtOfAudioVisualization folder.

The modules that can do transformations for vertex coordinates are listed in Table 5-1.

Table 5-1. Transformation Modules

| Module | Access Point |
|-------------------|---|
| Translate | Renderers → opengl_modifiers → gl_translate |
| Rotate | Renderers → opengl_modifiers → gl_rotate |
| Rotate | Renderers → opengl_modifiers → gl_rotate_quat |
| Scale | Renderers → opengl_modifiers → gl_scale |
| Scale (uniformly) | Renderers → opengl_modifiers → gl_scale_one |

We start with a basic 3D setup. Using the menu on the left side, place the following modules on the canvas:

- Renderers → opengl_modifiers → cameras → orbit_camera
- Renderers → opengl_modifiers → light_directional
- Renderers → opengl_modifiers → auto_normalize
- Renderers → opengl_modifiers → material_param
- Renderers → opengl_modifiers → backface_culling
- Renderers → opengl_modifiers → depth_buffer
- RENDERERS → mesh → mesh_basic_render
- Mesh → solid → mesh_box

Set the following anchor values, leaving unmentioned values at their defaults. See Tables 5-2 to 5-8.

Table 5-2.

| renderers → opengl_modifiers → cameras → orbit_camera | | |
|---|-----|---|
| perspective_correct | yes | Make sure the computer screen’s aspect ratio is being taken care of and a cube looks like a cube, with all sides the same size on the screen. |

Table 5-3.

| renderers → opengl_modifiers → light_directional | | |
|--|-----|----------------------|
| enabled | YES | Switch the light on. |

Table 5-4.**renderers → opengl_modifiers → material_param**

| | | |
|----------------------|-----------|--|
| specular_reflectance | Any color | Shining color; you usually use white or dim white, or at least a bright color. |
| diffuse_reflectance | Any color | Usually the basic color of your object. |
| specular_exponent | 30 | Minimum 0, maximum 128. Specifies the shininess, spottiness, or anti-fuzziness of the specular reflectance effect. |

Table 5-5.**renderers → opengl_modifiers → backface_culling**

Makes sure the backface is not rendered. See 4.5 for a description.

| | | |
|--------|---------|-----------------------------|
| status | ENABLED | Switch backface culling on. |
|--------|---------|-----------------------------|

Table 5-6.**renderers → opengl_modifiers → depth_buffer**

Makes sure hidden surfaces are not rendered.

| | |
|------------|---------|
| depth_test | ENABLED |
| depth_mask | ENABLED |

Table 5-7.**renderers → mesh → mesh_basic_render**

Leave all values at their defaults.

Table 5-8.**mesh → solid → mesh_box**

The object to be drawn: a cube.

Leave all values at their defaults.

Some notes on setting the anchor values:

- Some anchors, like for example the enabled anchor of `light_directional`, are by default not seen in the module. Instead they are part of the so-called *complex* anchors. Complex anchors can be opened by clicking on them to unveil their children.
- For enumeration type values, just click on them and select the appropriate value. The anchor enabled of `light_directional` is an example for an enumeration value.
- For color values, double-click on them and select the color. Click and drag on the top row of the color cube, as shown in Figure 5-1 to select a HUE value, then click on a point inside the bigger bottom-left square to choose SATURATION and VALUE.

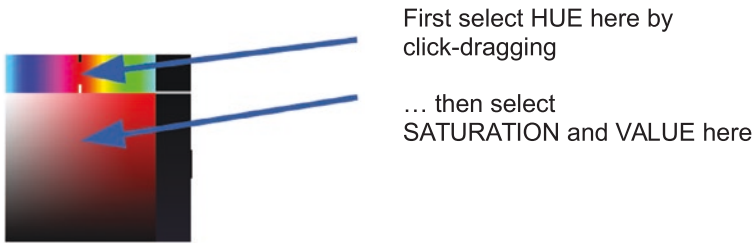


Figure 5-1. *The color cube for entering color values*

This is the HSV color space used throughout ThMAD. Internally it is converted into RED-GREEN-BLUE values, which OpenGL needs instead. Close it by double-clicking on the top-right small square.

Connect the modules as shown in Figure 5-2.

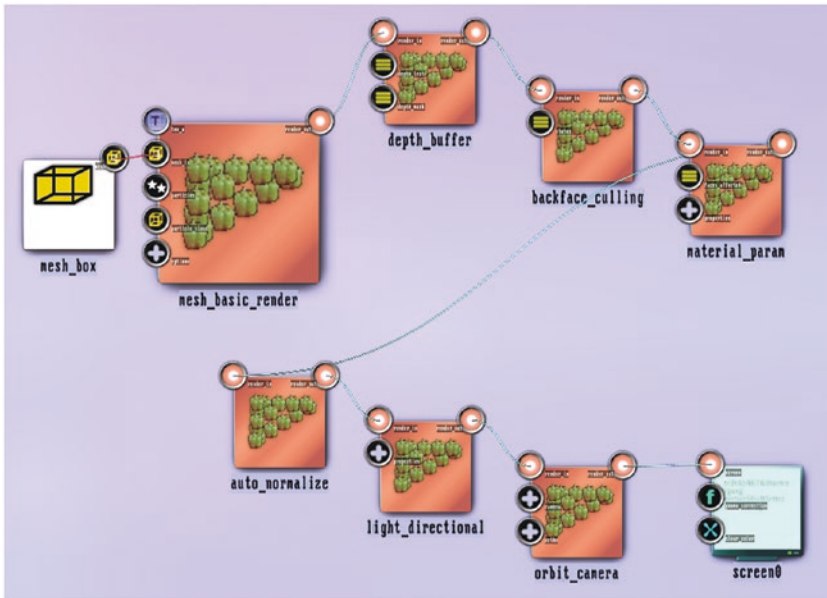


Figure 5-2. Basic transformation setup

Click and drag on an output anchor and release over the input anchor you want to connect to.

A note about order here: OpenGL under the hoods of ThMAD is a state machine and order does not matter at all points. It however *is* important quite often, especially when it comes to matrix operations. In the state just described, the `orbit_camera` and `light_directional` modules have an orientation in space described by a matrix. The order is crucial here: if we had the light *after* the camera, the light's position would rotate with the camera, as if the light and the camera were tied. But here we want to have the light being fixed if the camera gets rotated, hence the light needs to be positioned before the camera in the state.

The inconspicuous module called `auto_normalize` is new here. It fixes the normal vectors after transformations and before applying lighting. Under certain circumstances, especially with any kind of scaling transformations, forgetting it might give you strange lighting issues and sleepless nights.

■ **Caution** In any 3D state where there's the slightest chance of scaling happening, add the module `Renderers` → `opengl_modifiers` → `auto_normalize` before the lighting.

The outcome will be a front-facing view of a cube, as shown in Figure 5-3.



Figure 5-3. Basic state outcome

What we want to do now is add a swinging rotation, with the elongation modulated by incoming sound basses, and add a scaling of the size, modulated by incoming sound trebles. We start with the rotations. To proceed, add the `Renderers` → `opengl_modifiers` → `gl_rotate` module twice and insert both between `mesh_basic_render` and `depth_buffer`. To do so, right-click on the line between them, choose `Disconnect`, and then draw the new connections shown in Figure 5-4.

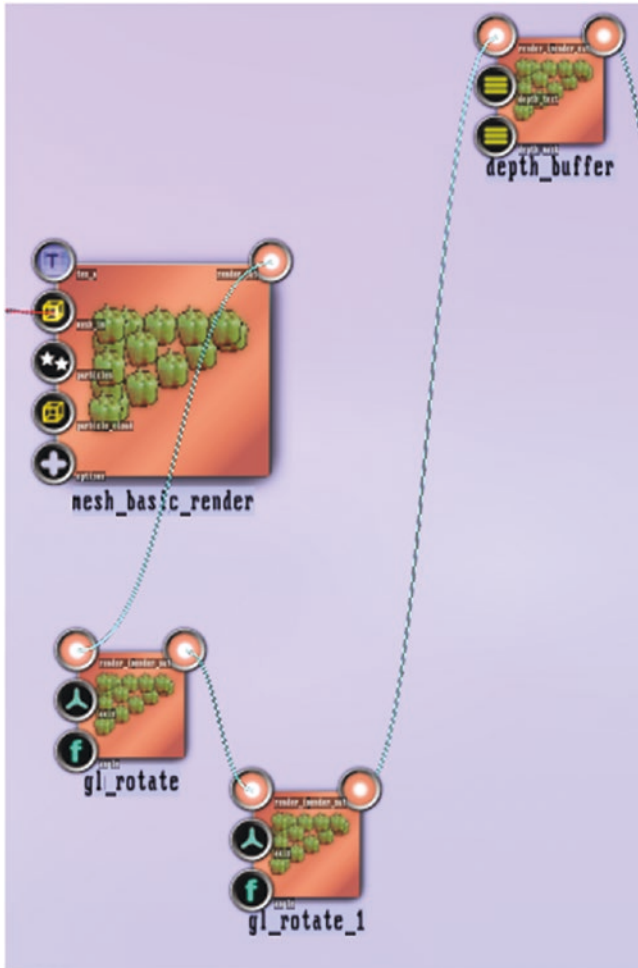


Figure 5-4. Adding rotation modules

When adding several modules of one type, ThMAD automatically numbers them. For example, for the `gl_rotate` we just added twice, the first will have the name `gl_rotate`, the second `gl_rotate_1`. The next one would be called `gl_rotate_2`, and so on. This is the external name that you see on the canvas under the module as well as the internal ID, which never changes until the module is deleted.

The rotation module by default has a rotation axis (0;0;1), which means parallel to the z-axis, and an rotation angle of 0.0. The anchors are accordingly called *axis* and *angle*. We now change the axis of the first rotation module to (1;0;1): double-click on the axis anchor and move the x-slider. This is the first one, all the way up to 1.0. See Figure 5-5.

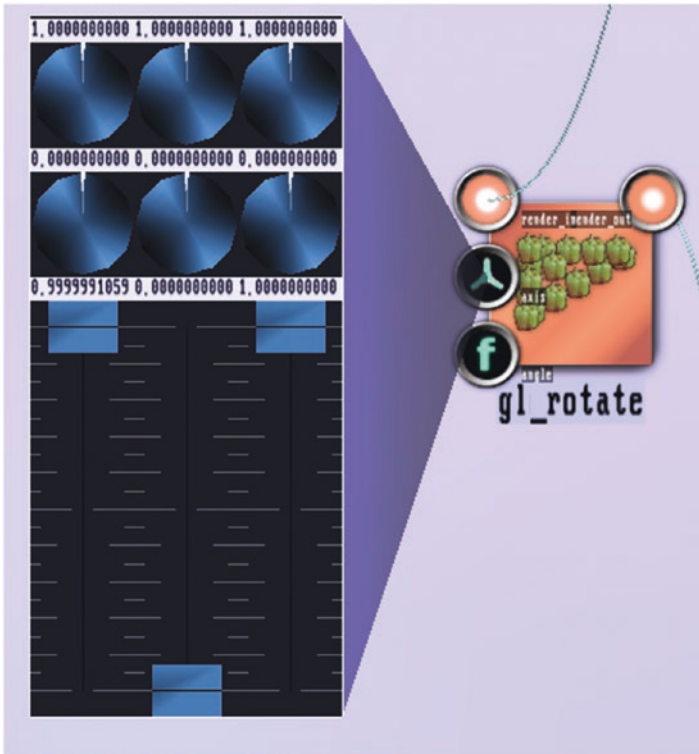


Figure 5-5. *Rotation axis slider*

Close the triple slider by double-clicking anywhere on it. Leave the other rotation module’s axis unchanged, so that one will rotate about its default, the z-axis. For the steady rotation effect, we add two Maths → oscillators → oscillator modules and set their anchors as shown in Tables 5-9 and 5-10.

Table 5-9.

maths → oscillators → oscillator

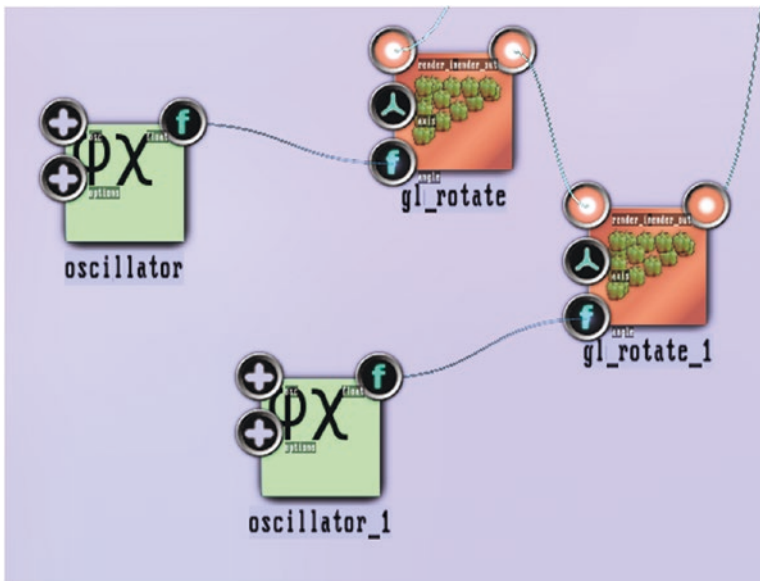
First “rotator engine”

| | | |
|----------|------|----------------------------------|
| osc_type | saw | Explanation below. |
| freq | 0.55 | Can choose other values as well. |

Table 5-10.**maths → oscillators → oscillator***Second “rotator engine”*

| | | |
|----------|------|--------------------------------------|
| osc_type | saw | For an explanation, see below. |
| freq | 0.25 | You can choose other values as well. |

Now connect their outputs to the angle input anchors of the `gl_rotate` modules. See Figure 5-6.

**Figure 5-6.** Steady rotation

Setting the float type anchor values happens by double-clicking and then clicking



and dragging on the knob. After you click on a knob, you drag the mouse up and down for coarse changing the value, or drag left and right for fine changing the value. Or click on the digit and change a value using the keyboard, followed by Enter (use Ctrl+Del to clear the string at any time).

The reason we use a saw oscillator is its linear transition from 0.0 to 1.0 and then abruptly going back to 0.0. On the other hand, angular input is modulus 1.0, which means an angle of 1.0 represents a full rotation. That way, the saw oscillator provides for a steady uniform rotation. Now for the translation, we add these modules:

- Renderers → opengl_modifiers → gl_translate
- Maths → converters → 3float_to_float3

Put the `gl_translate` module between the `gl_rotate_1` and `depth_buffer` modules and connect the `3float_to_float3` module to the translation anchor of the `gl_translate` module, as shown in Figure 5-7.

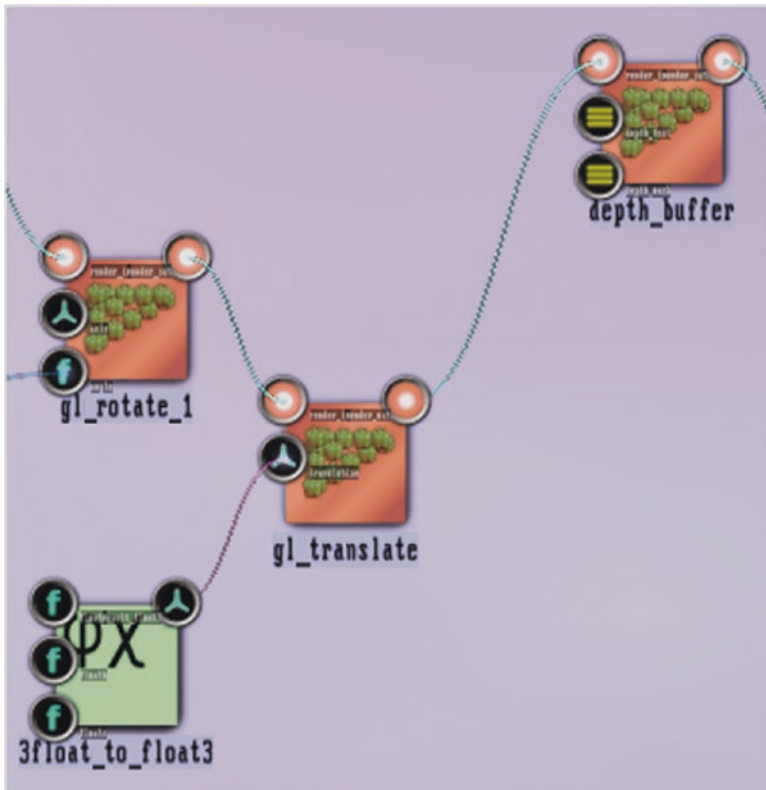


Figure 5-7. Translation added

The `3float_to_float3` allows for multiplexing the translation vector—each of the x, y, and z components of the translation can be addressed independently that way. For the movement motor, we add another oscillator:

Maths → oscillators → oscillator

We connect it to the first input anchor of `3float_to_float3` and set its anchors as shown in Table 5-11.

Table 5-11.

maths → oscillators → oscillator

The second oscillator, oscillator_2

| | | |
|----------|-----|------------------------------|
| osc_type | sin | |
| ofs | 0.0 | Let it oscillate around 0.0. |

As a last transformation module, we add a module that blows up or shrinks the box, the `_one` in the name means uniform scaling in all three dimensions:

Renderers → `opengl_modifiers` → `gl_scale_one`

Put it between `gl_translate` and `depth_buffer`. See Figure 5-8.

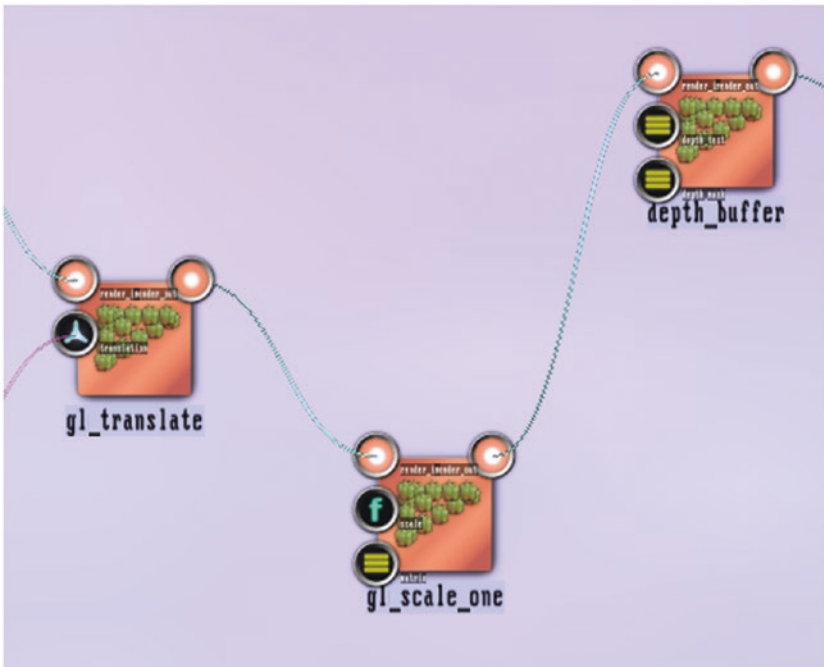


Figure 5-8. *Scaling added*

As a last step, we add sensitivity to sound. To do so, place the Sound → input_visualization_listener module on the canvas, and twice maths → arithmetics → ternary → float → mult_add. Connect the octaves_1_0 output anchor of the visualization listener module to the float_in anchor of the first mult_add module, and the octaves_1_4 anchor similarly to the second mult_add module. See Figure 5-9. The octaves_1_0 is for bass tones and the octaves_1_4 is for treble tones. The other input anchor values are for sound volume scaling and setting a base value. More precisely, see Tables 5-12 and 5-13.

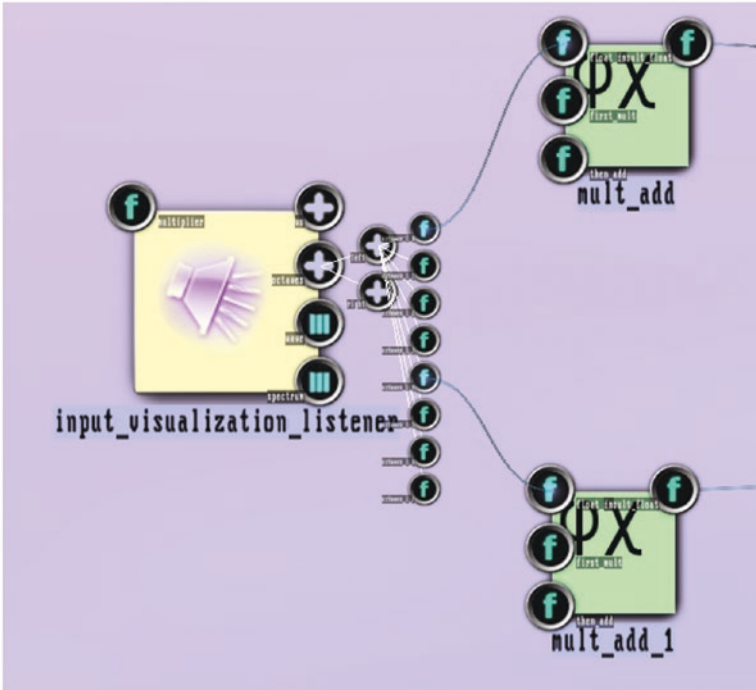


Figure 5-9. Sound input added

Table 5-12.

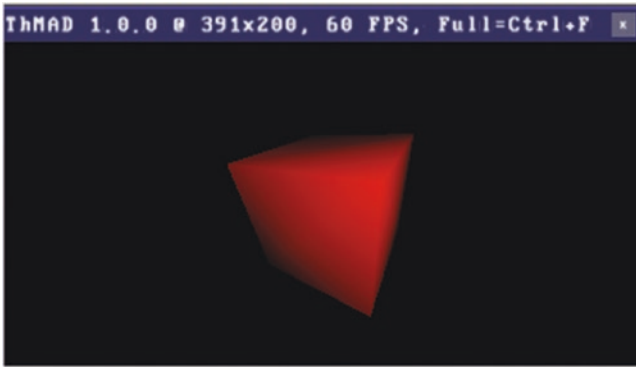
| maths → arithmetics → ternary → float → mult_add | | |
|---|-----|----------------------------------|
| <i>The one connected to the bass tones and the translation module</i> | | |
| first_mult | 3.0 | Volume scaling |
| then_add | 0.1 | Some movement even with no sound |

Table 5-13.**maths → arithmetics → ternary → float → mult_add***The one connected to the treble tones and the scaling module*

| | | |
|------------|------|---------------------------------|
| first_mult | 10.0 | Volume scaling |
| then_add | 0.5 | Some size even with no sound |

Finally, connect the bass `mult_add`'s output to the `amp` anchor of `oscillator_2`, and the treble `mult_add`'s output to the `scale` anchor of the `gl_scale_one` module.

Play some music and you can see the box moving and inflating to the incoming sound, as shown in Figure 5-10.

**Figure 5-10.** *Outcome of the sound input added*

If the effect shows up too reluctantly, change the amplification input anchor multiplier of the visualization listener.

The movement and inflation of the box is a little edgy, since incoming sound is reacted on very fast. You can smooth that if you insert `maths → interpolation → float_smoother` modules after the `mult_add` modules. This is left as an exercise for you.

Wireframes

Wireframe models reduce the complexity of rendering 3D objects by simply not drawing any areas and instead only drawing the edges of objects. It was introduced for debugging purposes, since not drawing inside area points improves speed quite noticeably. But wireframe models also may show their own aesthetics.

■ **Note** Samples from this subsection are available as source under A-5.1.2_* inside the TheArtOfAudioVisualization folder.

As a starting point, we paint three types of spheres and compare their wireframes. To do so, let's first build the basic 3D setup.

Insert the module Renderers → opengl_modifiers → blend_mode between mesh_basic_render and depth_buffer. This module is a good way to express that you want to combine several sub-rendering pipelines. There you can also specify what happens to the pixel color when several objects overlap. We want to present objects side-by-side, so do not expect an overlap here, but using the blend_mode as a combiner is a good means to structure your state. Remember you can insert a module into an existing connection by first deleting that connection. You do so by right-clicking the connection and choosing Disconnect, then reconnecting everything as desired.

Insert these modules between mesh_basic_render and blend_mode:

- Renderers → opengl_modifiers → gl_scale_one
- Renderers → opengl_modifiers → gl_translate

See Figure 5-11.

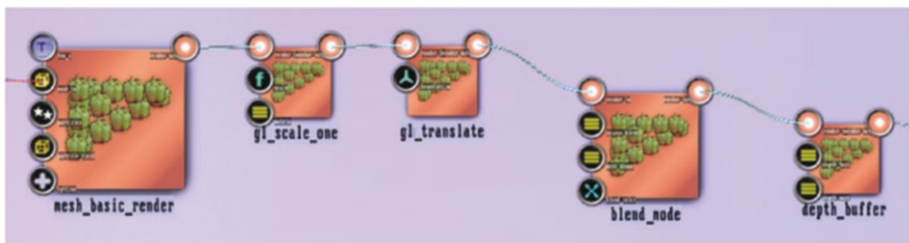


Figure 5-11. Preparing the wireframed views

Change the scale anchor of gl_scale_one to 0.75, as shown in Table 5-14.

Table 5-14.

| | |
|---|------|
| renderers → opengl_modifiers → gl_scale_one | |
| scale | 0.75 |

Change the x-coordinate of the translation anchor of module gl_translate to -1.8, as shown in Table 5-15.

Table 5-15.

| renderers → opengl_modifiers → gl_translate | |
|---|--------------|
| translation | (-1.8; 0; 0) |

To do so, double-click on the translation anchor and use the left knobs and the left slider to adjust the x-value, as shown in Figure 5-12.

**Figure 5-12.** Triple value slider

The upper knob specifies the range available to the slider, and the lower knob the offset. To change a knob's value, click and drag it. A vertical mouse movement is for coarse changes and a horizontal mouse movement is for fine changes. So, if you change the lower knob to something like -2.0 and leave the upper knob at its default value 1.0, you can use the slider to achieve the desired -1.8. To move the slider, also click and drag on it. Close the triple controller by double-clicking anywhere on it.

Delete the box module you had from following the instructions earlier. Click on it, then press Del on your keyboard. Or use the context menu you get after right-clicking to delete it. Now place the module mesh → solid → mesh_sphere next to mesh_basic_render and connect the sphere to the mesh_basic_render module's mesh_in anchor. Change the sphere's anchors as shown in Table 5-16.

Table 5-16.

| mesh → solid → mesh_sphere | |
|----------------------------|----|
| num_sectors | 30 |
| num_stacks | 10 |

After changing the camera's settings a little, I use 2.8 as a distance and 65.0 as a fov to diminish the perspective effect a little. You can see this in Figure 5-13.

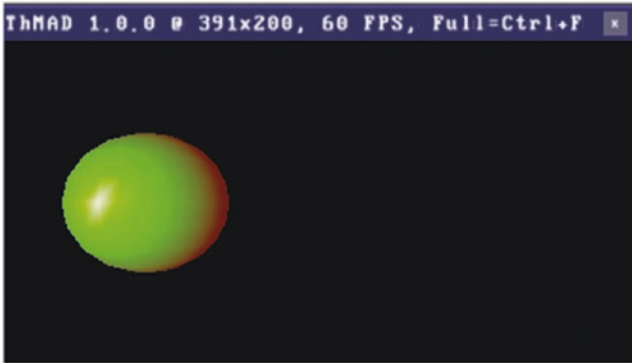


Figure 5-13. A basic sphere

For the other two sphere types, we clone the `mesh_basic_render`, `gl_scale_one`, and `gl_translate` modules. Cloning is a copy with all of the actual values, and it helps you quite often to simplify state generation. To perform a cloning, press and hold Ctrl and Alt, then click and drag the module in question.

For the cloned modules, connect them similarly to the original, connect them to `blend_mode`, and this time, add the `Mesh → solid → mesh_sphere_icosahedron` module as input. Change the x-translation of the cloned `gl_translate` to read 0.0. Make another clone the very same way, but use a x-translation value of +1.8 this time and add `Mesh → solid → mesh_sphere_octahedron`.

As for the spheres' parameters, set the values as shown in Tables 5-17 and 5-18.

Table 5-17.

| <code>mesh → solid → mesh_sphere_icosahedron</code> | |
|---|-----|
| <code>subdivision_level</code> | 4.0 |
| <code>max_normalization_level</code> | 10 |

Table 5-18.

| <code>mesh → solid → mesh_sphere_octahedron</code> | |
|--|-----|
| <code>subdivision_level</code> | 4.0 |
| <code>max_normalization_level</code> | 10 |

The generator part of the state will look like Figure 5-14, and the output like Figure 5-15. Now actually switching to the wireframe mode view is easy. Between the camera module and the screen module, insert a module `Renderers` → `opengl_modifiers` → `rendering_mode`.

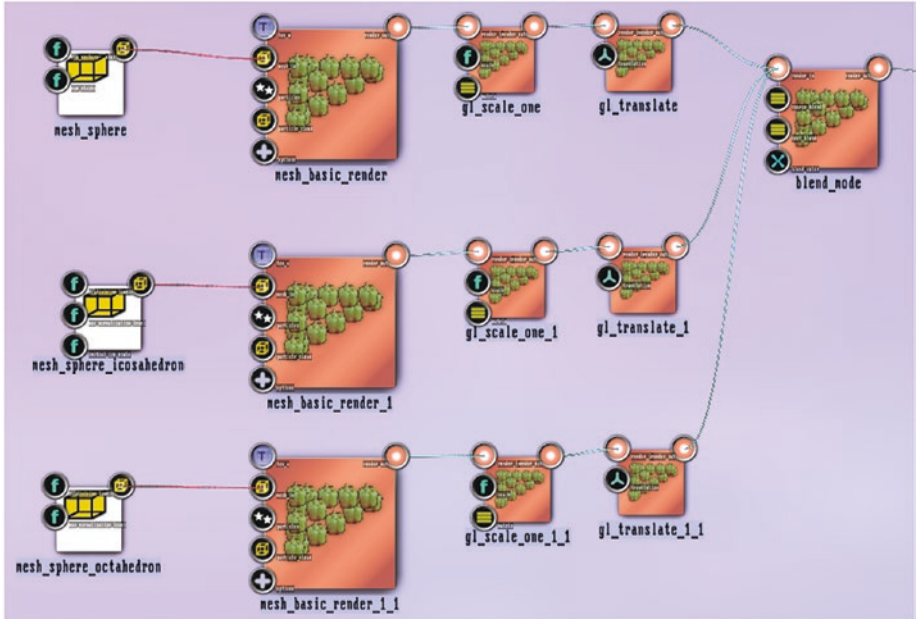


Figure 5-14. Three spheres: state (part)

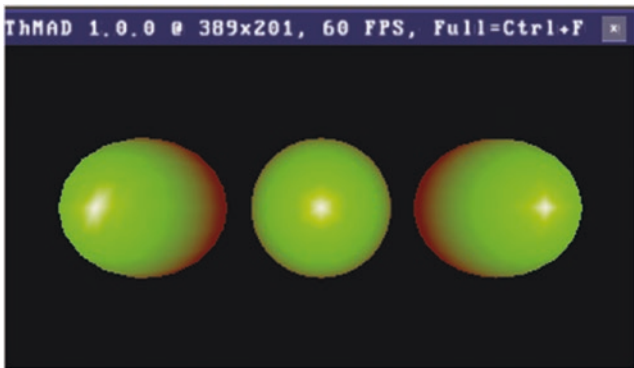


Figure 5-15. Three spheres of different kinds

Change its anchors as shown in Table 5-19.

Table 5-19.

| <code>renderers</code> → <code>opengl_modifiers</code> → <code>rendering_mode</code> | |
|--|--------------------|
| <code>back_facing</code> | <code>lines</code> |
| <code>front_facing</code> | <code>lines</code> |

The final output can be seen in Figure 5-16. You can press Ctrl+F to switch to fullwindow mode and investigate the details.

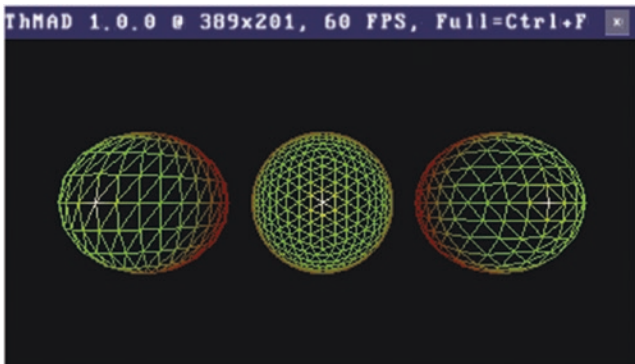


Figure 5-16. Three spheres of different kinds, in wireframe mode

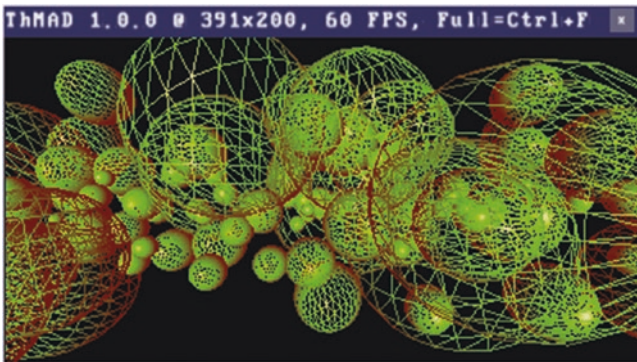
If you want to use this as a basis for something more dynamic and more rich, this can be achieved as follows. Remove the octahedron and icosahedron part of the generation and then move the remaining sphere back to (0; 0; 0) by accordingly changing the x-coordinate of the translation anchor of `gl_translate`.

Then add the module `Particlesystems` → `generators` → `basic_spray_emitter` and set its anchors as shown in Table 5-20.

Table 5-20.**particlesystems → generators → basic_spray_emitter**

| | | |
|--|-----|--------------------------------------|
| num_particles | 100 | |
| particles_per_second | -1 | Because of -1, “num_particles” rules |
| spatial / size / speed / speed_x | 9.0 | |
| spatial / size / particle_size_base | 1.0 | |
| spatial / size / particle_size_random_weight | 3.2 | |
| appearance / time / particle_lifetime_base | 4.0 | |
| appearance / time / particle_lifetime_random_weight | 1.0 | |

What this particlesystem spray emitter does is steadily produces, moves, and after some time, kills wireframe cubes. Connect the `basic_spray_emitter` to the `particles` input anchor of `mesh_basic_render` to start the dynamics. The output will look like Figure 5-17.

**Figure 5-17.** Massive and dynamic wireframes

There is a lot more to say about particle systems and we will meet them again at various places throughout the text.

Responsiveness to sound can be added, as shown earlier in this chapter. You just connect sound to any input anchor. The speed, the number, the base size, and/or the size randomization are all suitable candidates.

The Ocean Module

ThMAD has a quite appealing ocean module, explained in this section.

Start with the basic setup described earlier in the chapter until `mesh_box`. You do not need that one here. Instead add the Mesh → generators → ocean module and connect it to `mesh_basic_render`. As for the anchor values, start with setting the sky color in the screen module, as shown in Table 5-21.

Table 5-21.

| screen0 | | | |
|-------------|----------------------|---------|--|
| clear_color | 0.19; 0.40; 1.0; 1.0 | The sky | |

Note that double-clicking on an anchor selects its default input method. For colors, this is usually the color controller you already know, but not for the screen module. This decision is made by each module separately. In this case, the quadruple slider combo shows up, as shown in Figure 5-18.



Figure 5-18. The quadruple slider combo

Here, you can set each color channel value using the slider, with the range of each slider given by the knob in the first row, and the lowest value or offset given by the knob underneath it. Those limits have default values, and the range = 1.0 and offset = 0.0 makes total sense here for color values naturally ranging from 0.0 to 1.0. You don't have to change the limit knobs here. Or, you can click on the textual slider value right above the slider to enter a value using the keyboard (use Ctrl+Del to first clear the field).

Or, you can force the color picker to show up by right-clicking on the anchor and choosing Color, as shown in Figure 5-19.

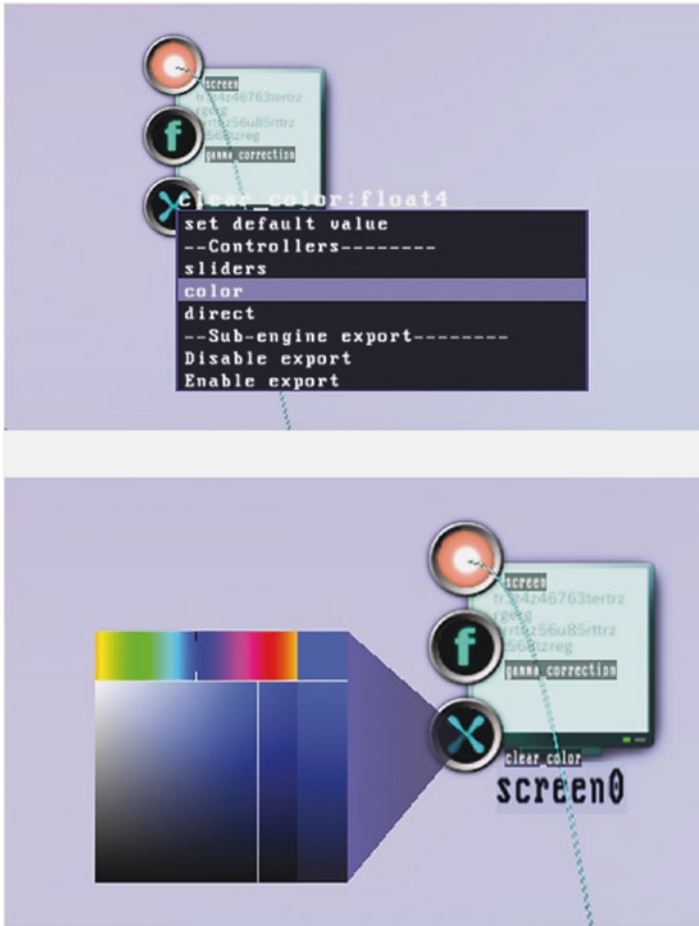


Figure 5-19. Accessing the graphical color chooser

Or, you can choose the direct entry mode using the same context pop-up, as shown in Figure 5-20.

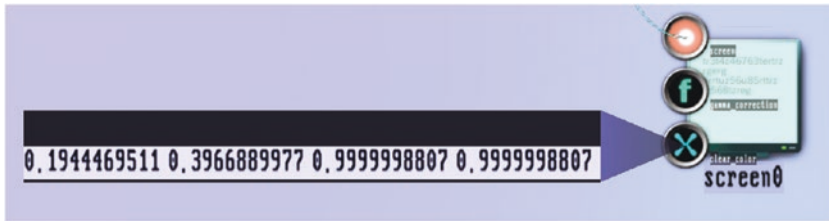


Figure 5-20. Direct color entry process

You can then click on each value and change it by using the keyboard and pressing Enter afterward. Also here, you use Ctrl+Del to clear a value. The first three values stand for RGB, each component lies inside [0.0;1.0], and the last for the ALPHA value. As usual, close any of the controllers by double-clicking on it. The other module anchor values for the ocean are shown in Tables 5-22 to 5-28.

Table 5-22.

| renderers → opengl_modifiers → cameras → orbit_camera | | |
|---|------------------|--|
| rotation | 0.0; -0.82; 0.12 | We need a suitable camera view, since the ocean plate in ThMAD is limited, but we want the impression of an endless horizon. |
| distance | 20.0 | |
| fov | 66.0 | |

Table 5-23.

| renderers → opengl_modifiers → light_directional | | |
|--|---------------------------|------------------------------------|
| position | -0.19; 0.71; 1.0 | |
| ambient_color | 0.16; 0.22; 0.23; 0.42 | Turquoise ambient color |
| diffuse_color | 0.94; 0.92; 1.0; 1.0 | Ocean blue diffuse color |
| specular_color | 0.95; 0.94; 0.81; 1.0 | A little yellowish bright specular |

Table 5-24.**renderers → opengl_modifiers → material_param**

| | | |
|-------------------|-------------------|--------------------------|
| ambient_ | 0.21; 0.19; 0.06; | Dark yellow ambient |
| reflectance | 0.98 | reflectance |
| diffuse_ | 0.03; 0.13; 0.83; | Blue diffuse reflectance |
| reflectance | 1.0 | |
| specular_ | 0.92; 0.91; 0.68; | Light yellow specular |
| reflectance | 1.0 | reflectance |
| specular_exponent | 20.0 | |

Table 5-25.**renderers → opengl_modifiers → backface_culling**

| | | |
|--------|----------|---|
| status | DISABLED | Needs to be disabled because of an internal bug |
|--------|----------|---|

Table 5-26.**renderers → opengl_modifiers → depth_buffer**

| | |
|------------|---------|
| depth_test | ENABLED |
| depth_mask | ENABLED |

Table 5-27.**renderers → mesh → mesh_basic_render**

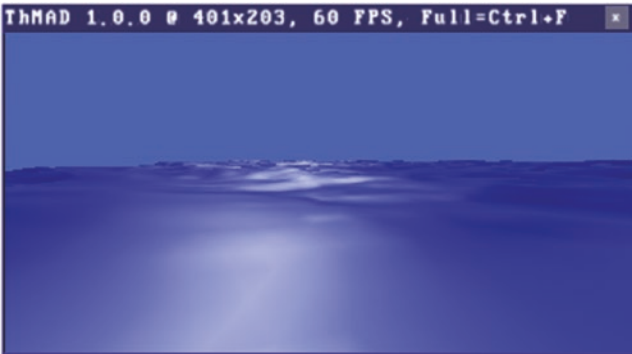
| | | |
|---------------------------|----|--|
| vertex_colors | no | Let the material_param module define the color |
| use_display_list | no | |
| use_vertex_colors | no | |
| particles_size_center | no | |
| particles_size_from_color | no | |
| ignore_uv_in_vbo_updates | no | |

Table 5-28.**mesh → generators ocean**

| | | |
|--------------|------|---------------------------------------|
| time_speed | 3.2 | Time multiplier for the wave movement |
| wind_speed_x | 5.0 | |
| wind_speed_y | 12.0 | |
| lambda | 1.12 | Wave speed |
| normals_only | no | |

■ **Note** The source is available under A-5.1.3_Ocean inside the TheArtOfAudioVisualization folder.

The result is shown in Figure 5-21.

**Figure 5-21.** Ocean output

Texture Mapping

A texture, more precisely a texture *map*, is a way to efficiently describe the surface characteristics of each part of a 3D object using 2D images that were preloaded on the graphic hardware.

There is a technical reason for doing this. Consider a real-world moving 3D object. In order to represent it realistically, we must describe it by looking at millions of microscopic colored points. Each of these points is moving and needs to be calculated several dozens of times each second. If all that happened in the CPU of a computer, way too much data would have to be shifted to the graphics hardware each second. Instead, as a quite clever way out of this dilemma, we describe each 3D object by a much smaller

number of points or vertices building a mesh spanned over the surface, and map a suitable part of a 2D preloaded image over each face of the mesh. This effectively reduces the amount of data to be transferred per second to the graphics hardware by magnitudes.

The primary mathematical description of this texture mapping consists of assigning two coordinate sets to each vertex building up the shape. For example, if you draw a quad face made of these coordinates

$$(x_1, y_1, z_1) \rightarrow (x_2, y_2, z_2) \rightarrow (x_3, y_3, z_3) \rightarrow (x_4, y_4, z_4) \rightarrow (x_1, y_1, z_1)$$

and you want to map a part of a 2D texture on it, more precisely a quad:

$$(u_1, v_1) \rightarrow (u_2, v_2) \rightarrow (u_3, v_3) \rightarrow (u_4, v_4) \rightarrow (u_1, v_1)$$

then this double set of coordinates is exactly what you have to provide inside a program to describe a texture map:

$$(x_1, y_1, z_1, u_1, v_1) \rightarrow (x_2, y_2, z_2, u_2, v_2) \rightarrow$$

$$(x_3, y_3, z_3, u_3, v_3) \rightarrow (x_4, y_4, z_4, u_4, v_4) \rightarrow$$

$$(x_1, y_1, z_1, u_1, v_1)$$

Graphically, this is depicted in Figure 5-22. Note that there is a lot more to say about textures. Textures also come in 1D, 3D, or even 4D flavors, stacked storage types for different resolutions, and more. In this book, we only describe a part of all these features, but the material you can read on the Internet or other books can be overwhelming, so feel free to extend your research using other sources if you like.

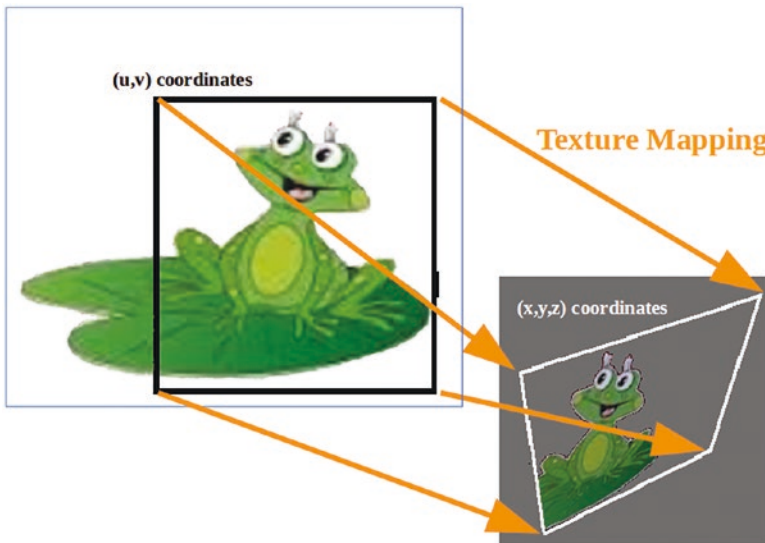


Figure 5-22. Basic texture mapping

First, in ThMAD there is no generic way to describe such a point-wise texture mapping in any way that the structure of the objects to be textured and the textures require. That is because you usually don't enter point coordinates, but shapes of one or the other form, and there is no reliable way to guess where each texture point needs to be mapped. But there are workarounds covering some interesting cases, using automatic texture coordinates generation, and also a feature in OpenGL called *shaders*. Both give intriguing effects and are important for your visualization ideas.

Automatic Texture Coordinates

OpenGL has a function called `glTexGen()`. It describes an old legacy technique to let the rendering engine automatically generate a texture mapping. It is now superseded by shading, which is not covered in this book. It has some shortcomings in expressiveness.

■ **Note** This source is available under `A-5.2.1_Automatic_texture_coordinates` inside the `TheArtOfAudioVisualization` folder.

Nevertheless, you can use it for your sketches, since there is a module in ThMAD covering it. The name of this module is `Texture → opengl → texture_coord_gen`. In order to use it, you also need a basic 3D pipeline, a mesh, and a texture. As walk-through examples, the following sections describe the mapping of a texture onto a sphere and on one side of a cube. Start with these modules:

- `orbit_camera`
- `light_directional`
- `material_param`
- `backface_culling`

Connect the camera to the screen module `screen0`. This is the backend of the pipeline. From the other side, choose the left side of the canvas if you like. We start with a chain and connect them all:

- `Texture → loaders → png_tex_load`
- `Texture → modifiers → scale`
- `Texture → modifiers → tex_parameters`
- `Texture → modifiers → translate`
- `Renderers → mesh → mesh_basic_render`
- `Texture → opengl → texture_coord_gen`

Also connect the `render_out` anchor of the `texture_coord_gen` module to the `render_in` anchor of the `material_param` module. As a last module, we add `Mesh → solid → mesh_sphere_octahedron` and connect its output to the `mesh_in` anchor of the `mesh_basic_render` module. See Figure 5-23.

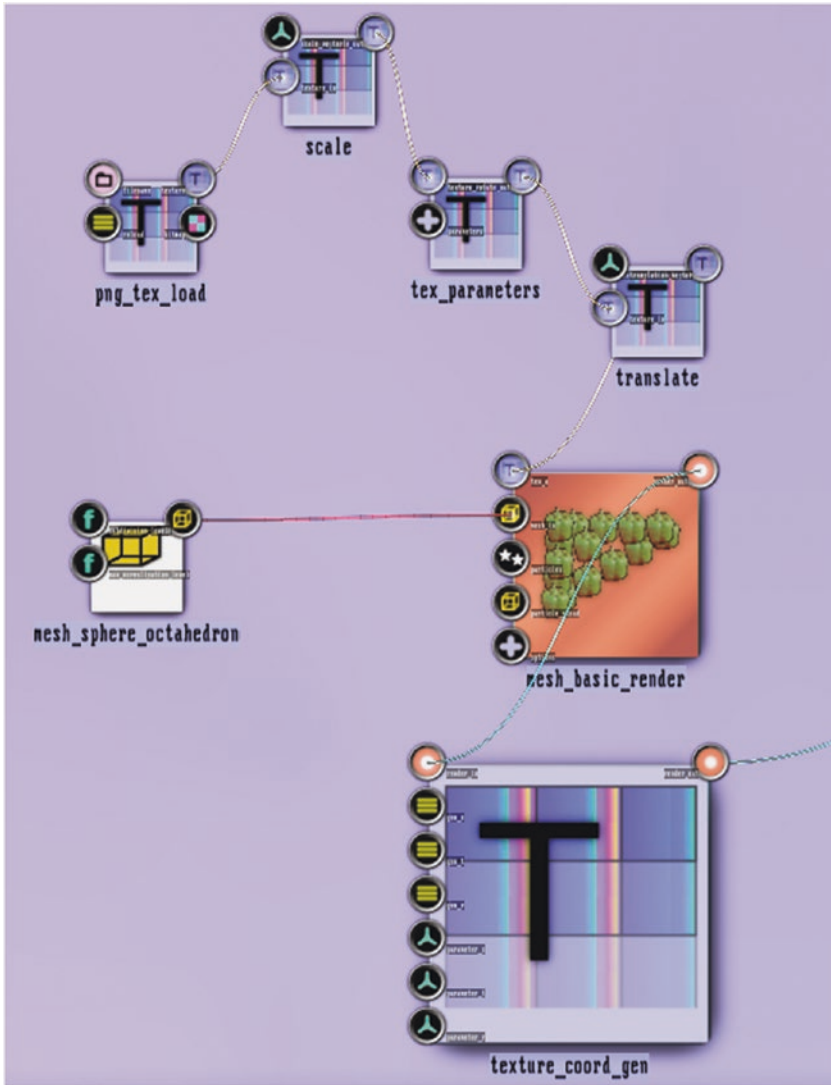


Figure 5-23. Automatic texture coordinates

To prepare for these anchor parameters, place a PNG image inside this folder:

[HOME]/thmad/[VERSION]/data/resources

Make sure it has a size of 128x128 or 256x256, and let its background be transparent. Having the size of textures be the same in both dimensions and a power of two is generally a good idea. The little frog that I'm going to use is already there inside the `TheArtOfAudioVisualization` folder; it's called `frosch2.png`. The anchor parameters are shown in Tables 5-29 to 5-34.

Table 5-29.

| texture → loaders → png_tex_load | | |
|---|--------------------------|--|
| filename | PNG image from resources | Selection via resource browser; double-click on your choice. |

Table 5-30.

| texture → modifiers → scale | | |
|------------------------------------|---------|---|
| scale_vector | 1; 1; 1 | The mapped size of the texture; you can later play with those values. |

Table 5-31.

| texture → modifiers → translate | | |
|--|---------|--|
| translation_vector | 0; 0; 0 | The mapped position of the texture - you can later play with those values. |

Table 5-32.

| texture → modifiers → tex_parameter | | |
|--|--------|---|
| parameters / wrap_s | repeat | You can later change the repetition mode to show only one single mapped texture, or a row, or a column of repeating textures. |
| parameters / wrap_t | repeat | |

Table 5-33.

| mesh → solid → mesh_sphere_octahedron | | |
|---------------------------------------|------|---|
| max_normalization_level | 10.0 | Smoothness; change this one prior to the subdivision_level! |
| subdivision_level | 6.0 | Smoothness |

Table 5-34.

| texture → opengl → texture_coord_gen | | |
|--------------------------------------|---------------|---|
| gen_s | OBJECT_LINEAR | Note that texture coordinates in OpenGL are regularly denoted by S, T, and R. This is not a typo, the strange order is due to historic reasons. |
| gen_t | OBJECT_LINEAR | |
| gen_r | <any> | Ignored, since we have 2D textures. |

In your backend, make sure the anchor status of `backface_culling` is switched to `ENABLED`, the light is switched on at `properties/enabled`, and the perspective correction is switched on in the camera module. Also, you can set the camera rotation parameter to `(0.0; 0.0; 1.0)`. The result will look something like Figures 5-24 and 5-25.

**Figure 5-24.** Automatic texture coordinates, take 1



Figure 5-25. Automatic texture coordinates, take 2

At first sight this seems to be a correct mapping of the picture on to the sphere. But in fact, the formula used is not realistic. Due to the `OBJECT_LINEAR` mode of the module, each of the (u,v) coordinates is mapped to a linear combination of the (x,y,z) space: $u = a \cdot x + b \cdot y + c \cdot z$ and $v = d \cdot x + e \cdot y + f \cdot z$. This is only realistic near the center of the sphere's face, and only if looking straight on the x,y -plane. Playing around with scaling, shifting, and camera position, the picture might look like Figure 5-25, unveiling the deficiencies of this automatic texture mapping.

The other modes of the module `texture_coord_gen` act differently. With the `EYE_LINEAR` mode used, the a, b, c, d, e, f values, which you can change in the `parameter_s` and `parameter_t` anchors of the module, are projected according to the camera or eye position. Even then, the projection is not correct at the edges of the sphere, as shown in Figure 5-26.



Figure 5-26. Automatic texture coordinates, take 3

The `NORMAL_MAP` mode at first sight seems to do the correct thing, as shown in Figure 5-27. It uses the normal vectors of each mesh fragment to span the corresponding part of the texture. The map is glued to the camera position though, i.e., it is not fixed to the object. In fact you'll see the lighting seemingly rotate, which comes from the sphere being rotation invariant.



Figure 5-27. Automatic texture coordinates, take 4

The SPHERE_MAP and REFLECTION_MAP modes are corner cases we do not cover here. Of course, you still can play around with them. If you want to know more about those, look at the OpenGL specification, which describes the `glTexGen` function in detail.

You can also project the texture on the surface of a cube. Just replace the `mesh_sphere_octahedron` module with the `mesh_box` module in the same module folder. You might guess that the deficiencies we faced in the case of a sphere with an unnatural distortion at the edge for the OBJECT_LINEAR mode might disappear, and that is true. But only if we will be looking at a single one of the six faces, and will not be allowing the mapping to reach the edges. Why is that? Well, at the edges the mapping formula stops advancing and the adjacent faces will be filled with the pixels from the edge, as shown in Figure 5-28.



Figure 5-28. Automatic texture coordinates on a cube

But if you change the repetition mode inside `tex_*` parameters to “clamp” for both coordinates, and then scale and shift the texture appropriately, you might end up with a realistic texture mapping on *one* face of a cube, as shown in Figure 5-29.

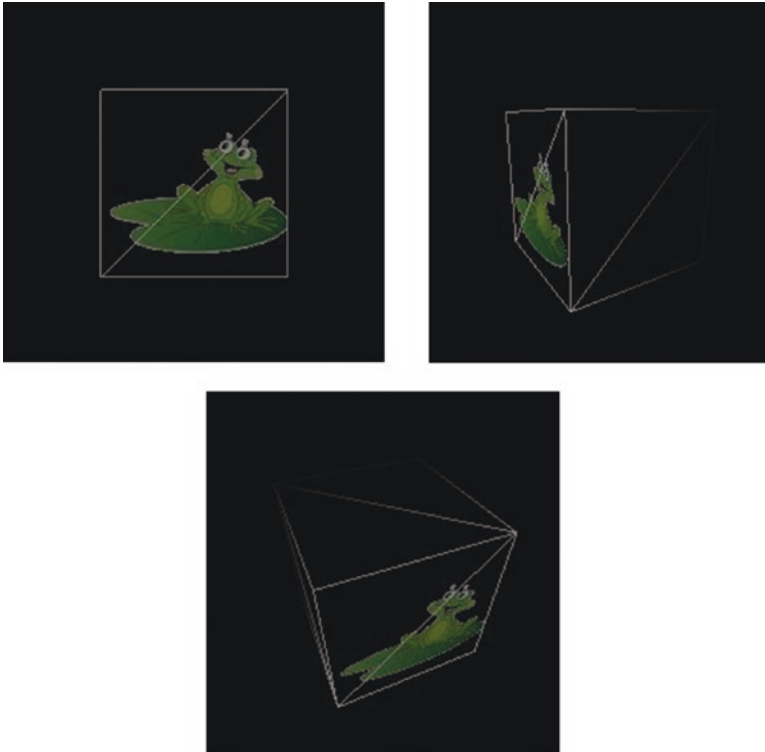


Figure 5-29. Automatic texture coordinates on a cube, take 2
For illustration purposes, a wireframe has been added to show the cube's orientation.

Having said all that, the `texture_coord_gen` module may work well with flat surfaces, but it shows a lack in physical correctness for bend surfaces. In any case, it can still be of help for effects you have in mind.

Floating Textures I

Another technique, which I call *floating textures*, uses a plane mesh generator. There are two:

- Mesh → solid → mesh_planes
- Mesh → solid → mesh_grid

The first is used to generate an arbitrary number of stacked slides filled with a texture. This could give impressive effects, but you can also set the number to just one and thus get a single slide hovering in 3D space. This hovering or floating in 3D space is where the name *floating texture* comes from. The second generator, `mesh_grid`, does not have the capability of stacking, so it will always be a single one. However, contrary to the `mesh_planes`, it consists of a grid of vertices evenly mapped to texture coordinates.

■ **Note** The source is available under A-5.2.2_Floating_texture inside the TheArtOfAudioVisualization folder.

We first describe the `mesh_planes` component. To use it, place the usual 3D chain on the canvas—`orbit_camera`, `light_directional`, `material_param`, `backface_culling`—and connect the camera to the screen module. Add these modules:

- Texture → loaders → `png_tex_load`
- Texture → modifiers → `scale`
- Texture → modifiers → `tex_parameters`
- Texture → modifiers → `translate`
- Renderers → mesh → `mesh_basic_render`
- Mesh → solid → `mesh_planes`

Connect them all, as shown in Figure 5-30. To prepare for the anchor parameters, place a PNG image inside the `[HOME]/thmad/[VERSION]/data/resources` folder.

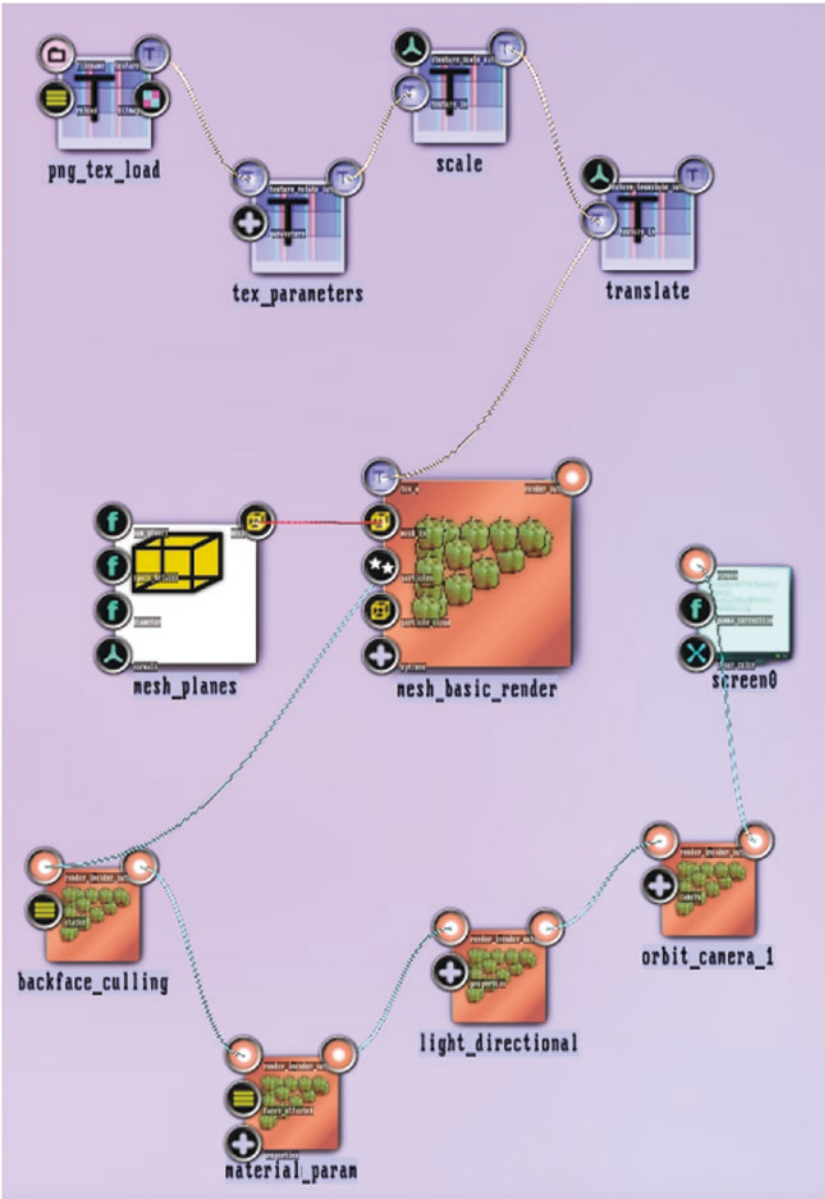


Figure 5-30. Floating textures modules connected

Note that `backface_culling` is connected to the `render_out` anchor of `mesh_basic_render`.

Make sure it has a size 128x128 or 256x256, and let its background be transparent. Having the texture size the same in both dimensions and a power of two is generally a good idea. The little frog that I'm going to use is already there inside the `TheArtOfAudioVisualization` folder; it's called `frosch2.png`.

The anchor parameters are shown in Tables 5-35 to 5-39.

Table 5-35.

texture → **loaders** → **png_tex_load**

| | | |
|----------|--------------------------|--|
| filename | PNG image from resources | Selection via resource browser; double-click on your choice. |
|----------|--------------------------|--|

Table 5-36.

texture → **modifiers** → **scale**

| | | |
|--------------|---------|--|
| scale_vector | 1; 1; 1 | The mapped size of the texture—you can later play with those values. |
|--------------|---------|--|

Table 5-37.

texture → **modifiers** → **translate**

| | | |
|--------------------|---------|--|
| translation_vector | 0; 0; 0 | The mapped position of the texture—you can later play with those values. |
|--------------------|---------|--|

Table 5-38.

texture → **modifiers** → **tex_parameter**

| | | |
|---------------------|-------|---|
| parameters / wrap_s | clamp | Do not repeat texture while mapping it. |
| parameters / wrap_t | clamp | |

Table 5-39.

| mesh → solid → mesh_planes | | |
|----------------------------|-----------|--|
| num_planes | 1.0 | Smoothness, change this one prior to the subdivision_level! |
| normals | 0; 0; 1.0 | Does not affect the slide position or rotation, but helps correctly render the slide, depending on context like lighting or correspondence with other objects. |
| diameter | 1.0 | The size. You can later play around with it. |

In your backend, make sure the anchor status of `backface_culling` is switched to `ENABLED`, the light is switched on at `properties / enabled`, and the perspective correction is switched on in the camera module. Also, you can set the camera rotation parameter to `(0.0; 0.0; 1.0)`. The result so far will look like Figure 5-31.

**Figure 5-31.** *Floating texture output*

What we have now is a size 2x2 slide in the x-y-plane going through $z=0$. If you want to have a different size, you can change the `diameter` anchor inside `mesh_plane`. In order to move the slide to another place or rotate it, you can easily achieve that via the two additional modifiers:

- Renderers → `opengl_modifiers` → `gl_translate`
- Renderers → `opengl_modifiers` → `gl_rotate`

Between `mesh_basic_render` and `backface_culling`.

If you look at the rendering pipeline so far, you might notice that there is an identifiable sub-pipeline, which could be reused for different texture files and different sizes, locations, and orientation. See Figure 5-32.

For reusable sub-pipelines, there exists a technique inside ThMAD, called *macros*. To make a macro, right-click somewhere at an empty spot of the canvas and choose Create Macro from the pop-up menu. Then open the macro. Right-click it and select Open from the pop-up menu. Now select all the modules from the sub-pipeline shown in Figure 5-32.

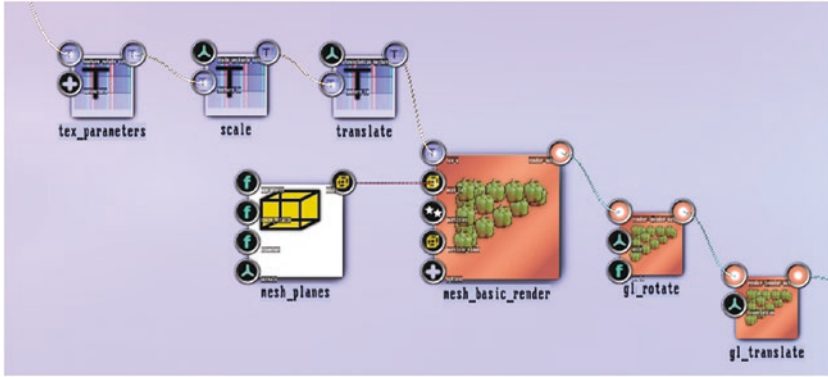


Figure 5-32. Floating texture sub-pipeline

Click on all these modules while holding down the Ctrl key while clicking. Or make sure all other modules are farther away from them, and then use the mouse to draw a selection rectangle over them, again while holding down the Ctrl key.¹

Now press the Shift and the Ctrl keys, click on any of the selected modules, keep the mouse clicked and drag into the macro. The result will look like Figure 5-33.

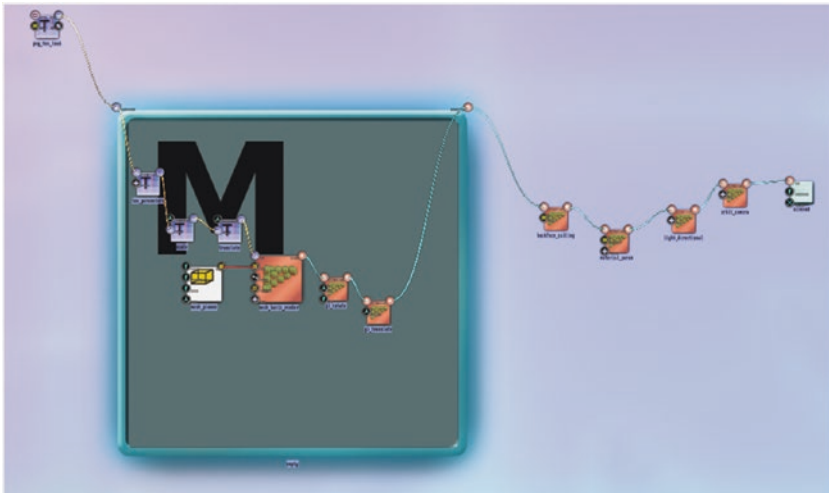


Figure 5-33. Moving modules to a macro

¹Press and hold down the Ctrl key. Click on the left upper corner of a rectangle containing the sub-pipeline modules, hold down the mouse key, and move the cursor to a suitable bottom-right point. Then release the mouse and the Ctrl keys.

Give the new macro a suitable name. Click on it, and in the pop-up menu that appears on the top-right of the ThMAD window, enter a new name, such as `FloatingTexture`. See Figure 5-34.



Figure 5-34. Naming the macro

After clicking on the Rename button, the new name will immediately appear under the macro module.

Because there are connections from or to outside modules, the macro automatically got one ingoing and one outgoing anchor assigned. The macro however would be of limited use if we couldn't establish more anchors, more precisely the size, position, and rotation. To accomplish that, from `mesh_planes`, drag the diameter anchor to an empty spot of the macro. Click on the anchor, and while holding the mouse button down, move it to an empty spot of the macro. A new anchor on the left side of the macro will appear, and it is connected to the diameter anchor of `mesh_planes`. Repeat the same process for the axis and angle anchors of `gl_rotate`, and the translation anchor of `gl_translate`. In the end you will have a macro with five input anchors, as shown in Figure 5-35.

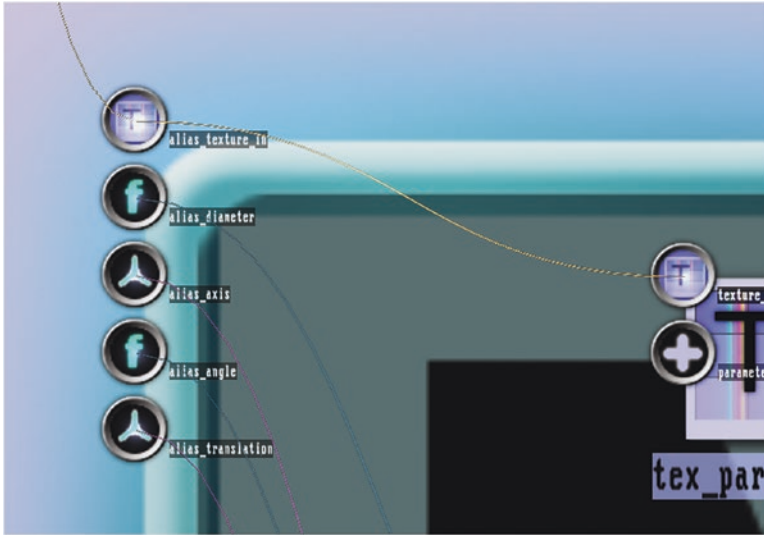


Figure 5-35. Macro with all input anchors You can rename the new anchors, if you like. Click on any of them and use the pop-up to rename it.

Right-clicking on the macro, using an empty spot, and choosing Close from the pop-up menu, will not remove the macro, but will minimize it. You can then treat it as kind of a black box. See Figure 5-36.



Figure 5-36. New state with a minimized macro

Macros can be saved independently from the state. Right-click on a macro and choose Save Macro. Enter a suitable name, such as FloatingTexture. The macro is now saved and can be used in other states, or multiple times in any state, including the current one. To use it immediately a second time in your current state, save it via Ctrl+S after moving the mouse to an empty spot of the canvas, then restart ThMAD. Open the Macro section from the module menu. The restart is necessary, since the current implementation of the module list browser is not reacting dynamically to structure changes.

Floating Textures II

For the other floating texture capable module, Mesh → solid → mesh_grid, use whatever method works for you in and replace the mesh_planes module with mesh_grid.

■ **Note** Samples in this subsection are available under A-5.2.3_Floating_texture* inside the TheArtOfAudioVisualization folder.

Use the values shown in Table 5-40 as parameters.

Table 5-40.

| mesh → solid → mesh_grid | | |
|--------------------------|-----|---|
| power_of_two_size | 5.0 | Subdivision of grid coordinate space (0,0) – (1,1) into $25 \cdot 25 = 32 \times 32 = 1024$ pieces. |
| plane | XY | Otherwise The camera looking at the x-y plane will not work. Z will be automatically 0.0 |

Everything else that worked earlier will work with that change as well. There is an important difference, though: the mesh_grid module uses a single planar grid of $n \cdot n$ vertices inside the quad $(-0.5, -0.5, 0) - (0.5, 0.5, 0)$ and linearly maps a given texture to all the vertices. Up to now, this does not make a difference, because all the grid's pieces are correctly mapped to the corresponding pieces of the texture. But we can mess around with the texture coordinates for a distortion effect. To see an example of that, delete the connector between the mesh_grid and the mesh_basic_render module, place a Mesh → texture → mesh_tex_sequ_distort between them, and connect all three, as shown in Figure 5-37.

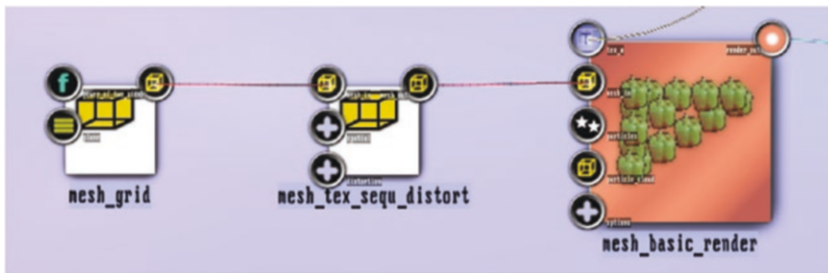


Figure 5-37. Mesh texture distortion

The `mesh_tex_sequ_distort` has a spatial complex anchor with `scale_u`, `scale_v`, `translate_u`, and `translate_v` sub-anchors. Set the values to read, as shown in Table 5-41.

Table 5-41.

| mesh → texture → mesh_tex_sequ_distort | | |
|--|-----------|---|
| spatial / scale_u | 1.3 | A scaling to apply to the texture coordinates. |
| spatial / scale_v | 1.5 | |
| spatial / translate_u | 0 | A translation to apply to the texture coordinates. |
| spatial / translate_v | 0 | |
| distortion / u_shape | See below | Shapes the distortion curve, for details see later in this chapter. |
| distortion / v_shape | See below | |

You can later play with all these values to fine-tune the texture mapping. Note that inside the distortion module, the translation happens *after* the scaling.

The other complex anchor, `distortion`, contains two sequences for the distortions along the x-axis and along the y-axis. By default these sequences map the range 0→1 linearly to 0→1, meaning there is no distortion aside from the scaling and translation. In order to change this, see the detailed description in Chapter 6. After you have made something like Figure 5-38, the output might look like Figure 5-39.

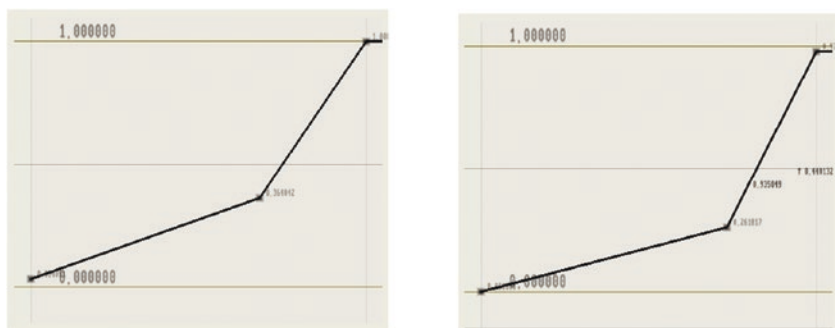


Figure 5-38. Distortion sequences

Left for the *u*-axis, and right for the *v*-axis.



Figure 5-39. Floating texture with distortion
On the left of the undistorted image.

The distortions shown in Figure 5-38 have a positive slope everywhere; you can also have parts with a negative slope which is like *going back* in the texture while rendering, and you can also simulate a mirror that way, as shown in Figures 5-40 and 5-41.

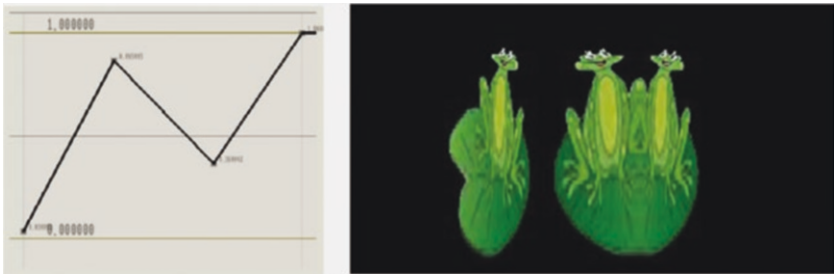


Figure 5-40. Texture distortion (*u*-axis) with negative slope



Figure 5-41. Texture distortion (*u*-axis) simulating a mirror

Blobs, Blobs, Blobs

Blobs are the workhorses of ThMAD—you can use them wherever something simple needs to be drawn. And don't be misled by the word “simple”—with all those features that ThMAD offers, you can get really interesting sketches done by just using blobs.

Blobs can be round, stars, or leaves, depending on the parameters. Technical details can be found in the reference at Bitmaps → Generators → Blob for for generating blob bitmaps and at Texture → Particles → Blob for generating blob textures that go directly to the graphics hardware for faster processing.

In this section, we describe different usage scenarios. The possibilities are endless, so go ahead and try your own ideas as well.

Basic Blobs

Basic blobs are single figures sent to a bitmap or a texture. This subsection is about bitmaps.

■ **Note** The samples of this section are available under A-5.3.1_* inside the TheArtOfAudioVisualization folder.

Start with an empty canvas. Right-click on any empty spot and choose New → Empty Project. Place these modules on the canvas and connect them all and the textured_ - rectangle to the screen module:

- Bitmaps → generators → blob
- Texture → loaders → bitmap2texture
- Renderers → basic → textured_rectangle

Set the parameters as shown in Table 5-42.

Table 5-42.

bitmaps → generators → blob

| | | |
|-------|--------------------|---|
| color | 1.0; 1.0; 1.0; 1.0 | White |
| alpha | yes | If set to yes, the shape of the blob is defined by each pixel's ALPHA value set appropriately. Otherwise, the ALPHA is set to 1.0 and the shape is defined by calculated color values on a per-pixel basis. |
| size | 512x512 | |

We could use the other blob module texture → particles → blob as well, but first having it as a bitmap allows for more variations from the bitmap modifying modules. More about that later.

Now, playing with the arms, attenuation, and star_flower anchors gives you different blob shapes, as shown in Figure 5-42. The corresponding state is shown in Figure 5-43.

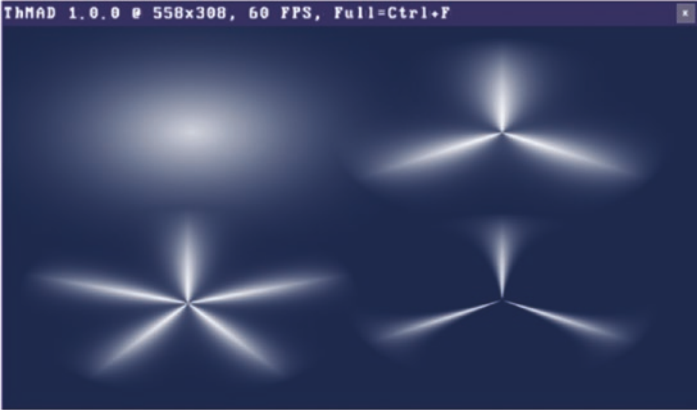


Figure 5-42. Blobs in different variations

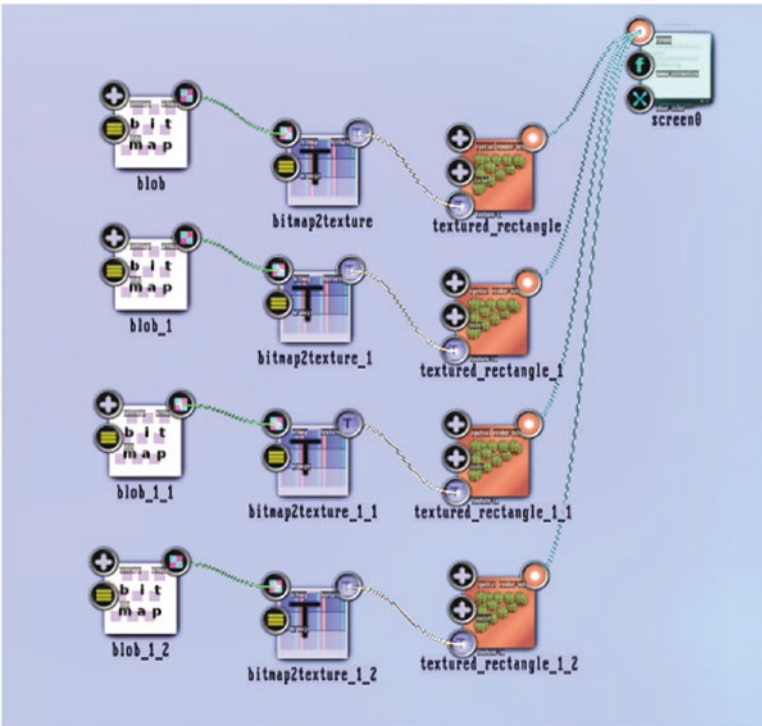
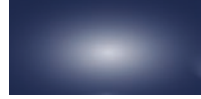


Figure 5-43. Blobs in different variations state

The distinct parameters for each sub-pipeline of this state are shown in Tables 5-43 to 5-46.

Table 5-43.

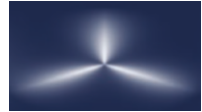
bitmaps → generators → blob



| | |
|--------------------|-----------|
| color, alpha, size | See above |
| arms | 0 |
| attenuation | 1.9 |
| star_flower | 0.12 |

Table 5-44.

bitmaps → generators → blob



| | |
|--------------------|-----------|
| color, alpha, size | See above |
| arms | 3.0 |
| attenuation | 2.14 |
| star_flower | 0.5 |

Table 5-45.

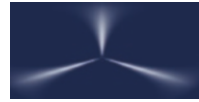
bitmaps → generators → blob



| | |
|--------------------|-----------|
| color, alpha, size | See above |
| arms | 5.0 |
| attenuation | 2.14 |
| star_flower | 0.5 |

Table 5-46.

bitmaps → generators → blob



| | |
|--------------------|-----------|
| color, alpha, size | See above |
| arms | 3.0 |
| attenuation | 2.0 |
| star_flower | 3.9 |

Because blobs are optimized for performance, they are good candidates for particle systems, where many small objects are needed. Particle systems are covered in Chapter 6.

The basic building blocks for the good performance blobs consist of storing and handling them inside graphics hardware. This is why there is nothing like a direct blob renderer, but instead it's called a blob texture generator. For the same reason, you do not usually add sound responsiveness directly to blobs. Rather what you do is add sound responsiveness at places in your states where you deal with textures, such as texture transformation, texture mapping, or later when it comes to composing textures.

Nevertheless, having that in mind, you are not forbidden to add sound input to blob parameters. Just have in mind that your mileage may vary depending on the power of your system.

Try the following. Using the previous system, remove all but the first blob branch, then add these modules and connect them as shown in Figure 5-44:

- Maths → arithmetics → ternary → float → mult_add, twice
- Sound → input_visualization_listener

The parameters use the values shown in Tables 5-47 and 5-48.

Table 5-47.

maths → arithmetics → ternary → float → mult_add

The one connected to module textured_rectangle

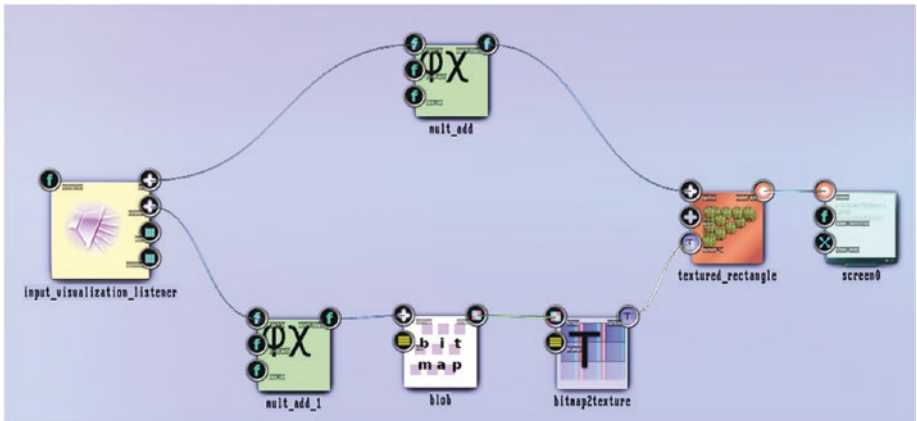
| | |
|------------|-----|
| first_mult | 1.0 |
| then_add | 5.0 |

Table 5-48.

maths → arithmetics → ternary → float → mult_add

The one connected to module blob

| | |
|------------|------|
| first_mult | 20.0 |
| then_add | 0.0 |

**Figure 5-44.** Blobs with sound awareness, state

The upper `mult_add` is connected to the `spatial / size` anchor of `textured_rectangle`, and to the `vu / vu_1` anchor of `input_visualization_listener`. The lower `mult_add` is connected to the `arms` anchor of module `blob`, and to the `octaves / left / octaves_left_0` anchor of the `input_visualization_listener`.

Start the sound input and you will see the blob changing its size and shape.

Perlin Noise Blobs

Perlin noise is an algorithm that greatly improves the natural look of 2D or 3D scenes with undulating parts, like surface structures, water waves, fog, or fire. Apart from using it inside scenes, ThMAD enables you to add Perlin noise to blobs. You will see here how this can be accomplished.

■ **Note** The samples of this section are available under `A-5.3.2_*` inside the `TheArtOfAudioVisualization` folder.

Create an empty state by right-clicking an empty spot and then choose New → Empty Project.

Then place the following modules on the canvas:

- Renderers → basic → textured_rectangle
- Texture → loaders → bitmap2texture
- Bitmaps → generators → perlin_noise
- Maths → oscillators → oscillator

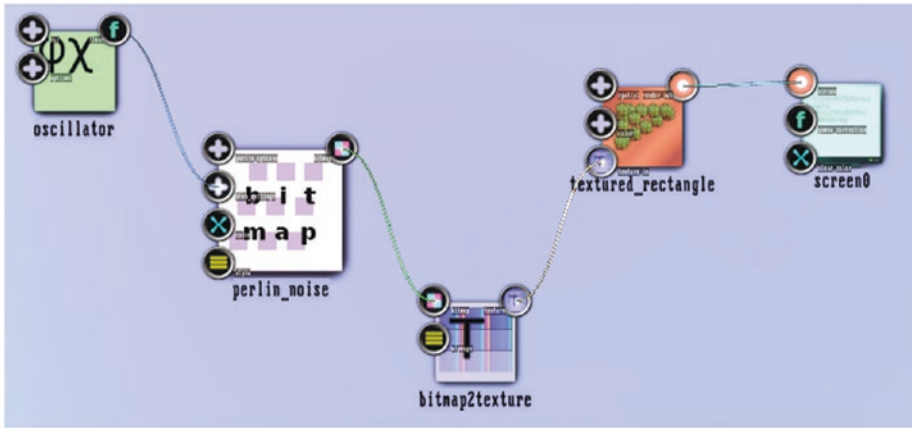


Figure 5-45. Perlin noise blobs state

The oscillator is connected to blob_settings/angle.

Connect them all, as shown in Figure 5-45. Set the parameters as shown in Tables 5-49 to 5-51.

Table 5-49.

| screen0 | |
|-------------|-----------------------|
| clear_color | 0.08; 0.15; 0.31; 1.0 |

Table 5-50.

| bitmaps → generators → perlin_noise | | |
|-------------------------------------|-----|---|
| perlin_options / rand_seed | 4.0 | Change to alter the shape of the noise. |
| perlin_options / perlin_strength | 1.0 | Amount of noise - 0 means no noise (= plain blob) |
| perlin_options / octave | 7 | Noise grain level, 1 = very coarse, 16 = very detailed calculation |
| perlin_options / frequency | 4 | Noise scale |
| blob_settings / enable_blob | yes | |
| blob_settings / arms | 3.0 | |
| blob_settings / attenuation | 1.0 | |
| blob_settings / star_flower | 0.0 | |
| blob_settings / angle | - | Connected to oscillator |

Table 5-51.

| maths → oscillators → oscillator | |
|----------------------------------|--------|
| osc / osc_type | saw |
| osc / freq | 1.0 |
| osc / amp | 3.1415 |

■ **Caution** Usually for performance reasons you do not add dynamics to a bitmap that is then going to be transported as a texture to graphics hardware. For illustration purposes, you can of course do that, or also if you know what you are doing.

The output is shown in Figure 5-46.



Figure 5-46. Perlin noise blobs

Summary

This chapter contained a collection of tutorials (or stories) covering many aspects of ThMAD and showing its capabilities. You learned several 3D techniques, including transformation, special graphics rendering modes, and a very special module named ocean. The chapter also introduced textures and blobs.

While this chapter more or less focused on basic concepts and techniques, Chapter 6 is a collection of more advanced stories.

CHAPTER 6



Stories: Advanced Level

In this chapter, we dive more into the art of audio visualization with more elaborated walk-throughs covering various aspects. We will by no means be able to describe all possible kinds of sketches, but it is the aim of this chapter to provide you with techniques, which eventually will be apt to make you realize your own ideas. There is no particular order for the sections, so you can sequentially work through them or cross-read them—whatever best suits you.

Backfeeding Textures

Textures can be described as 2D ordered sets of colors stored on the graphics hardware. They can be scissored freely and stretched, rotated, translated, or otherwise transformed and in a final step rendered to a collection of freely shaped polygons of the output screen.

You can even go further and misuse the texture colors for anything you might think of, because inside shader programs, which run on the graphics hardware themselves, we can do quite a lot with these four-color values at each texture coordinate. With that in mind, clever people readily realized that the usage scenarios of textures cover more than just the spanning of pictures over 3D objects.

ThMAD provides for an extremely powerful and versatile capability of backfeeding rendering data into an earlier stage of the rendering pipeline. This is possible, because image data can be copied to a texture and this texture can be used for modules that need a texture as an input. The module that allows for this backfeeding is called `Texture → Buffers → render_surface_single`. For a technical description of this module, see Chapter 8.

Blurring in Two Dimensions

Blurring makes visualizations appear fancier and makes them more interesting. The objective here is not to make objects unsharp, but to introduce some shininess to them, or to create special effects.

■ **Note** Various states shown here are as sources available under `A-6.1.1_Blurring_*` inside the `TheArtOfAudioVisualization` folder.

For now, we stick to seeing the texture color values as colors, and we want to use this backfeeding technique for a blurring effect.

A note of caution: Backfeeding happens at frame rate, which means 60 times a second. This makes states using backfeeding extremely sensitive to the parameters. A minuscule change of some of them might show tremendous effects, or it might lead to a boring or black or white screen and you being a disappointed user needing hours of readjusting to make anything meaningful appear again. For that reason, I try to be careful with explaining everything needed, to help you in achieving a satisfactory if not a good or very good experience. Of course, experimenting with changes to the parameters will in the end give you a feeling about that brittleness, but as a rule of thumb, you can make such changes happen on a scale at least ten times smaller than what you might be used to with respect to what you have seen in ThMAD before.

We start by rendering a rotating rectangle on the screen. To do so, open a clean canvas. If not already empty by chance, right-click on an empty spot, then choose New → Empty Project. Then choose Save As.... The latter is to ensure you do not unintentionally overwrite previous work. Then choose Renderers → Basic → colored_rectangle on the screen. For the parameters, see Table 6-1.

Table 6-1.

| renderers → basic → colored_rectangle | | |
|---------------------------------------|------------------|---|
| spatial/ size | 0.77, 0.11, 0.0 | |
| color | 0.2, 0.4, 0.9, 1 | The fourth value is the ALPHA value, or the <i>opacity</i> . Here, it's 1.0. With ALPHA less than 1.0 some of the other values needed to be adjusted to achieve a noticeable blurring effect. Other than that choose at will. |

Place a module by choosing Renderers → opengl_modifiers → gl_rotate next to it. Connect it to the colored_rectangle module.

Place a module by choosing Maths → oscillators → oscillator next to it. Connect the oscillator's float anchor with the angle anchor of gl_rotate. For the correct parameters, see Table 6-2.

Table 6-2.

| maths → oscillators → oscillator | |
|----------------------------------|------|
| osc/ freq | 0.10 |

Place a module Renderers → opengl_modifiers → blend_mode next to it. Connect the blend_mode module with the gl_rotate module. For parameters, see Table 6-3.

Table 6-3.

| renderers → opengl_modifiers → blend_mode | |
|---|-----------|
| source_blend | SRC_ALPHA |
| dest_blend | ONE |

The oscillator and the blend-mode modules are new at this point of the book. While you can find the technical details in Chapter 8, it is worthwhile to mention a few things here. The oscillator spends an oscillating float value, which we use here to change the angle of the rectangle drawn. This is important here to actually see the blurring effect fully at work. While blurring works for static objects, too, the effect will be much more noticeable for moving objects. Even more important, however, is the blending mode module. We need it for mixing, or *blending*, the original pixel data coming from the rectangle module, with the backfeeding from the blurring sub-pipeline, which we describe and add soon. If you choose the wrong blending modes, you will see the wrong thing or maybe just nothing at all. Why we choose SRC_ALPHA and ONE as just said, will be explained a little later. For now, we continue with the state description.

Connect the blend_mode module with the screen module. Now you should see the rotating rectangle, as shown in Figure 6-1.

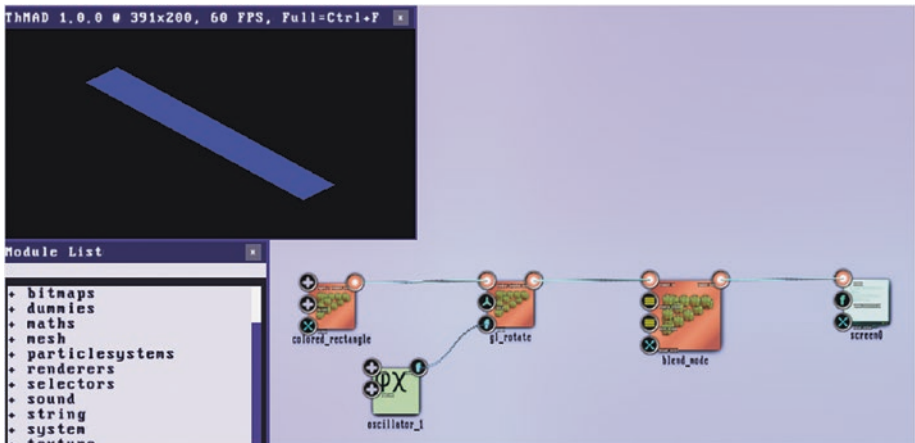


Figure 6-1. Blurring rectangle part I

Next we commence with the backfeeding renderer. To do so, place the module Mesh → solid → mesh_grid on the canvas. This will later allow you to project the backfeeding texture onto a planar rectangular grid. Why we use a grid here instead of a simple rectangle stems from the fact that a grid will provide more interesting distortion effects. More about that later. For the parameters, see Table 6-4.

Table 6-4.

| | |
|---------------------------------|----|
| mesh → solid → mesh_grid | |
| plane | xy |

1. Next to it, place the module Mesh → texture → mesh_text_sequ_distort and connect them. This will later allow us to add distortion. For now, leave the parameters as they are.
2. Next to it, place the module Renderers → mesh → mesh_basic_render and connect the output anchor from mesh_text_sequ_distort to its input anchor mesh_in. This module signs responsible for rendering the mesh_grid mesh. Leave all its parameters at their default.
3. Next to it, place the module Renderers → opengl_modifiers → gl_scale_one and connect it to the output of mesh_basic_render. This introduces a zooming, or scaling, which is actually quite important. For blurring it is necessary to have a close match of the sizes of the rendered graphics' size and the backfed graphics' size. Here we need a scaling of 2.0, to be set at the scale anchor. See Table 6-5. The factor of two comes from the difference between the colored_rectangle module, which paints into a 2 x 2 size rectangle, and the texture coordinate space extent, which reads 1 x 1.

Table 6-5.

| | |
|--|-----|
| renderers → opengl_modifiers → gl_scale_one | |
| scale | 2.0 |

4. Connect the output anchor of `gl_scale_one` to the input anchor `render_in` of the `blend_mode` module. See Figure 6-2.

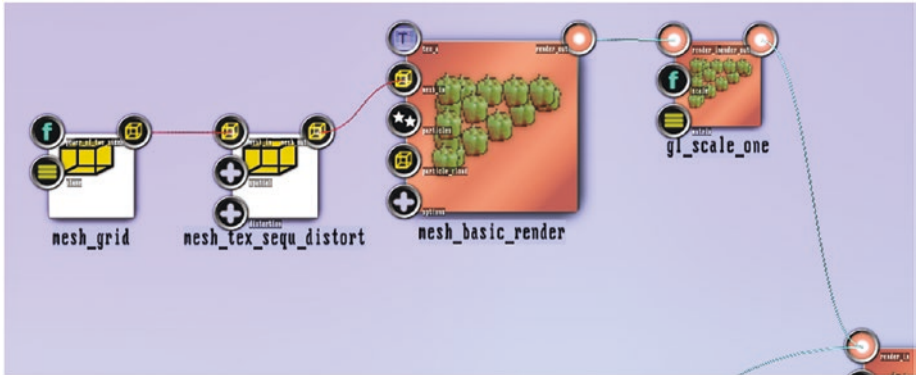


Figure 6-2. *Blurring rectangle part II*

5. This is the point where we introduce the backfeeding module. It is called `texture → buffers → render_surface_single` and will allow for sending the graphical data to a texture instead of a screen output. Internally two textures are getting used to copy the double buffering behavior of OpenGL. The latter means OpenGL rendering happens into a hidden pixel data buffer and only after everything has been rendered a fast copy to the physical screen will occur. This is to avoid flickering. Place the module somewhere near the `blend_mode` and draw a *second* connection from the `blend_mode` module's output to the `render_surface_single` module's `render_in` input anchor.
6. Quite some modules allow for their output being multiplexed—in this case, we render the output from `blend_mode` to *both* the screen and the texture buffer. Make sure the `support_feedback` sub-anchor of the `options` complex anchor is set to “yes”. See Table 6-6.

Table 6-6.

| texture → buffers → render_surface_single | | |
|--|---------------|---|
| options / support_feedback | yes | |
| options / texture_size | VIEWPORT_SIZE | |
| options / support_feedback | yes | Important for the backfeeding to work. |
| options / clear_color | 0, 0, 0, 0 | If not all zero, background will not stay black |

7. After the `render_surface_single` place the module `texture → effects → highblur` and connect them both. Set anchors of `highblur`, as seen in Table 6-7.

Table 6-7.

| texture → effects → highblur | | |
|-------------------------------------|----------------|---|
| translation | 0.43, -0.65, 0 | Introduced a translational blurring, like wind |
| blowup_center | 0.5, 0.5, 0 | The point from where to blow apart |
| blowup_rate | 80.0 | The blow apart rate |
| attenuation | 200.0 | Tells about the rate at which the blurring decays |
| texture_size | VIEWPORT_SIZE | |

More details about this module can be found in Chapter 8.

Draw a connection from the `texture_out` anchor of `highblur` to the `tex_a` input anchor of `mesh_basic_render`. The backfeeding is actually complete at this stage, but there is one important thing worth mentioning: it is the order of incoming data into module `blend_mode`. It is absolutely necessary that the rendered rectangle's data arrive first at the blend mode module, and the backfeeding data second. Otherwise, no blurring effect will happen. To make sure the order is correct, double-click on the input anchor of `blend_mode` and in case the order is wrong, so change it via clicking and dragging one of the anchors. See Figure 6-3.

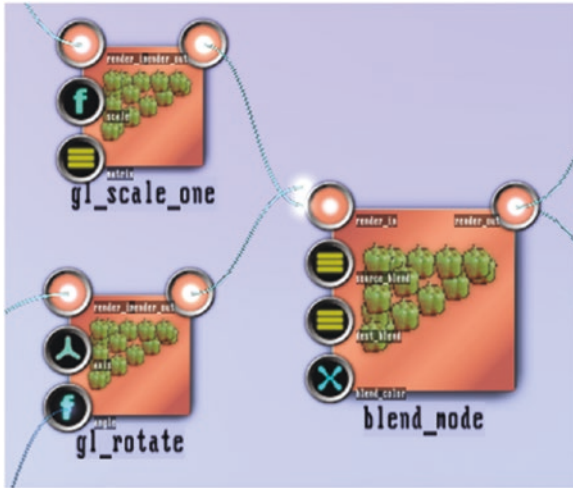


Figure 6-3. Blurring rectangle part III

Checking and fixing the blend order. The `gl_rotate` comes from the rectangle's renderer, the `gl_scale_one` from the backfeeder renderer. The input anchor of `blend_node` has been opened via double-click and the order of the connections can be changed via clicking and dragging on one of the anchors.

■ **Caution** Input order when several incoming connections to a graphical (renderer) exist can be critical, since the order is only visible after double-clicking on the anchor. So have this in mind when your pipeline does not work as intended.

If everything is set up correctly, the rotating rectangle will be blurred, as shown in Figure 6-4. The complete state is shown in Figure 6-5.

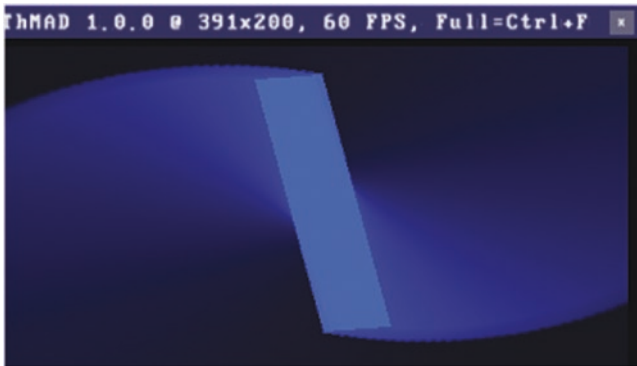


Figure 6-4. Blurring rectangle part IV. Final output

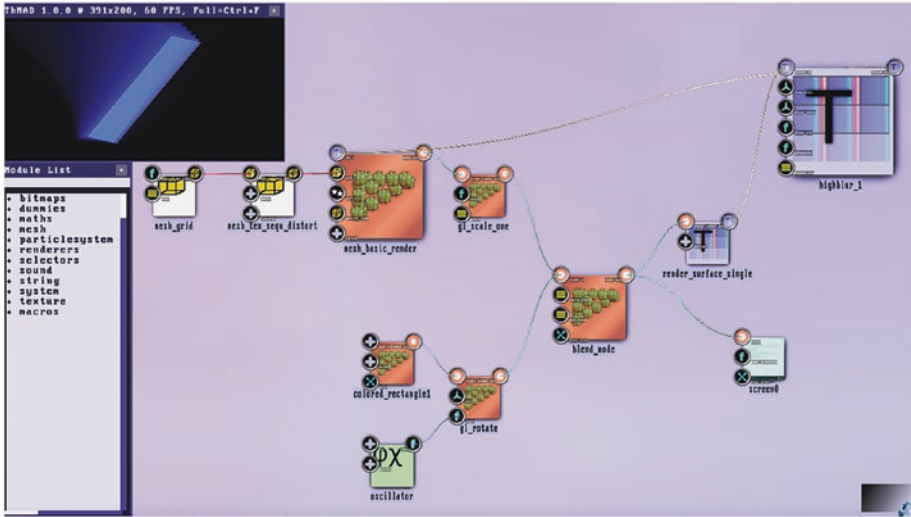


Figure 6-5. Blurring rectangle complete state

I told you the blending mode set in module `blend_mode` is important. A concise description of this module including all anchors is given inside the reference in Chapter 8, but an explanation is given already here. The `dest_blend` anchor tells about what is currently seen in the pixel buffer, and since the primary renderer of the rectangle is the first by input order, `dest_blend` describes the incoming solid and unblurred rectangle. The value `ONE` set here just means it goes unchanged into the blending. The second one, `source_blend`, is connected to what is drawn over what is already there and we set it here to `ONE`, meaning we just overlay them. Since the source by that definition comes from the blurring sub-pipeline, we will draw the unblurred data first and overlay the blurred data over it. Note that under circumstances both the involved color values and the blend mode, including ordering, are subject to change depending on your actual sketch.

To show the blurring effect for maybe something more interesting compared to a rectangle, and also to see how it can be tweaked via texture distortion, proceed as follows:

1. Cut the connection from `colored_rectangle`: right-click on the connection line and choose `Disconnect`. Or delete the `colored_rectangle`, select it via right-click, then press the `DEL` key on your keyboard.
2. Add the module `Texture` → `particles` → `blob` and set its parameters listed in Table 6-8.

Table 6-8.

| texture → particles → blob | | |
|----------------------------|--------------------|---|
| settings / arms | 5 | |
| settings / attenuation | 7.0 | |
| settings / star_flower | 0.4 | |
| settings / color | 0.4, 0.7, 0.1, 1.0 | Any color you like, but not too bright to avoid over-saturation |
| size | VIEWPORT_SIZE | |

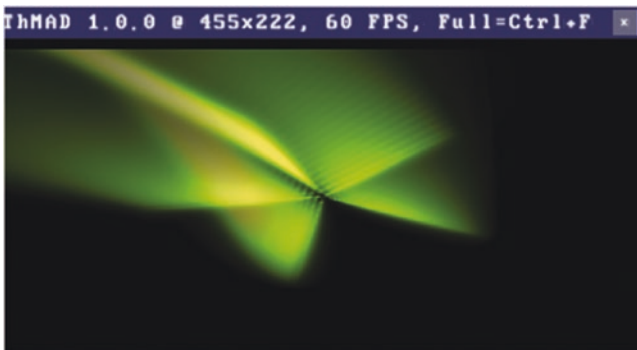
3. Add the module Renderers → basic → textured_rectangle and connect it to the blob. For its parameters, see Table 6-9.

Table 6-9.

| renderers → basic → textured_rectangle | |
|--|------|
| spatial / size | 0.6 |
| color / global_alpha | 0.95 |

4. Connect the textured_rectangle to the input of blend_mode. Again make sure the gl_rotate module's connection to it is above the gl_scale_one module's connection; see Figure 6-3.

The output will now look like Figure 6-6. Let's make that even more interesting via some dynamics to the blurring direction. To accomplish that, follow these steps:

**Figure 6-6.** Blurring star part I

5. Place the module Maths → converters → 3float_to_float3 close to highblur. Connect its output to the anchor translation.
6. Place another Maths → oscillators → oscillator next to it. Remember, we already have one for the rotation module. Inside the second oscillator, set the parameters as shown in Table 6-10.

Table 6-10.

| maths → oscillators → oscillator | |
|----------------------------------|------|
| osc / freq | 1.0 |
| osc / amp | 0.25 |

Connect it to both the floata and floatb input anchors of 3float_to_float3.

You should now have the blurring periodically changing its direction, as shown in Figure 6-7. As a last step, we add a distortion to the blurring:

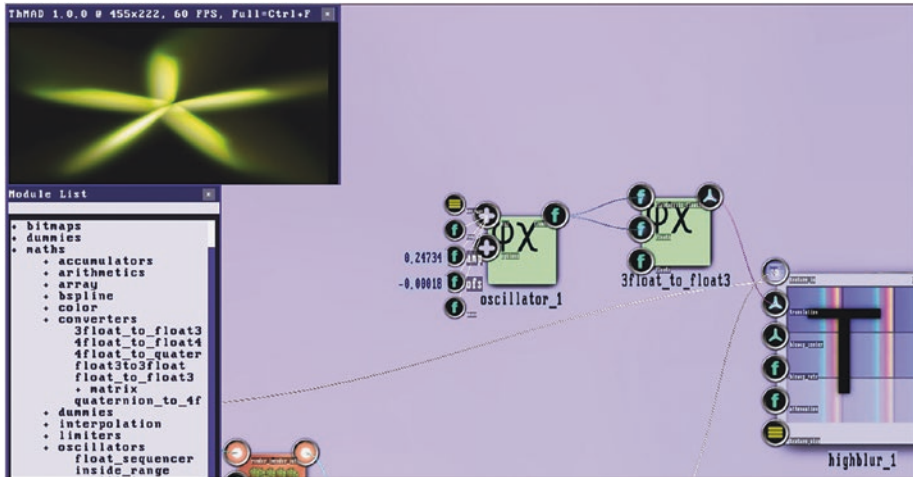


Figure 6-7. Blurring star part II

7. Change the osc_type sub-anchor of the oscillator near gl_rotate to “SAW”. Because of the oscillator internal time value going from 0.0 to 1.0 on the “angle” input anchor of gl_rotate being interpreted to map 1.0 to a full rotation, the saw shape effectively appears to create a never-ending rotation. See Table 6-11.

Table 6-11.**maths → oscillators → oscillator***Near gl_rotate*

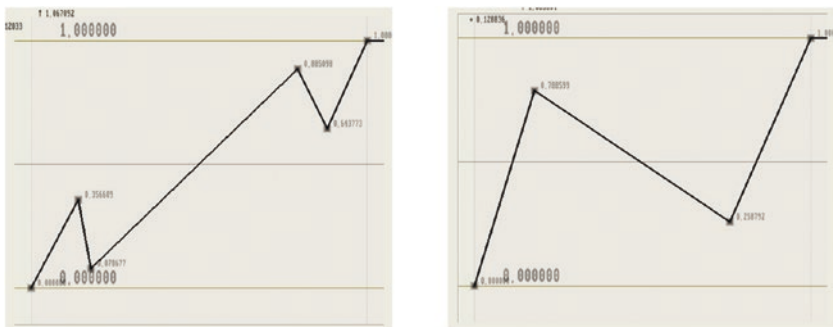
| | |
|------------|------|
| osc / freq | 0.10 |
| osc / type | SAW |

- Change the arms input anchor of blob to 6.0 to improve symmetrization. See Table 6-12.

Table 6-12.**texture → particles → blob**

| | | |
|------------------------|--------------------|---|
| settings / arms | 6 | |
| settings / | 7.0 | |
| attenuation | | |
| settings / star_flower | 0.4 | |
| settings / color | 0.4, 0.7, 0.1, 1.0 | Any color you like, but not too bright to avoid over-saturation |
| size | VIEWPORT_SIZE | |

- Open the “distortion” complex anchor of module `mesh_sequ_distort`. Open both sub-anchors `u_shape` and `v_shape` via double-click and add some value anchors inside the sequence editors: Shift-click on the line and afterward click and drag the value anchors, such as shown in Figure 6-8. See also Chapter 7.

**Figure 6-8.** Blurring star part III. Blurring distortion

The result of all that might look like Figure 6-9. A lot of other interesting effects may be produced altering anchor values of the module. The possibilities are essentially endless, so go ahead and play with the values at will. And of course you can easily add sound input to all of that as well. Just instantiate `input_visualization_listener` and use its output anchors to control anything you might think of.

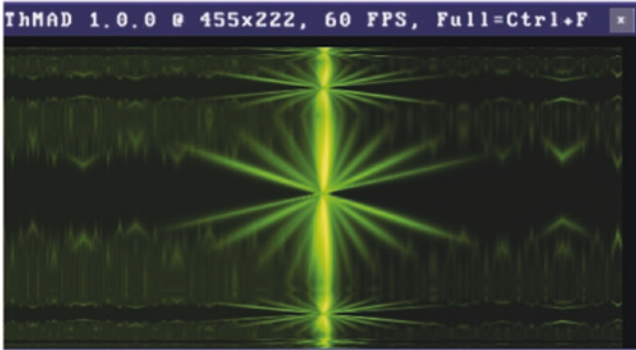


Figure 6-9. *Blurring star part IV. Blurred distortion output*

A nice variation of the distortion just shown consists of replacing the `mesh_tex_sequ_distort` module by `Mesh` → `texture` → `mesh_tex_bitmap_distort` with the additional module `Bitmaps` → `loaders` → `png_bitm_load` loading any PNG file from the `[HOME]/thmad/[VERSION]/data/resources` folder. See Figures 6-10 and 6-11. You can also let sound input control the rate of distortion.

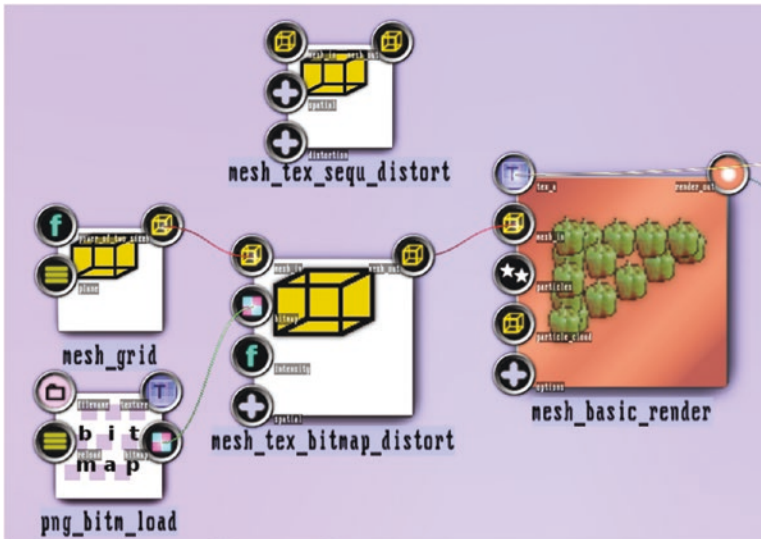


Figure 6-10. *Blurring distortion by bitmap, state changes*

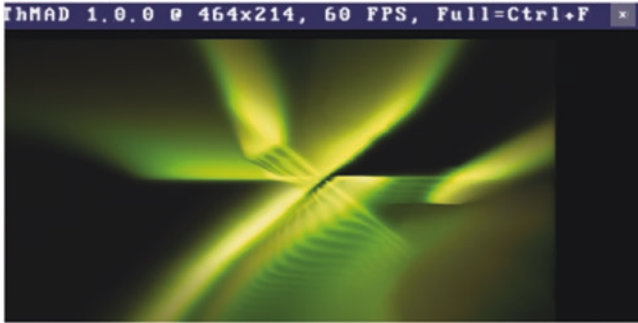


Figure 6-11. Blurring distortion by bitmap, output

■ **Note** A-6.1.1_Blurring_Star_E inside the TheArtOfAudioVisualization folder.

Self-Similarity

Self-similarity is an extremely important feature in both nature and arts. As for nature, chemical and biological processes are influenced by it, and in arts you will find it everywhere—in painting, architecture, music, poetry, photography, and design.

What exactly does self-similarity mean? It is the emerging of structures of the same or similar kind and shape appearing at different levels of analysis. Look at a tree, for example. Seeing from a far distance, you can see the trunk and an undefined head. Coming a little closer, or looking at a *level* deeper, we can see the major branches branching out of the trunk. Even closer and one more level deeper, we can see the smaller branches branching out of the major branches. Continuing that procedure, we can eventually see the leaves branching out of the smallest branches. So a tree inherently shows self-similarity; see also Figure 6-12. Or watch a snowflake, with branching at different levels just as well.

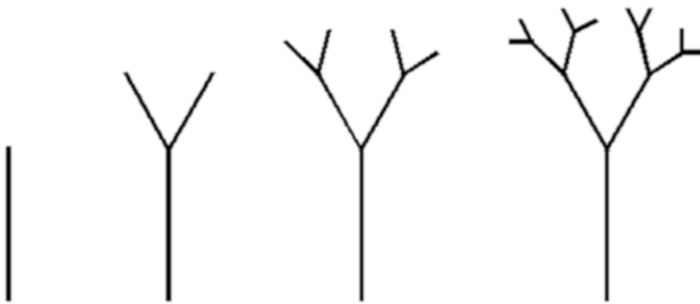


Figure 6-12. Self-similarity in trees

Both of them have artificial counterparts. Trees have a long history and Pythagoras constructed a fractal tree from a simple construction rule by just adding two smaller quads at a fixed angle to a given quad, and repeating that all over; see Figure 6-13.

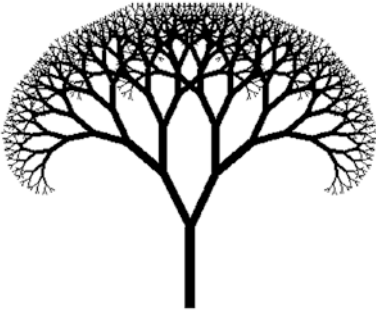


Figure 6-13. A Pythagoras tree

Each segment spreads into two segments of 75% the size of the basis segment, each at 50 degrees.

An artificial snowflake is given by the KOCH curve, which simply cuts a line into three equal pieces and replaces the middle piece with two sides of a triangle. This is perpetually repeated for the resulting lines. See Figure 6-14.

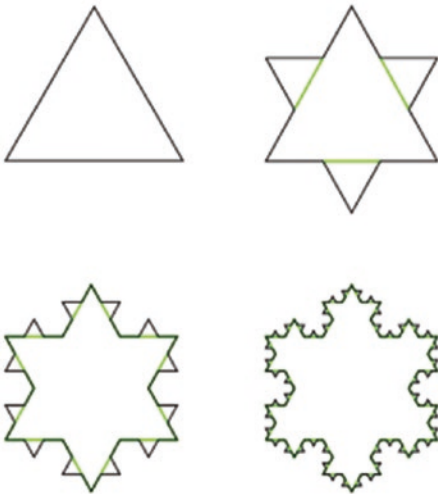


Figure 6-14. The KOCH curve, an artificial snowflake

As for ThMAD, we have two options for applying the concept of self-similarity. First we can take the contents of the frame buffer, i.e., what is shown on the screen, and backfeed it to some earlier stage of the rendering pipeline. Second, we can let some renderer use a fractal algorithm to do its work. As a correct follow-up, where we were backfeeding to achieve a blurring effect, we will for now concentrate on backfeeding for generating self-similarity. The other one, algorithmic self-similarity, will not be covered in this book.

We start with a basic self-similarity pipeline and from there continue making it fancier. After any previous work has been saved, clean the canvas by right-clicking and choosing New → Empty Project. Then immediately save it under a new name to make sure it will not mess up your saved files. Now place the following modules on the canvas:

- Bitmaps → generators → blob
- Texture → loaders → bitmap2texture
- Renderers → basic → textured_rectangle
- Renderers → opengl_modifiers → blend_mode

Connect them all and the `blend_mode` to the screen. See Figure 6-15. Inside the modules, set parameters as described in Tables 6-13 to 6-16.

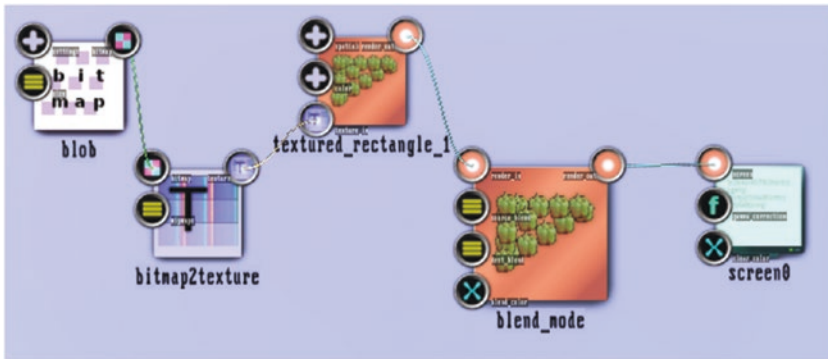


Figure 6-15. *Self-similarity main generator pipeline, blender and output*

Table 6-13.

bitmap → generators → blob

| | |
|------------------------|---------|
| settings / arms | 5 |
| settings / attenuation | 2.0 |
| settings / alpha | yes |
| size | 512x512 |

Table 6-14.

| renderers → basic → textured_rectangle | | |
|---|-----------------|---|
| spatial / position | -0.4; 0.27; 0.0 | If you are not yet used to the scrollbar controller, setting negative numbers seems to be tricky at first. The solution is to lower the knob value in the second row of knobs, and then change the slider. The first knob is the range, the second knob the offset. |
| spatial / size | 0.6 | |

Table 6-15.

| renderers → opengl_modifiers → blend_mode | | |
|--|---------------------|--|
| source_blend | SRC_ALPHA | Incoming pixels appear according to their ALPHA value |
| dest_blend | ONE_MINUS_SRC_ALPHA | Existing pixels are getting diminished by (1-ALPHA) of incoming pixels |

Table 6-16.

| output → screens → screen0 | | |
|-----------------------------------|------------|--------------------|
| clear_color | 0, 0, 0, 0 | Avoid any clearing |

You should now see a single star in the upper-left part of the output screen; see Figure 6-16. Now for the backfeeding sub-pipeline, place the modules on the canvas and draw the connections between them; see Figure 6-17.

- Mesh → solid → mesh_grid
- Renderers → mesh → mesh_basic_render
- Texture → buffers → render_surface_single



Figure 6-16. Self-Similarity main generator pipeline, output

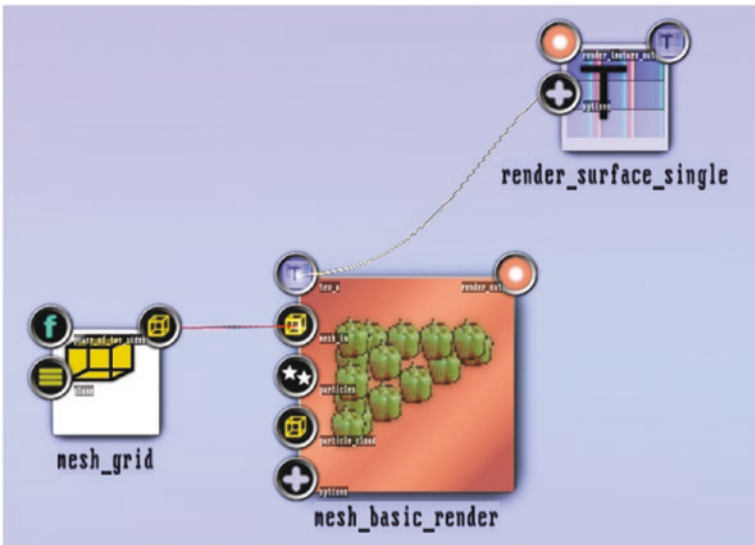


Figure 6-17. Self-Similarity backfeeding pipeline

Set the parameters as shown in Tables 6-17 and 6-18.

Table 6-17.

mesh → solid → mesh_grid

| | | |
|-------|----|-----------------------------|
| plane | xy | We need “xy” for 2D states. |
|-------|----|-----------------------------|

Table 6-18.

| texture → buffers → render_surface_single | | |
|---|------------|---|
| options / texture_size | 512x512 | |
| options / support_feedback | yes | Without “yes” backfeeding would not work. |
| options / alpha_channel | yes | |
| options / clear_color | 0, 0, 0, 1 | Fat black. Makes the backfed texture visible, for clearness. Later we can change ALPHA to 0 to mystify the output, see below. |

To combine the two sub-pipelines draw a connection from the output of `blend_mode` to the `render_in` anchor of `render_surface_single`, and another one from the output of `mesh_basic_render` to the `render_in` anchor of `blend_mode`. Double-click on the latter anchor and make sure the generator pipeline is on top of the backfeeding pipeline. If this is not the case, drag any of them vertically to the correct position. See Figure 6-18.

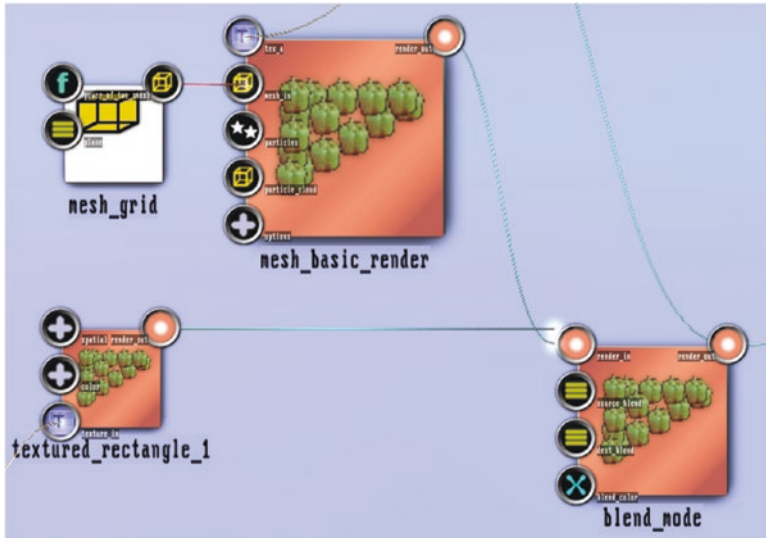


Figure 6-18. Combining main rendering and backfeeding pipeline

You should now see the output shown in Figure 6-19. This result obviously shows self-similarity: zooming into the picture using a factor $\frac{1}{2}$, $\frac{1}{2} \cdot \frac{1}{2}$, $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2}$, and so on, reproduces the original picture. Let me explain you what happens in detail. The main rendering pipeline constructs a star and places it on the top-left corner of the screen. Its position and size are determined by the settings of the `textured_rectangle` module. The `blend_mode` module blends it together with the outcome of the backfeeding sub-pipeline, which is empty yet, and sends the overall output to both the screen and the `render_surface_single` module. The latter stores the screen inside a texture storage on the graphics hardware. In the next frame, the `mesh_basic_render` module projects this stored texture onto the flat grid defined by the `mesh_grid` module, which this time provides a second input to the `blend_mode`. Because the `mesh_grid` module defines a grid at coordinates ranging from $(-0.5;-0.5)$ to $(+0.5;+0.5)$ while the screen output is ranging from $(-1;-1)$ to $(+1;+1)$ the backfeeding picture has half the size of the incoming picture. The `blend_mode` has now two non-nil inputs and outputs the original image overlaid by the backfeeding image. The next frame this procedure will produce another clone of size $1/4^{\text{th}}$ and the frame after that one more at size $1/16^{\text{th}}$ and so on.



Figure 6-19. *Self-Similarity basic state output*

If you want to dismiss the black rectangle around the star, we left it here to make things clearer; just set the ALPHA value of the `clear_color` anchor of `render_surface_single` module to 0.0; see Table 6-19.

Table 6-19.

| <code>texture</code> → <code>buffers</code> → <code>render_surface_single</code> | |
|--|------------|
| <code>options / clear_color</code> | 0, 0, 0, 0 |

Remember, in ThMAD the ALPHA value is always the fourth color value. Zeroing this will effectively prevent the module from clearing its area each time before drawing the contents of the internal texture buffer.

In order to change this scaling factor and also to allow for translation of the backfeeding image, we add two more modules Mesh → modifiers → transformers → mesh_scale → modifiers → transformers → mesh_translate between the mesh_grid and the mesh_basic_render modules. We can use them later to change the overall characteristics of the self-similar stage.

The state we have so far is shown in Figure 6-20. Save it as your own state, because you can use it as a basis for lots of different experiments.

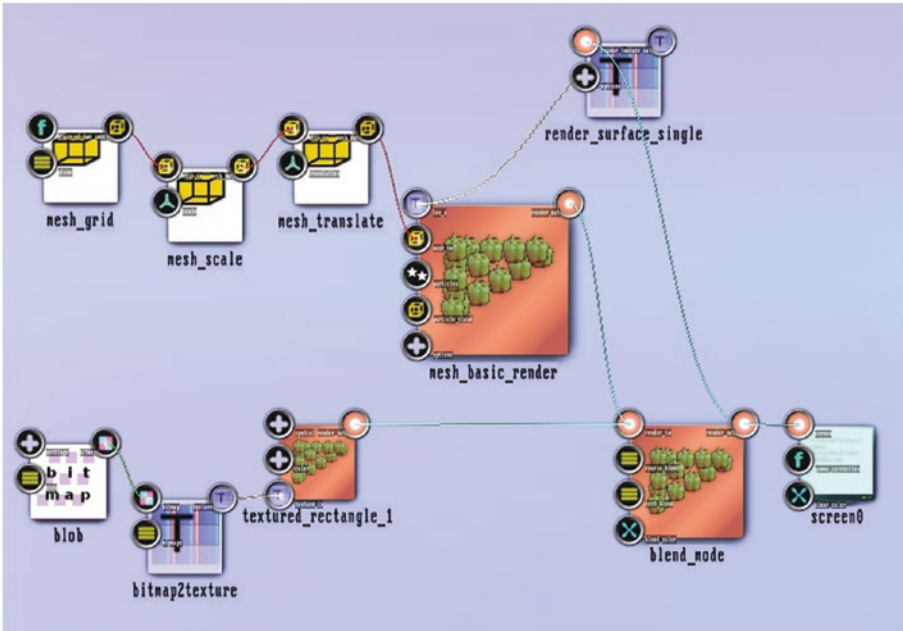


Figure 6-20. Self-similarity basic state

■ **Note** The backfeeding basic state is also available under A-6.1.2_Backfeeding_Basic inside the TheArtOfAudioVisualization folder.

Now to make things a little bit more interesting, we will construct a state using an algorithm similar to the KOCH curve; see Figure 6-14. It uses the same principles as the backfeeding just presented, but with some differences that allow for more clarity if we start from scratch. We will have three sub-pipelines in the end, and we start with the first.

■ **Note** The following state is also available under A-6.1.2_Backfeeding inside the TheArtOfAudioVisualization folder.

For the first sub-pipeline, place the following modules on an empty canvas:

- Mesh → generators → ribbon
- Renderers → mesh → mesh_basic_render
- Renderers → opengl_modifiers → gl_color
- Renderers → opengl_modifiers → blend_mode
- Renderers → opengl_modifiers → gl_translate

Connect them all in that order, and the `gl_translate` to the screen; see Figure 6-21. Note that `mesh_basic_render` has two mesh input anchors—use the one with the name `mesh_in`.

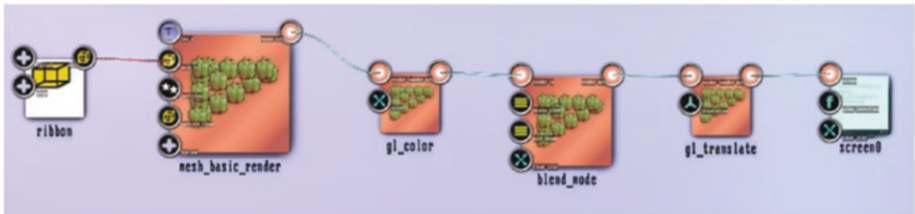


Figure 6-21. Fancy Koch curve, pipeline I

We use the `ribbon` module as a basis for the drawing, since it introduces some shape sugar and also moves with the time. The parameters are shown in Tables 6-20 to 6-24.

Table 6-20.

| mesh → generators → ribbon | |
|----------------------------|---------------|
| spatial / start_point | -0.7, -0.5, 0 |
| spatial / end_point | 0.7, -0.5, 0 |
| spatial / up_vector | 0.8, 0.8, 0.9 |
| shape / width | 0.1 |
| shape / skew_amp | 4.0 |
| shape / time_amp | 1.5 |
| shape / segm_count | 60 |

Table 6-21.

| renderers → mesh → mesh_basic_render | | |
|---|----|---|
| options / vertex_colors | no | We provide our own colors, so this is set “no” here |
| options / use_display_list | no | |
| options / use_vertex_colors | no | We provide our own colors, so this is set “no” here |
| options / particles_size_center | no | |
| options / particles_size_from_color | no | |
| options / ignore_uvvs_in_vbo_updates | no | |

Table 6-22.

| renderers → opengl_modifiers → gl_color | | |
|--|-----|---------------------|
| color | any | Any color you like. |

Table 6-23.

| renderers → opengl_modifiers → blend_mode | | |
|--|--|-----------|
| source_blend | | SRC_ALPHA |
| dest_blend | | ONE |

Table 6-24.

| renderers → opengl_modifiers → gl_translate | | |
|--|-------------|---|
| translation | 0, 0.3, 0.0 | Just move the blended output into the center of the screen. |

This should produce the basic fluttering ribbon; see Figure 6-22.

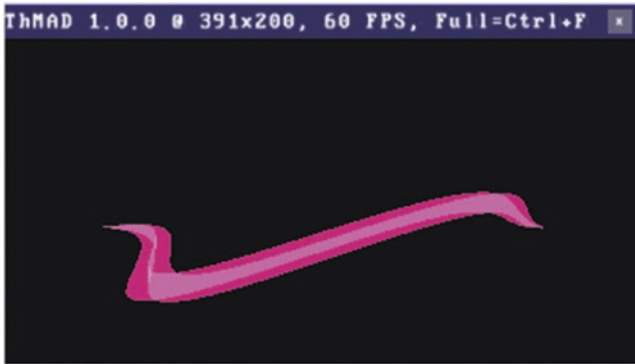


Figure 6-22. Fancy Koch curve, pipeline I output

The `gl_color` module defines the color of the ribbon. The ribbon module does not do it anyways, and we told the `mesh_basic_render` to not do it as well.

The sub-pipeline number two consists of the modules:

- Texture → buffers → `render_surface_single`
- Texture → modifiers → `scale_one`
- Texture → modifiers → `rotate`
- Renderers → basic → `textured_rectangle`

The `render_surface_single` is as in the state before the basis for the backfeeding. For the connections, see Figure 6-23.

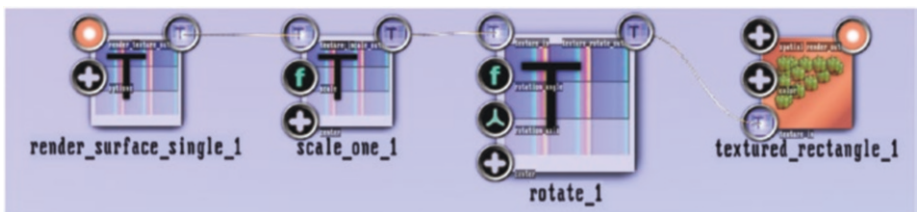


Figure 6-23. Fancy Koch curve, pipeline II

The parameters of those modules are shown in Tables 6-25 to 6-28.

Table 6-25.

| texture → buffers → render_surface_single | | |
|--|------------|-----------------------------------|
| options / texture_size | 1024x1024 | |
| options / support_feedback | yes | Mandatory for backfeeding to work |
| options / alpha_channel | yes | |
| options / clear_color | 0, 0, 0, 0 | |

Table 6-26.

| texture → modifiers → scale_one | | |
|--|--------------|---|
| scale | 3.0 | Scaling down by 1/3 rd as for the original KOCH curve. |
| center / use_scale_center | yes | Using the center below |
| center / scale_center | 0.5, 0.25, 0 | Scale at the center of the ribbon |

Table 6-27.

| texture → modifiers → rotate | | |
|-------------------------------------|---------------|---|
| rotation_angle | 0.15 | Roundabout 1/6 th of a full rotation, thus 60° |
| rotation_axis | 0, 0, 1 | Rotate on the x-y plane |
| center / use_rotate_center | yes | Using the center below |
| center / rotate_center | 0.66, 0.25, 0 | Makes for the figure |




Table 6-28.

| renderers → basic → textured_rectangle |
|---|
| Keep all anchors at their default values |

As for sub-pipeline number three, clone all modules from pipeline number two—for each module click and drag while pressing Ctrl+Alt. Connect the modules of the sub-pipeline III the same way you did for sub-pipeline II. See Figure 6-24.

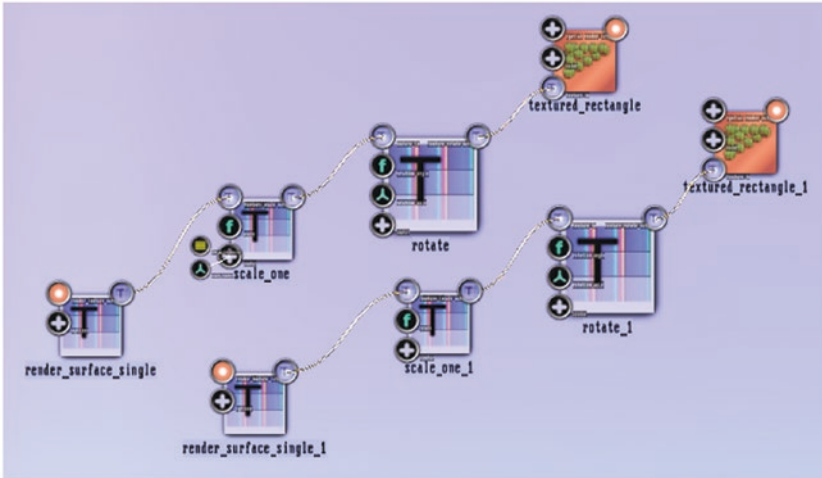


Figure 6-24. Fancy Koch curve, pipeline II + III

Change just the rotation module of pipeline III to read, as shown in Table 6-29.

Table 6-29.

texture → modifiers → rotate

| | | |
|----------------------------|---------------|--|
| rotation_angle | 0.15 | Roundabout 1/6 th of a full rotation, thus 60°. Note the difference of the sign compared to pipeline II |
| rotation_axis | 0, 0, 1 | Rotate on the x-y plane |
| center / use_rotate_center | yes | Using the center below |
| center / rotate_center | 0.33, 0.25, 0 | Makes for the figure |



Connect the output from the `blend_mode` to the `render_ - surface_single` modules of both pipelines II and III. Connect both pipelines II and III output anchors of modules `textured_ - rectangle`. Change the order of the input connections to `blend_mode` so that pipeline I comes first; see Figure 6-25.

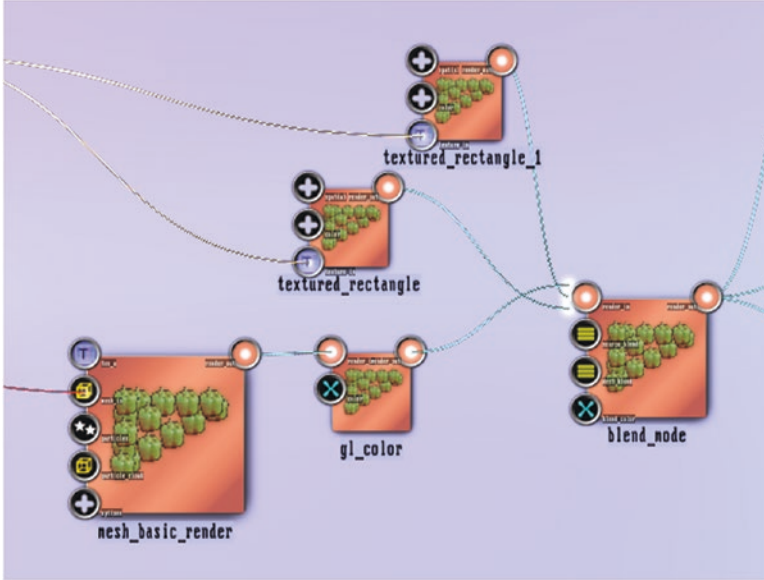


Figure 6-25. Fancy Koch curve, `blend_mode` input order

The output should now look like Figure 6-26. This is a correct Koch curve with the original mapping algorithm, just for ribbons instead of lines. This does not yet look overwhelmingly interesting, but we can easily make it appear more compelling. Just change anchor values shown in Tables 6-30 to 6-33.



Figure 6-26. Fancy Koch curve, output

The `skew_amp` of module `ribbon` is temporarily reduced to 0.0 to make things clearer.

Table 6-30.**texture → modifiers → scale_one**

Pipeline II

scale

2.2

Make it bigger

Table 6-31.**texture → modifiers → rotate**

Pipeline II

No need to change, but you can play with it.

Table 6-32.**texture → modifiers → scale_one**

Pipeline III

scale

1.2

Make it bigger.

Table 6-33.**texture → modifiers → rotate***Pipeline III*

rotation_angle

-0.12

Changing angle a little

center / rotate_center

0.37, 0.25, 0

Changing a little

The output of the altered state now looks much more fancy; see Figure 6-27. The complete state from a bird view perspective is depicted in Figure 6-28.

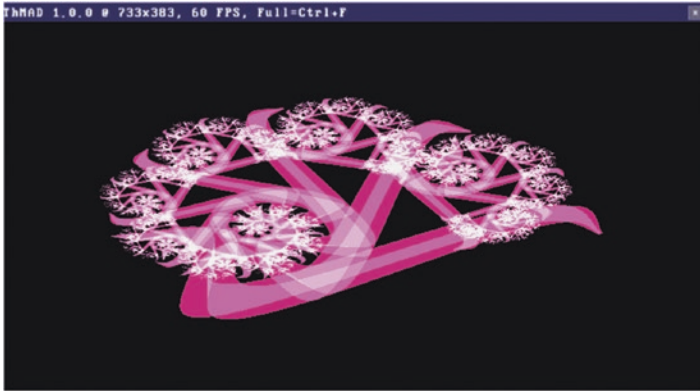


Figure 6-27. Altered fancy Koch curve, output
Just a snapshot; all the ribbons are fluttering.

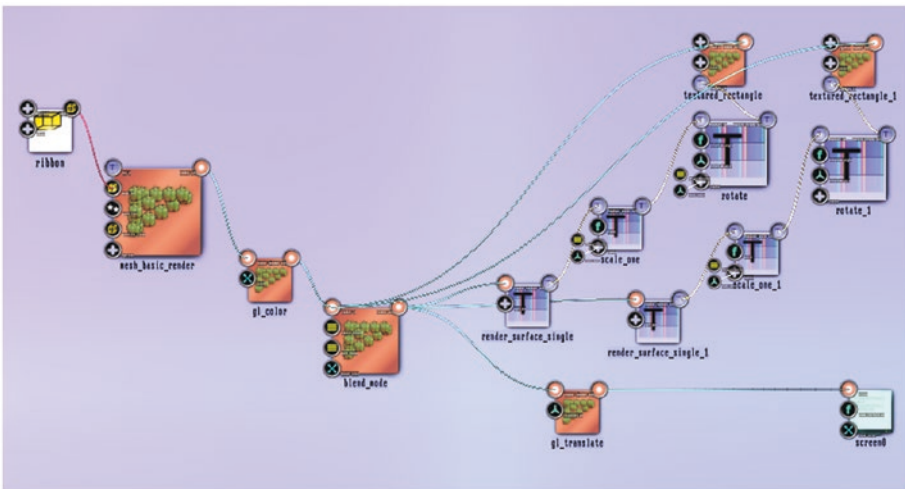


Figure 6-28. Fancy Koch curve, complete state

Feel free to play around with anchor values and to connect one or the other with sound input.

■ **Caution** Due to the backfeeding, it might easily happen that the output screen completely fills with white. This usually does not happen when the backfed image data scales down, but if it happens to you while playing with the values, it might be necessary to revert changes, save the state, and restart ThMAD to get some meaningful output again. Or temporarily change the blend modes to ZERO/ZERO inside the `blend_mode` module and then revert.

Particle Systems

Particle systems are about two things:

- Multiplicity of objects
- Physics movement laws and collision rules

Since ThMAD does not allow for something like a loop construct, the multiplicity property of particle systems is the only way of generating object many times. Furthermore, particle systems may obey physics rule like gravity, wind, or the bouncing at rigid walls. ThMAD allows for particle systems with these properties via the methodology shown in Figure 6-29.

An EMITTER continuously creates particles—we have the following emitter types:

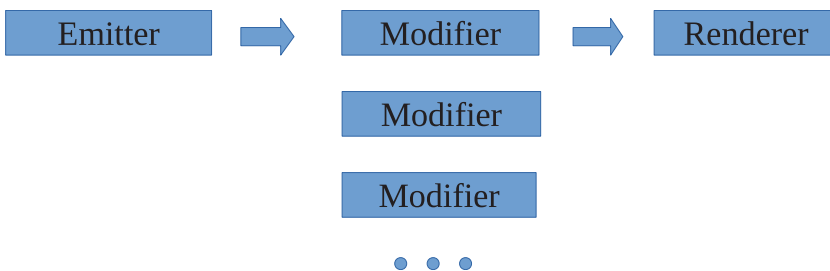


Figure 6-29. Particle system methodology

- *A spray emitter.* Continuously creates particles at some single point in space.
- *A bitmap emitter.* Converts each pixel of a bitmap, no matter whether defined foreground or background, to a single particle with the same color as the bitmap pixel.
- *A mesh emitter.* Like spray emitter, but uses each vertex of a mesh as a possible source point.

A MODIFIER applies rules to existing particles. This covers the following actions:

- Apply gravity.
- Apply wind.
- Place a wall where particles will stop their movement or bounce back.
- Apply fluid like interactions between particles.
- Apply particle resizing.

A RENDERER converts a particle system into renderable objects. Only after the renderer, the particles are graphical objects—prior to that point they are just collections of numbers. ThMAD renders in one of the following ways:

- *Simple rendering.* Converts the particles into graphical objects using a texture. Augments the possibilities we have inside the modification step by defining: how size evolves during the lifespan of each particle, how color evolves during the lifespan of each particle. Despite the *simple* in its name, it is actually quite powerful and you can use shader programs to control the rendering operations.
- *Center rendering.* This is a special effects variant where one side of the blob is clamped to a center point.
- *Ribbon rendering.* Another special effects variant where the particles are converted to ribbons moving in 3D.
- *Extended renderer.* A renderer using textures, a size and color lifespan definition, and if you want to have custom shader language control.
- *Spark renderer.* Allows for the interaction of particles with particles approaching each other, creating a spark.

In the rest of this section, we create a couple of particle system samples.

Waterfall

A waterfall is a good candidate for particle system. A lot of objects fall down, obeying the physics of gravity and wind, and eventually end up on the ground of the waterfall.

■ **Note** The samples of this subsection are as sources available under A-6.2.1_Particlesystems_Waterfall* inside the TheArtOfAudioVisualization folder.

For simulating a waterfall, we place the following modules on an empty canvas. To create one, right-click and then choose New → Empty Project:

- Renderers → opengl_modifiers → cameras → orbit_camera
- Renderers → opengl_modifiers → light_directional
- Renderers → opengl_modifiers → material_param
- Renderers → opengl_modifiers → depth_buffer
- Renderers → opengl_modifiers → backface_culling
- Renderers → opengl_modifiers → blend_mode

Connect them all and also the camera to the screen module; see Figure 6-30. Adjust the following parameters for those modules. See Tables 6-34 to 6-37.



Figure 6-30. Waterfall, basic 3D setup

Table 6-34.

renderers → opengl_modifiers → light_directional

| | |
|----------------|-----------------------|
| enabled | YES |
| position | 0.29; 0.29; 0.12 |
| ambient_color | 0; 0; 0; 1 |
| diffuse_color | 0.67; 0.69; 0.95; 1.0 |
| specular_color | 0.16; 0.79; 0.92; 1.0 |

Table 6-35.

renderers → opengl_modifiers → material_param

| | |
|----------------------|-----------------------|
| ambient_reflectance | 0.2; 0.2; 0.2; 1.0 |
| diffuse_reflectance | 0; 0; 1; 1 |
| specular_reflectance | 0.96; 0.82; 0.82; 1.0 |
| emission_intensity | 0; 0; 0; 1 |
| specular_exponent | 5.0 |

Table 6-36.

renderers → opengl_modifiers → depth_buffer

| | |
|------------|---------|
| depth_test | ENABLED |
| depth_mask | ENABLED |

Table 6-37.

| | |
|--|---------|
| renderers → opengl_modifiers → backface_culling | |
| status | ENABLED |

The two modules `depth_test` and `backface_culling` are not too important here, since we deal with small particles only. For special effects or playing around later, we add them anyway.

Now for the particle system sub-pipeline, add the following modules:

- Texture → particles → blob
- Renderers → particlesystems → simple
- Particlesystems → modifiers → floor
- Particlesystems → modifiers → basic_wind_deformer
- Particlesystems → modifiers → size_mult
- Particlesystems → modifiers → basic_gravity
- Particlesystems → generators → basic_spray_emitter

Connect them as shown in Figure 6-31.

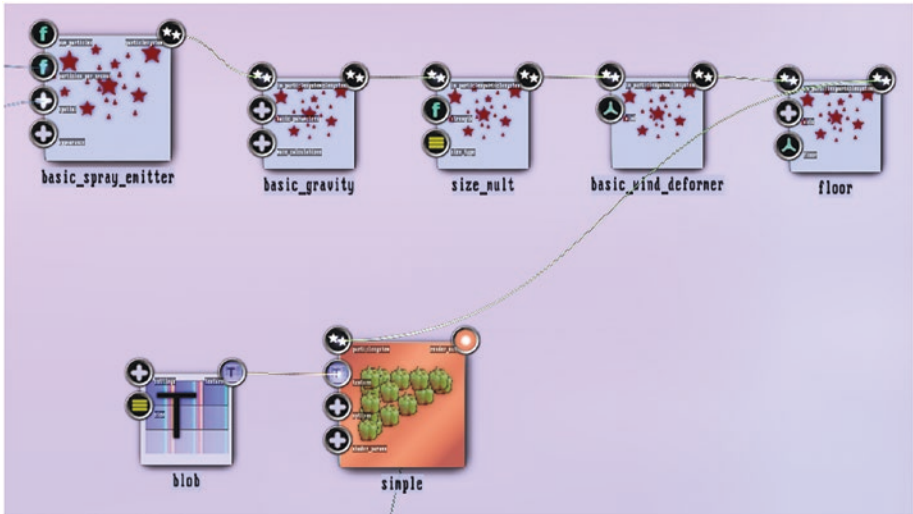


Figure 6-31. Waterfall, particle system sub-pipeline

For those parameters, adjust them as shown in Tables 6-38 to 6-42.

Table 6-38.

| particlesystems → modifiers → floor | | |
|--|-------------|---|
| axis / y / y_floor | yes | Enable a x-z-plane floor |
| axis / y / y_bounce | yes | Let particles bounce on the floor |
| axis / y / y_loss | 85.0 | Percentage of particles lost before bouncing. If we make this number too small, the particles behave more like rubber balls instead of water. |
| axis / refraction | yes | Water does not behave like rigid balls, instead upon hitting the floor it spreads into all directions |
| axis / refraction_amount | 100; 10; 0 | The amount of refraction—water mainly spreads parallel to the floor |
| floor | 0; -0.97; 0 | Position of the floor |

Table 6-39.

| particlesystems → modifiers → basic_wind_deformer | | |
|--|----------------|-------------------------|
| wind | 0.22; 0.0; 0.0 | Some wind blowing to +x |

Table 6-40.

| particlesystems → modifiers → size_mult | | |
|--|------|-------------------------------------|
| strength | 0.71 | Make the particles a little smaller |

Table 6-41.

| particlesystems → modifiers → basic_gravity | | |
|--|---------------|--|
| basic_parameters / center | 0; -1.5; 0 | Center of gravity |
| basic_parameters / amount | 0; 0.079; 0 | Gravity only along y-axis |
| basic_parameters / friction | 0.6; 0.6; 1.8 | If this is non-null, particles will not accelerate endlessly but be suspect to friction. |

Table 6-42.

| particlesystems → generators → basic_spray_emitter | | |
|---|-----------------|---|
| num_particles | 10000 | The number of particles to use. Stays constant, even if a modifier or renderer says a particle must die. In this case, it just gets reinitialized. |
| particles_per_second | 2000 | Later being controlled by sound. Limits the number of particles being initialized or reinitialized per second. If small enough, some dead particles will stay un-reinitialized for some time. Set to -1 to disable that limit. |
| spatial / emitter_position | 0; 1; 0 | Where particles get born or reinitialized. |
| spatial / speed / speed_x | 1.0 | Later controlled by sound input. The x-spread of particle velocities when born or reinitialized. |
| spatial / speed / speed_y | 0.45 | |
| spatial / speed / speed_z | 0.07 | |
| spatial / speed_type | random_balanced | How to distribute speed. The value here chooses a random number from [-speed_x; +speed_x] for the x-value. Similar for the other axes. |
| spatial / size / particle_size_base | 0.03 | Base size of a particle |
| spatial / size / particle_size_random_weight | 0.01 | Amount of size randomization |
| appearance / color | 1;1;1;1 | Unused |
| appearance / time / particle_lifetime_base | 3.48 | Base lifetime of a particle in seconds. If exceeded, a particle remains in undefined state until reinitialized. |
| appearance / time / particle_lifetime_random_weight | 1.0 | Amount of lifetime randomization |

Connect this sub-pipeline to the basic 3D rendering pipeline: use anchors `render_out` from the `simple` module and `render_in` from the module `blend_mode`. This will produce a waterfall; see Figure 6-32.

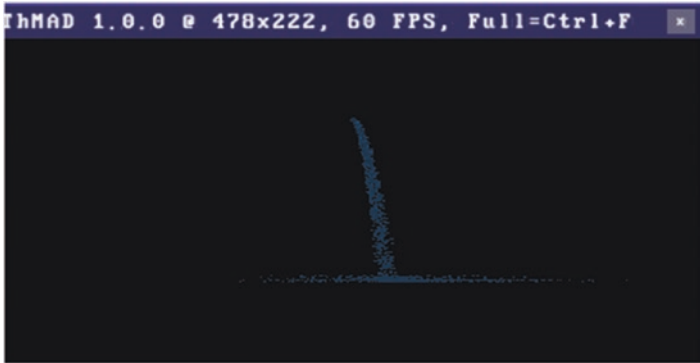


Figure 6-32. *Waterfall, no sound control*

We want to have some sound control added. To that aim, add these modules:

- Maths → limiters → float_clamp
- Maths → arithmetics → ternary → float → mult_add, twice
- Sound → input_visualization_listener

Connect them as shown in Figure 6-33. Apply parameters as shown in Tables 6-43 to 6-45.

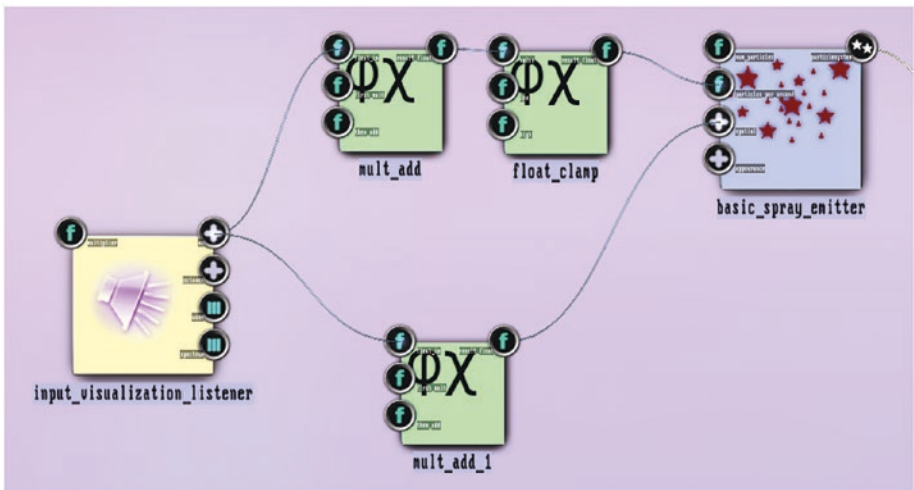


Figure 6-33. *Waterfall, sound control*

The lower *mult_add* is connected to the *speed_x* anchor of the *basic_spray_emitter*. Both *mult_add* modules are connected to any of the *vu* sub-anchors of *sound_visualization_listener*.

Table 6-43.

maths → limiters → float_clamp

Make sure we generate / re-initialize between 1000 and 5000 particles per second

| | |
|------|------|
| low | 1000 |
| high | 5000 |

Table 6-44.

maths mult_add → arithmetics → ternary → float →

The one connected to float_clamp.

| | |
|------------|------|
| first_mult | 1000 |
| then_add | 1000 |

Table 6-45.

maths → arithmetics → ternary → float → mult_add

The other one, connected to anchor “x_speed” of basic_spray_emitter.

| | |
|------------|------|
| first_mult | 0.4 |
| then_add | 0.13 |

Add sound input and the result will then be a waterfall with more mass and more width according to the sound volume. See Figure 6-34.

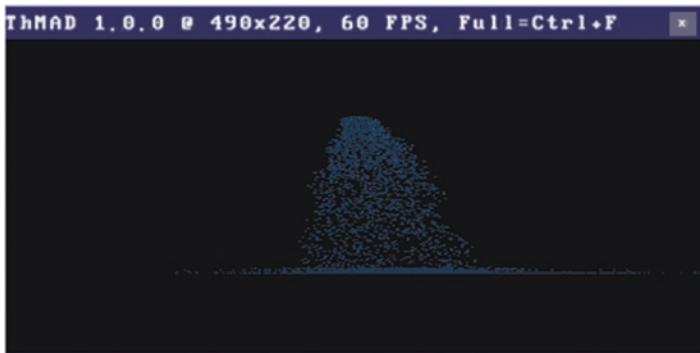


Figure 6-34. Waterfall, sound controlled, output

As of now, the waterfall looks like it might consist of grains instead of water. To improve the similarity to natural water, as a variation to what we just generated, we add a blurring effect.

Blurring is about mixing newer and older instances of a rendered picture. We already learned how to do that in two dimensions earlier in the chapter. For a simpler approach compared to the one we used there, the module `Texture→effects→blur` gets chosen here. We use blurring only after the camera did its work, which feels like blurring in 3D. This is because real blurring in 3D is not an option, since our textures can handle only two dimensions. But after the camera, we internally have a two-dimensional projection, so we can then utilize two-dimensional blurring. We need three modules for the blurring:

- `Renderers→basic→textured_rectangle`
- `Texture→effects→blur`
- `Texture→buffers→render_surface_single`

Insert them between the camera and the screen, as shown in Figure 6-35. Apply parameters as shown in Tables 6-46 to 6-48.

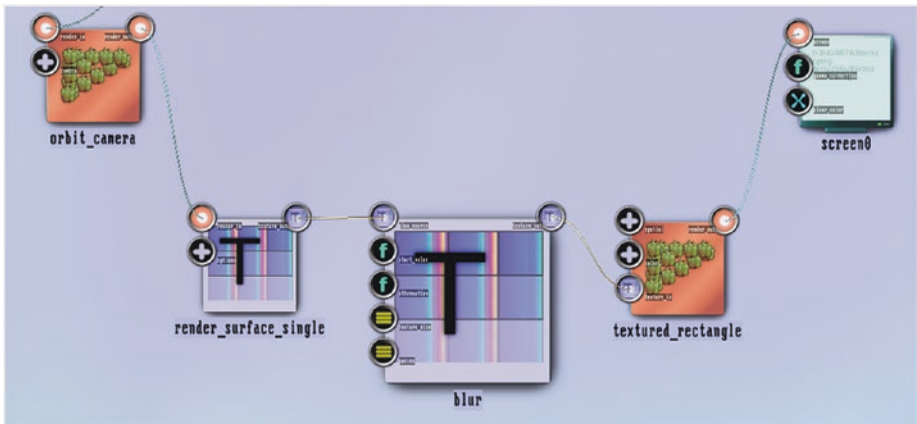


Figure 6-35. *Waterfall, blurred*

Table 6-46.

| |
|---|
| <code>renderers→basic→textured_rectangle</code> |
| Leave all parameters at their default value |

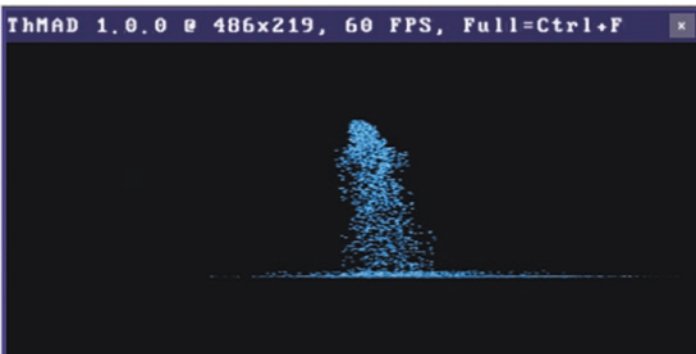
Table 6-47.**texture → effects → blur**

| | | |
|--------------|-----------|---|
| start_value | 2.2 | A shift, applied to texture coordinates while mixing original and older picture |
| attenuation | 1.22 | Mixing intensity |
| texture_size | 1024x1024 | You can use smaller sizes to enhance the blurring effect |
| passes | TWO | Intensify the blurring effect compared to “ONE” |

Table 6-48.**texture → buffers → render_surface_single**

| | | |
|--------------|-----------|--|
| texture_size | 1024x1024 | You can use smaller sizes to enhance the blurring effect |
|--------------|-----------|--|

The blurred waterfall will look like Figure 6-36.

**Figure 6-36.** *Waterfall, blurred, output*

As a side effect, this blurred waterfall looks somewhat more massive without increasing the number of particles used.

Image Bit Particles

ThMAD contains a module `Particulatesystem → generators → bitmap2particulatesystem`, which you can use to let a particle system be generated by the color pixels of a bitmap. We will provide a sample here.

■ **Note** The samples of this subsection are as sources available under A-6.2.2_* inside the TheArtOfAudioVisualization folder.

Start with an empty state, as usual, you can create one by right-clicking, the New → Empty Project. Start with placing the same basic 3D modules as described in the beginning of this chapter, as shown in Figure 6-30, but use parameters as shown in Tables 6-49 to 6-54.

Table 6-49.

| | |
|--|--------------------|
| renderers → opengl_modifiers → cameras → orbit_camera | |
| rotation | -0.03; 0.99; -0.11 |
| distance | 0.72 |
| perspective_correct | yes |

Table 6-50.

| | |
|---|--|
| renderers → opengl_modifiers → light_directional | |
| Values do not matter here, coloring is done by shader code, see below | |

Table 6-51.

| | |
|---|--|
| renderers → opengl_modifiers → material_param | |
| Values do not matter here, coloring is done by shader code, see below | |

Table 6-52.

| | |
|--|---------|
| renderers → opengl_modifiers → depth_buffer | |
| depth_test | ENABLED |
| depth_mask | ENABLED |

Table 6-53.

| | |
|--|---------|
| renderers → opengl_modifiers → backface_culling | |
| status | ENABLED |

Table 6-54.

| | |
|--|---------------------|
| renderers → opengl_modifiers → blend_mode | |
| source_blend | ONE |
| dest_blend | ONE_MINUS_SRC_ALPHA |

Now for the particle system sub-pipeline add following modules:

- Renderers → particlesystems → render_particle_shader
- Texture → particles → blob
- Particlesystems → modifiers → basic_wind_deformer
- Particlesystems → generators → bitmap_to_particlesystem
- Bitmaps → loaders → png_bitm_load

Connect them as shown in Figure 6-37. This will create particles from the bitmap pixels, then apply wind to it. Set a parameters the values from Tables 6-55 to 6-59.

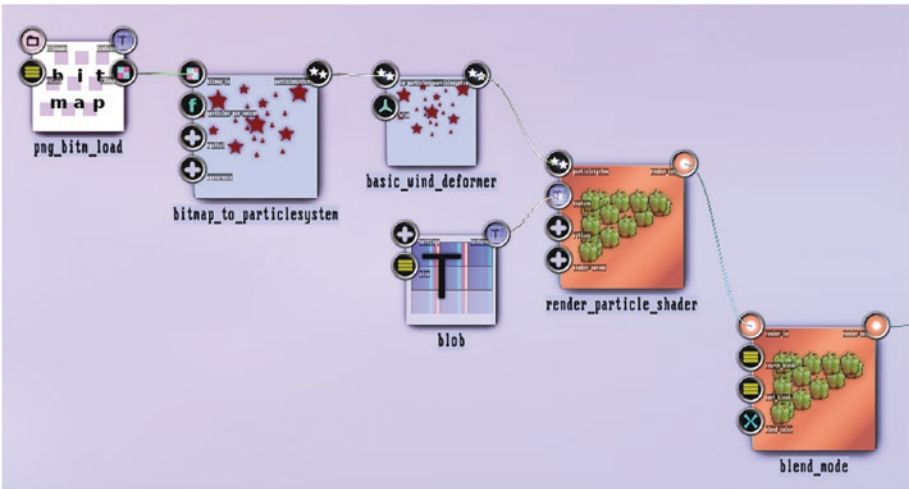


Figure 6-37. Bitmap particle system

Table 6-55.

renderers → particlesystems → render_particle_shader

You can make extremely interesting things by writing custom shader programs. For our current purpose we just use the texture programs provided as defaults.

| | | |
|----------------------------------|-------------------|---|
| shader_params / vertex_program | Leave the default | The vertex shader part of the shader pipeline. The default program will take care of spatial positioning and fetching the particle color. |
| shader_params / fragment_program | Leave the default | The fragment shader part of the shader pipeline. The default program will use the particle color mixed with the texture ALPHA, and discard the texture color. |

Table 6-56.**texture → particles →**

blob

settings / alpha yes

size 8x8 Since particles are small, we can use small textures

Table 6-57.**particlesystems → modifiers → basic_wind_deformer**

wind

0; 0; 0

We will later connect this to the sound input, see below

Table 6-58.**particlesystems → generators → bitmap_to_particlesystem**

| | | |
|---|-------------|---|
| particles_per_second | 10000 | The number of particles per second which may be born from the bitmap |
| spatial / bitmap_size | 1.0 | |
| spatial / bitmap_normal | 0; 1; 0 | Defines the orientation in space. Here we define the bitmap to live in an x-z plane. |
| spatial / bitmap_upvector | 1; 0; 0 | Defines the angle with the normal vector as the axis. |
| spatial / bitmap_position | 0; 0; -0.25 | Where the bitmap lies in space. |
| spatial / speed / speed_x | 0 | We let the particles rest immediately after creation. Only the wind module will thus move them. |
| spatial / speed / speed_y | 0 | |
| spatial / speed / speed_z | 0 | |
| spatial / size / particle_size_base | 0.1 | The particles' size. |
| spatial / size / particle_size_random_weight | 0.01 | A random contribution to the particles' size. |
| appearance / time / particle_lifetime_base | 1.16 | Number of seconds a particle lives before it dies and/or needs to be reinitialize. |
| appearance / time / particle_lifetime_random_weight | 1.0 | A random contribution to the particle lifetime. |

Table 6-59.

| bitmaps → loaders → → png_bitm_load | | |
|-------------------------------------|-----|--|
| filename | any | Choose a PNG file with dimensions not bigger than 256x256, to not overburden the hardware. |

The PNG file must reside in `/home/[USER]/thmad/[VERSION]/data/resources`.

You can now already see the particle system—since particles are not moving yet, your image seems to blurringly waft around. This comes from the creation and dying mechanism, and automatic size changes between that. See Figure 6-39.

To introduce sound control, place the following additional modules on the canvas:

- Maths → converters → 3float_to_float3
- Maths → interpolation → float_smoother
- Audio → input_visualization_listener

Connect them as seen in Figure 6-38. Apply the following parameters, as shown in Tables 6-60 to 6-62.

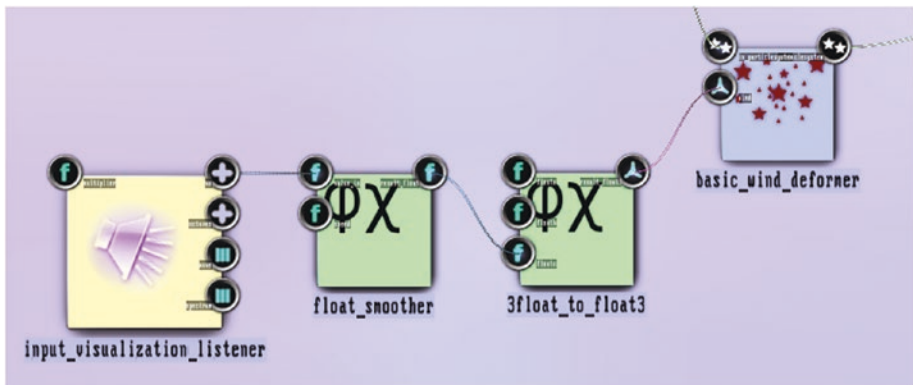


Figure 6-38. Bitmap particle system, sound control

The `float_smoother` is connected to the `vu / vu_1` anchor of the listener.

Table 6-60.

| maths → converters → 3float_to_float3 | | |
|---------------------------------------|---|---------------------------------|
| floata | 0 | The x-amount of the wind |
| floatb | 0 | The y-amount of the wind |
| floatc | - | Connected to the sound listener |

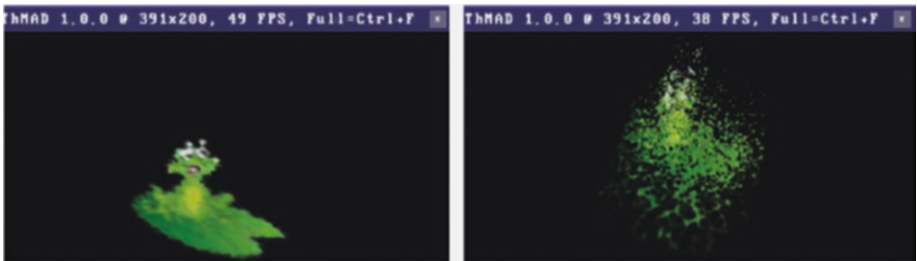
Table 6-61.

| maths → interpolation → float_smoother | | |
|--|-----|----------------------------------|
| value_in | - | Connected to the sound listener |
| speed | 5.0 | Lower values mean more smoothing |

Table 6-62.

| sound → input_visualization_listener | | |
|--------------------------------------|------|--|
| multiplier | 0.38 | Sound volume multiplier, change according to your needs. |
| vu / vu_l | - | As output connected to the smoother above |

Play some music and you will see the bitmap particles blowing away, as shown in Figure 6-39.

**Figure 6-39.** Bitmap particle system, output without sound input

Center Clamped Particle Systems

A somewhat unrealistic but impressive result that delivers a variant of the particle systems gives us using the Module renderers → particlesystems → render_particle_center.

What it basically does is clamp two of the points a texture defined to a center point. We will provide an example here, but in contrast to explaining it step by step, I ask you to refer to the sources provided with the installation. This section just explains it from a bird's eye view and point out important settings.

■ **Note** The samples of this subsection are as sources available under A-6.2.3_Particlesystems_Center* inside the TheArtOfAudioVisualization folder.

Load the A-5.5.3_Particlesystems_Center_Blur state into Artiste, and you will see what is shown in Figure 6-40.

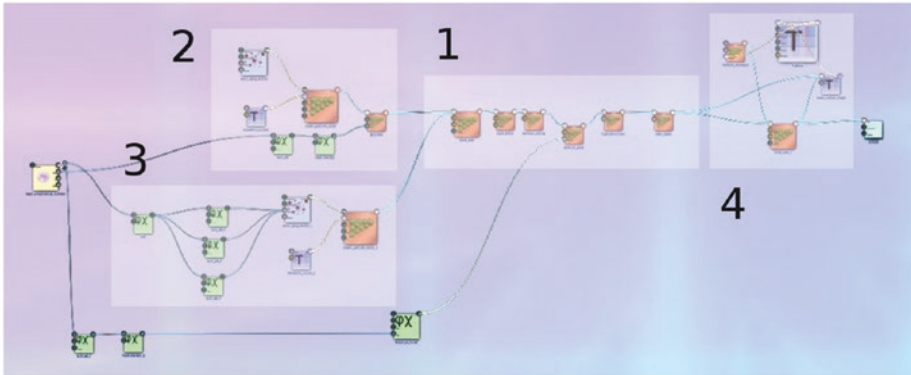


Figure 6-40. Centered particle system, full state

The part labeled with (1) is the usual 3D rendering pipeline. However, both `backface_culling` and `depth_buffer` are disabled; otherwise, the texture backgrounds will overlap other textures that we do not want here. The light is switched on and all white, and the material parameters are all white as well, with the addition that the ALPHA channel is connected to sound input, so that the picture will get more intense with increased volume.

The part (2) designates one set of the objects we will see. The source is concentric circles, and in this case we add a rotation around the y-axis controlled by the sound input.

Part (3) is another set of concentric clamped circles in a different color, and this time we control their speed using sound input.

Part (4) takes care of a blurring effect. If we did not use blurring, the fact that particles get clamped to the origin and have a limited lifetime would introduce some unwanted nervousness. The effect gets achieved by the `highblur` module, which used backfeeding and gets introduced in another story in this chapter as well. You can play around with the parameters of this sub-pipeline, and one very important parameter is the blending mode chosen. The chosen value set here is `ONE/ONE`, which means that images, the original image, and the backfed image are painted over another. For this to yield the expected result, the anchor `color/global_alpha`, here 0.89, is crucial, and tiny changes to this value yields vast changes in the outcome.

The result is shown in Figure 6-41, with the violet set of rays rotating according to the sound basses, and the yellow set changing its contribution with more sound volume.

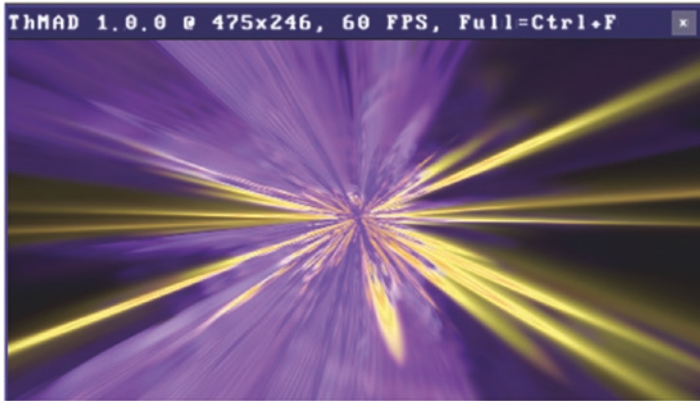


Figure 6-41. Centered particle system, output

Ribbon Particles

The module `Renderers → particlesystems → render_particle_ribbon` sends particles along a ribbon moving in space and driven by movement laws similar to gravity.

It can be used for interesting visuals, and we want to provide a sample here. Instead of presenting a step-by-step instruction manual we will be looking at the final state present after installation of ThMAD and explain important building blocks.

■ **Note** The state of this subsection is as a source available under `A-6.2.4_Particlesystems_Ribbon` inside the `TheArtOfAudioVisualization` folder.

The state is shown in Figure 6-42 together with the main parts highlighted.

- Part (1) is the typical 3D sub-pipeline; however, with the `depth_buffer` module disabled. You can try what happens when the `backface_culling` is disabled as well. Both light and material have all three light components: ambient light, diffuse light and specular light. The light will be rotated around the scene by part (4) and as for its ambient light intensity will be controlled by sound input.
- Part (2) defines the particle system and its renderer. The renderer itself does not have the capability to have a texture assigned to the rendering process, but by adding the module `Texture → opengl → texture_bind`, we can achieve that nevertheless. Here we assign a `blob` module, which will lead to the ribbon resembling a tube. Some sound input has been connected to the ribbon width here.

- Part (3) defines a constant main rotation, which will give depth to the scene. The oscillator is set to “saw” mode, which is what we use quite often when we want rotations with a constant angular velocity.
- Part (4) controls the rotation of the light source. It does so by combining two rotation quaternions using the `quat_mul` module. A quaternion internally consists of four numbers completely defining a rotation axis and a rotation angle, and by quaternion multiplication we have the exact counterpart of two subsequent rotations around two different axes with two different angles. The light position needs a vector (x, y, z) and if we use the module `float3_rotate_by_quat`, starting from any position like $(0,0,1)$ and using the aforementioned combined quaternion, the outcome will be a rotated variant of $(0,0,1)$. Using the two oscillators in “saw” and “sine” mode, but with some sound input added to the angle of one of them, we will have a steady rotation plus some modulation with a small reactivity to sound input. The latter is done by additively accumulating sound volume and the result to the phase anchor of one of the quaternion rotation oscillators.



Figure 6-42. Particle system rendered along a ribbon

The output will be a rotating splattered tube world, as shown in Figure 6-43.

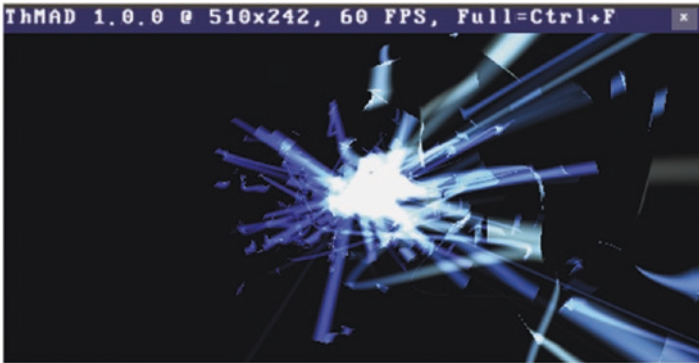


Figure 6-43. Particle system rendered along a ribbon, output

Glowing Objects

Objects can get a shiny glow by superimposing a blurred version over the original. As a sample, place the following modules on the canvas and connect them as shown in Figure 6-44.

- Renderers → opengl_modifiers → blend_mode
- Renderers → opengl_modifiers → cameras → orbit_camera
- Renderers → opengl_modifiers → light_directional
- Renderers → opengl_modifiers → depth_buffer
- Renderers → opengl_modifiers → backface_culling
- Renderers → opengl_modifiers → material_param
- Renderers → mesh → mesh_basic_render
- Mesh → solid → mesh_box

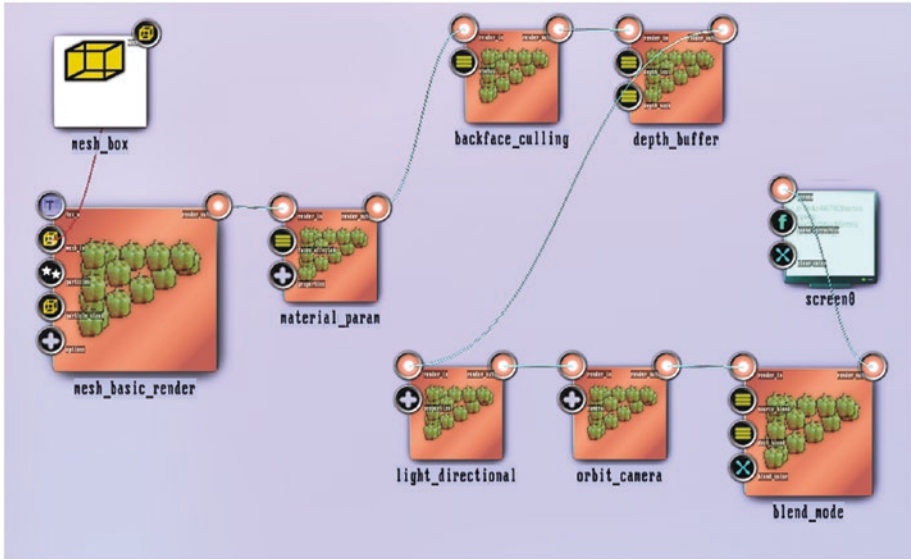


Figure 6-44. Objects with glow, basic state

■ **Note** The states of this subsection are as sources available under A-6.3_Glowing_Objects* inside the TheArtOfAudioVisualization folder.

Set the parameters in Tables 6-63 to 6-68.

Table 6-63.

| renderers → opengl_modifiers → cameras → orbit_camera | |
|---|------------------|
| rotation | 0.52; 0.05; 0.85 |
| distance | 3.5 |
| fov | 30.0 |
| perspective_correct | yes |

Table 6-64.**renderers → opengl_modifiers → light_directional***A white light*

| | |
|----------------|------------------|
| enabled | YES |
| position | 0.57; 0.25; 0.78 |
| ambient_color | 0; 0; 0; 1 |
| diffuse_color | 1; 1; 1; 1 |
| specular_color | 1; 1; 1; 1 |

Table 6-65.**renderers → opengl_modifiers → depth_buffer**

| | |
|------------|---------|
| depth_test | ENABLED |
| depth_mask | ENABLED |

Table 6-66.**renderers → opengl_modifiers → backface_culling**

| | |
|--------|---------|
| status | ENABLED |
|--------|---------|

Table 6-67.**renderers → opengl_modifiers → material_param**

| | |
|----------------------|-----------------------|
| ambient_reflectance | 0.2; 0.2; 0.2; 1.0 |
| diffuse_reflectance | 0.95; 0.76; 0.35; 1.0 |
| specular_reflectance | 0.95; 0.87; 0.87; 1.0 |
| emission_intensity | 0; 0; 0; 1 |
| specular_exponent | 13.0 |

Table 6-68.**renderers → mesh → mesh_basic_render**

Leave all values at their default

The output will be a box as shown in Figure 6-45. Now add the following modules to the scene:

- Renderers → basic → textured_rectangle
- Texture → effects → blur
- Texture → buffers → render_surface_single

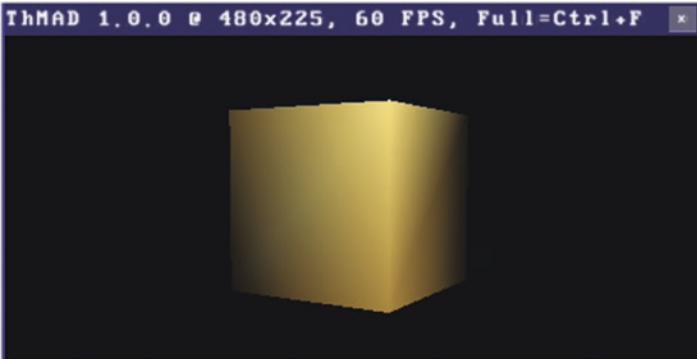


Figure 6-45. Objects with glow, basic state, output

Connect and insert them as shown in Figure 6-46. Note that the order at the input anchor of `blend_mode` is important. You usually cannot see it, but to check and fix it, double-click on the anchor and drop it on of the small sub-anchors to change the order; see Figure 6-47.

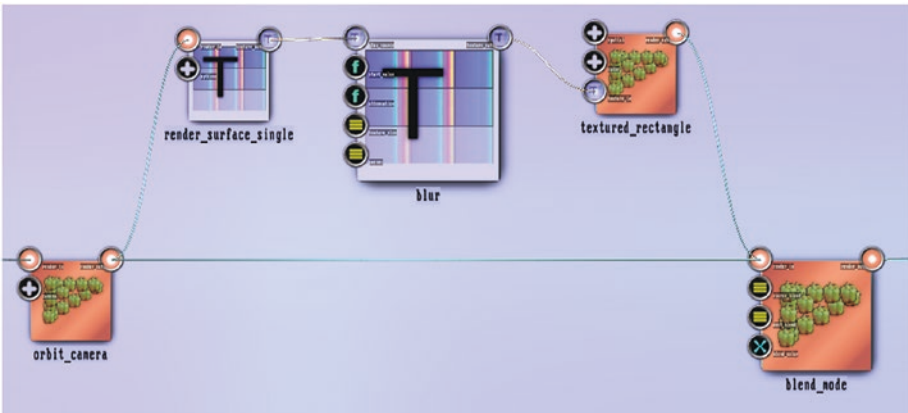


Figure 6-46. Objects with glow, glowing addition

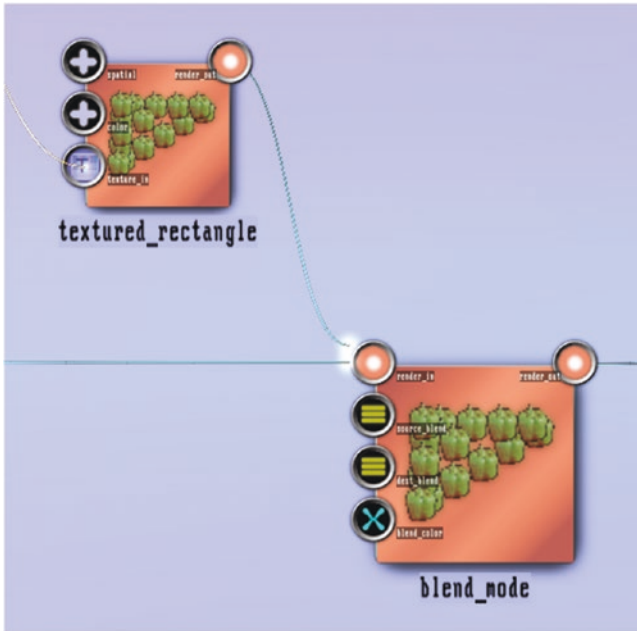


Figure 6-47. Objects with glow, blend input order

Set the added module parameters as shown in see Tables 6-69 to 6-71.

Table 6-69.

renderers → basic → textured_rectangle

| | | |
|------|-------|--|
| size | 1.007 | 1.0 will do as well, adding a small amount will increase the glow effect |
|------|-------|--|

Table 6-70.

| texture → effects → blur | | |
|--------------------------|---------------|---|
| start_value | 12 | Controls the size of the glow. Play with it and change it to 20 or 30 and see what happens. |
| attenuation | 1.15 | Controls the decay rate of the glow. Play with it and change start_value to 30 and this value to 2.0 to see what happens. Using high values will yield a comic-like effect. |
| texture_size | VIEWPORT_SIZE | This is important, since position and size of the original and blurred objects must match. |
| passes | TWO | Using TWO instead of ONE will increase the glow effect. |

Table 6-71.

| texture → buffers → render_surface_single | | |
|---|---------------|--|
| texture_size | VIEWPORT_SIZE | This is important, since position and size of the original and blurred objects must match. |

The output of the glow effect can be seen in Figure 6-48. With an exaggeration of the glowing parameters, as explained in the parameter tables, the output will look like seen in Figure 6-49.

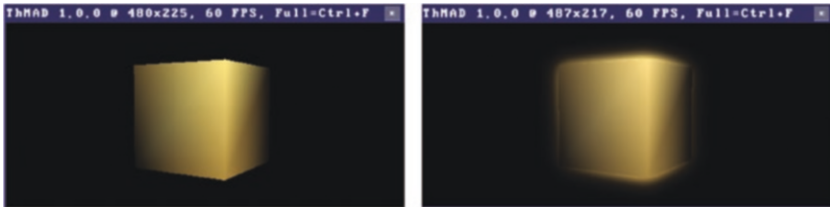


Figure 6-48. Objects with glow, output

The left image shows the original output without the glow.



Figure 6-49. Objects with glow, output for a different set of parameters

The parameter `start_value` of `glow` is set to `20.0` and the anchor attenuation is set to `1.6`.

Summary

This chapter contained a collection of advanced tutorials, or stories covering, more aspects of ThMAD and showing its capabilities. Chapters 7 and 8 contain a GUI reference for the ThMAD Artiste and ThMAD Player.

CHAPTER 7



ThMAD GUI Reference

The chapter is an exhaustive manual of ThMAD GUI operations. All frontend elements of Artiste are described, including the main menu, module choosers, some helper widgets, namely the Assistant and the Inspector, as well as different operation modes and keyboard shortcuts. This is followed by the concepts of saving and loading of states, which describe a visualization pipeline. We talk about what modules are, what types of modules we have, how they can be controlled, and how they can be connected to other modules. The export of states for the purpose of using them later from inside ThMAD Player, the description of macros for defining sub-pipelines, and the introduction of note widgets complete the Artiste section.

ThMAD Player is the part of the ThMAD program suite responsible for presenting finished states, which inside the Player's nomenclature are called *Visualizations*. Inside the Player section, we describe all GUI operation options, more precisely the keyboard shortcuts that can be used to control ThMAD Player.

ThMAD Artiste GUI

Starting and Stopping the GUI

Starting and stopping ThMAD Artiste was described in Chapter 3, “Program Operation”. The rest of this chapter assumes that Artiste is running.

The ThMAD Desktop and Its Parts

An empty state will look like Figure 7-1.

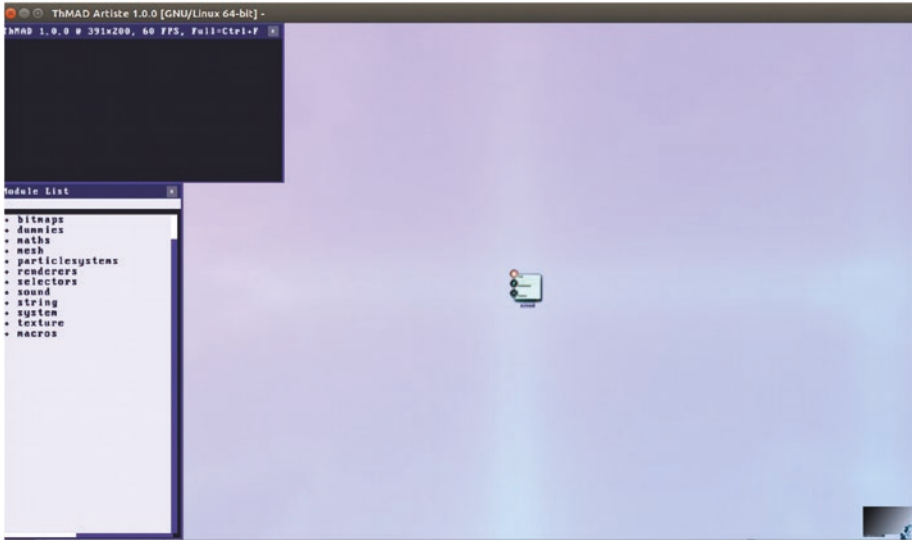


Figure 7-1. *Empty state*

It has a big, plain area where you can drop modules. This area is frequently called the *canvas*. You can see one module at the center—it is the `screen0` module and it is always there. It represents the screen output and cannot be deleted.

At the top-left, the output or preview window is shown; it is black since the state is empty. Underneath the output window, you find the module list. You will use this or the graphical module chooser to select modules for placement on the canvas. At the bottom-right of the Artiste window you can see the minimized assistant. It will help you explore modules or attend courses, described later in this chapter.

To control the view, make sure the canvas has the input focus. It will gain the focus whenever you move the mouse to an empty spot of the canvas. Take this literally—with some user interface actions, the canvas might not have the focus until you move the mouse at least a little bit. If the canvas has the focus, it will appear a bit brighter compared to when it does not have the focus.

Once the canvas has the focus, you can use the keyboard shortcuts listed in Table 7-1.

Table 7-1. Keyboard shortcuts

| | |
|-------------|--|
| S or ← | Pan left |
| F or → | Pan right |
| E or ↑ | Pan up |
| D or ↓ | Pan down |
| W or Page ↓ | Zoom out |
| R or Page ↑ | Zoom in |
| Ctrl+S | Save current state |
| Ctrl+F | Toggle fullwindow mode |
| Ctrl+U | Save current state in memory as an undo point. Many GUI activities lead to an automatic undo point setting and you do not need Ctrl+U. Some activities, namely setting module parameters, do not. If you want to allow for them to be undoable as well, press this key before you perform the action. The list of auto-undo actions is given in the “Menu” section. |
| Ctrl+Z | Do undo, if anything is in the undo buffer. |
| Ctrl+C | Close all open controllers. |
| Ctrl+D | Close all open complex anchors. |
| Ctrl+O | Enable or disable the object inspector. |
| Tab | Change assistant’s size. |
| Escape | Leave the program. If ThMAD detects unsaved changes, you will have to confirm an alert message first. |

With the mouse you can do the actions listed in Table 7-2.

Table 7-2. Mouse actions

| | |
|---------------------|--|
| Just move the mouse | If you’re moving to a resize handler position of the preview window, i.e., an edge or a corner, highlight it. This is for simplifying resizing actions. If you’re moving over the module list, let the module list get the focus. If you’re moving the mouse over the empty canvas, let this one gain input focus. |
| Left-click | If there is a module at that spot, select it and let the module get the focus. If there is no module, remove the focus from the canvas. If the object inspector is enabled, close it if shown. |

(continued)

Table 7-2. (continued)

| | |
|-----------------------------|---|
| Click and drag | If you're on a module, move the module. If you're on an empty spot of the canvas, pan the view. If you're on the head bar of either the preview window or the module list window or a note, move that one. If you're on the edge or corner of either the preview window or the module list window or a note, resize that one. |
| Right-click | If there is a module, module anchor, or connection at that spot, show the corresponding context pop-up menu. Otherwise, show the main menu. |
| Mouse scroll wheel down | Zoom out. |
| Mouse scroll wheel up | Zoom in. |
| Left-click on window closer | Leave the program. If ThMAD detects unsaved changes, you will have to confirm an alert message first. |
| Ctrl+Alt + click and drag | If you're on a module, clone it. Release the mouse key once outside the original module. Cloned modules will inherit all the parameters of the original modules, which can be quite helpful. |
| Ctrl + click and drag | Draw a multi-selection rectangle. All the modules inside get the focus and can be moved or deleted simultaneously. |
| Double-click | If you're at an empty spot, open the graphical module chooser. |
| Ctrl+double-click | Open the graphical module chooser to load a new state. If the current state has unsaved changes, the user must first confirm an alert message. |

Window Modes

ThMAD has a couple of different window modes:

- *Standard desktop mode.* This is the mode you see when ThMAD Artiste is started without any special flags. You get the canvas with preview window and module list, as shown in Figure 7-1.
- *Fullwindow mode.* The canvas is abandoned and the preview window takes the complete window area.
- *Performance mode.* This is a special variant of the desktop mode, where the preview window is closed and instead its contents are used as the canvas background. You can consider it as a mixing of standard and fullwindow mode.

- *Fullscreen mode.* Instead of presenting ThMAD inside an operating system desktop manager's window, ThMAD is forced to cover the complete screen. This mode is entered only if you use special flags when starting ThMAD.

The non-standard modes are described in the following subsections.

Fullwindow Mode

You can switch to the fullwindow mode by entering Ctrl+F. To switch back, press Ctrl+F again. See Figure 7-2.

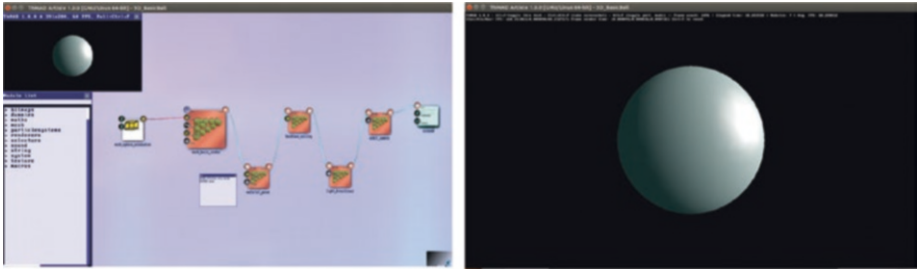


Figure 7-2. Standard and fullwindow mode

If in fullwindow mode, some special keys can be used, as shown in Table 7-3.

Table 7-3. Fullwindow key controls

| | |
|------------|--|
| Alt+T | Toggle header info visibility. This circulates through three states: hidden/small font/big font. |
| Ctrl+T | Reset info aggregation state. |
| Ctrl+F | Exit fullwindow mode. |
| Ctrl+Alt+P | Take a screenshot. You can then find it in the <code>/home/[USER]/thmad/[VERSION]/ data/screenshots</code> folder. |
| Alt+F | Toggle performance mode when inside fullwindow mode. |
| Ctrl+Alt+F | Switch between standard mode and performance mode. Same as pressing Ctrl+F and then Alt+F, |
| Escape | If there are no unsaved changes, quit ThMAD. Otherwise, the fullwindow mode is left and the user asked for a confirmation. |

Performance Mode

After you entered the performance mode from inside the fullwindow mode by pressing Alt+F, or from standard mode by pressing Ctrl+Alt+F, you will see something like Figure 7-3.

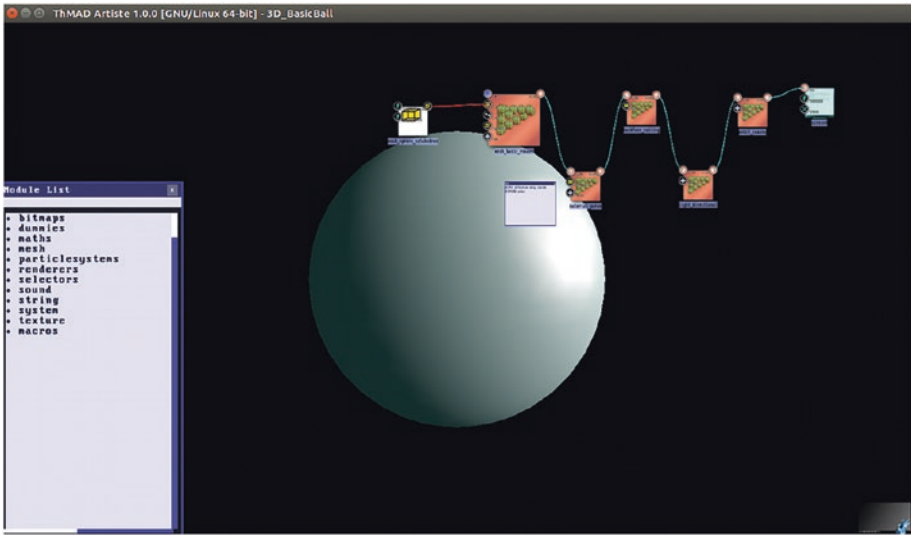


Figure 7-3. Performance mode

You have full control over the state, as in standard mode. To quit the performance mode, press Alt+F or Ctrl+Alt+F again, or leave the fullwindow mode by pressing Ctrl+F.

Fullscreen Mode

Following the description in Chapter 3 and, starting ThMAD in fullscreen mode, you have complete control over the state, as in standard mode. See Figure 7-4.

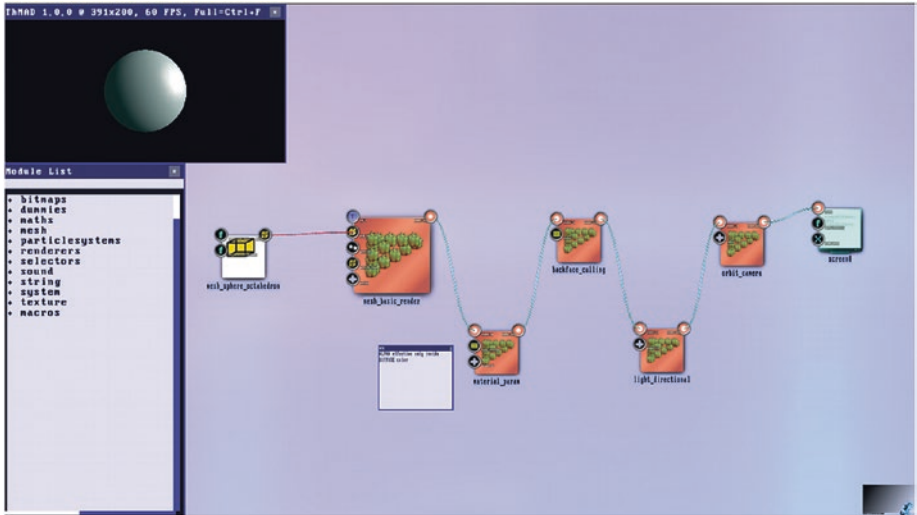


Figure 7-4. Fullscreen mode

Note the absence of operating system window controls.

Of course, you can switch to fullwindow or performance mode there as well using the same keys. The only difference in fullscreen mode compared to the windowed mode and apart from the presentation, is the absent window closer. You have to use the menu or the Escape key to quit ThMAD.

The Main Menu

Right-clicking anywhere on an empty spot of the canvas will show the main menu; see Figure 7-5.

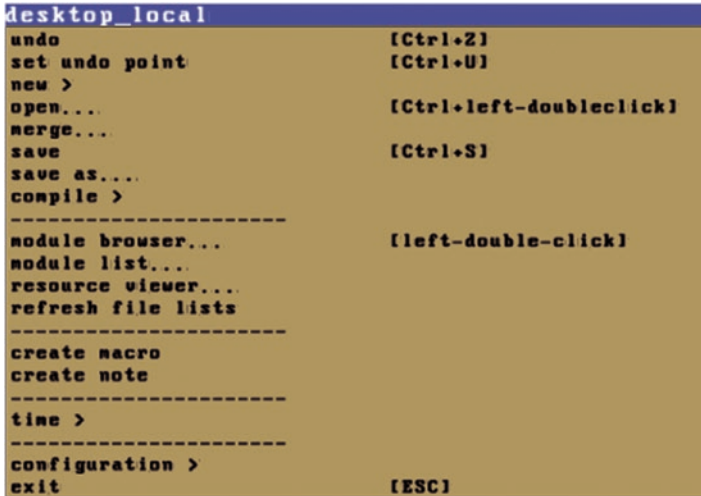


Figure 7-5. The main menu

The main menu functions are listed in Table 7-4.

Table 7-4. Main Menu Functions

| | |
|----------------|---|
| Undo | Undo the last change. The latest from an automatic undo point setting or an explicit undo point setting. Automatic undo points are set by: <ul style="list-style-type: none"> - Loading a new state - Drawing a connection between modules - Deleting a connection between modules - Changing connection order if an anchor has several incoming connections - Creating a macro - Creating a component - Assigning a component to a macro - Deleting a component - Clearing the state (menu: New ► Empty Project) - Loading a template (menu: New ► ...) |
| Set Undo Point | Set an undo point such that by undoing the state can be reverted to this point. This is only needed if no automatic undo point has been set. |

(continued)

Table 7-4. (continued)

| | | |
|-------------|---|--|
| New ► | Empty project | Start with an empty state. If the current state has unsaved changes, the user is asked for confirmation. |
| | Visualization | Start with a visualization project. Places some basic modules and notes on the canvas. If the current state has unsaved changes, the user is asked for confirmation. |
| | Translation for ThMAD Player | This is for starting with a basic fader. If the current state has unsaved changes, the user is asked for confirmation. |
| Open ... | Open a new state or visual from the file system. Will look only inside the <code>/home/[USER]/thmad/[VERSION]/data/</code> folder and present a graphical chooser. If the current state has unsaved changes, the user is asked for confirmation first. Inside the graphical chooser, click and drag to navigate, double-click to select, and right-click to cancel. | |
| Merge ... | Merge a state. Presents the same graphical chooser as in Open ..., but instead of replacing the current state, all the components from the merged state are added to the current state. Works only for states, not for visuals. | |
| Save | Save the current state under its current name. | |
| Save As ... | Save the current state under a new name. Meta information can be given that will be saved along with the state. You can only save as a state here; to save as a visual use the Compile functions. | |
| Compile ► | Music visual (...) As... | Export the current state as a visual into the <code>/home/[USER]/thmad/[VERSION]/data/visuals/</code> folder. The file name will have a <code>.vsx</code> suffix, no matter whether or not you provide one in the dialog. You can add meta information along with the file, if you wish. If a file with the same name already exists, it will be backed up with a time stamp added. |
| | Music Visual Fader (...) As... | Export the current state as a fader into the <code>/home/[USER]/thmad/[VERSION]/data/faders/</code> folder. The file name will have a <code>.vsx</code> suffix, no matter whether or not you provide one in the dialog. You can add meta information, which will be saved along with the file, if you wish. If a file with the same name already exists, it will be backed up with a time stamp added. |

(continued)

Table 7-4. (continued)

| | |
|--------------------------------|--|
| General Package (...) As... | Export the current <i>state</i> as a visual into the /home/[USER]/thmad/ [VERSION]/ data/prods/ folder. The file name will have a .vsx suffix, no matter whether or not you provide one in the dialog. You can add meta information, which will be saved along with the file, if you wish. If a file with the same name already exists, it will be backed up with a time stamp added. |
| Module Browser... | Opens the graphical module browser. You can navigate inside by click and dragging the mouse, close it by right-clicking, or select a module by clicking and dragging the mouse over a module. |
| Module List... | If for whatever reason you closed the window with the module list on the left side of the canvas, you can reopen it by using this menu entry. |
| Resource Viewer... | Investigate the resources inside the /home/[USER]/thmad/ [VERSION]/data/resources/ folder. If you're clicking on a JPG or PNG file, a preview window will be shown. Right-click to close it. |
| Refresh File Lists | Tell ThMAD to refresh its state, visuals, and resources list. Use this if you changed or added files from the outside. Especially useful after you added or changed resources. |
| Create Macro | Create an empty macro. |
| Create Note | Create an empty note. Enter any text you like, place it anywhere you like by clicking and dragging its title bar, close it by clicking on the closer, and resize it by clicking and dragging any edge or corner. |
| Time ► | Rewind Rewinds the sequencing timer to 0.0, then stops it. Play Starts the sequencing timer if not running. Stop Stops the sequencing timer if not running. |
| Configuration ► | Change configuration entries. |
| Exit | Exit ThMAD Artiste. If the current state has unsaved changes, the user is asked for confirmation. |

Module Choosers

Modules are the building blocks of ThMAD—if you want to achieve anything, you need to collect modules and assemble them via connectors. ThMAD has two module choosers—a list with hierarchical structure, present when ThMAD Artiste starts, and a graphical module chooser available on demand.

The Module List

ThMAD Artiste by default shows the module list; see Figure 7-6.



Figure 7-6. Module list

The modules themselves are described throughout this book, and you can find a reference in Chapter 8. Here we talk about usage patterns.

The list window will show up to the left whenever ThMAD Artist starts. To move it, click somewhere near the middle of the Module List title bar, keep the mouse button pressed, and move the mouse to a suitable place. This is what we call dragging. To resize it, find a position near any edge or corner and drag it. Unfortunately in version 1.0 of ThMAD there is no graphical feedback for finding the right position where to start the dragging—you just have to try. The inner contents can be shifted around by dragging one of the two white scrollbars to the right or at the bottom. Sliding up and down can be achieved by using the mouse scroll wheel.

The modules are presented in a hierarchical structure; in Figure 7-6 you see just the top level. To dive deeper into the structure, just click on any label. To fold a structure, click on the label again. Once you're deep enough, you can see the modules, and with the mouse hovering over any of them, an information tooltip will pop up. See Figure 7-7.

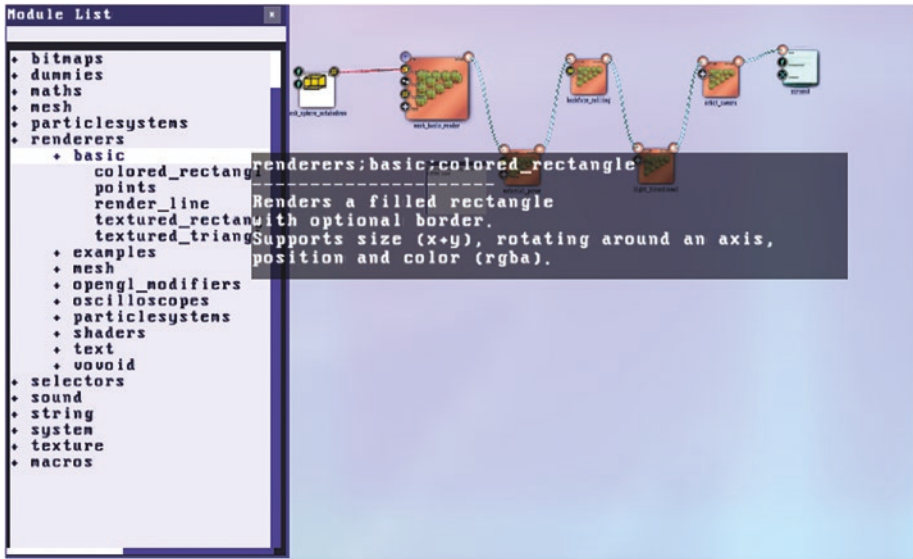


Figure 7-7. Module list unfolded and the mouse hovering over a module label

Once you find a module you need, you can place it by dragging the label onto the canvas, as shown in Figure 7-8.

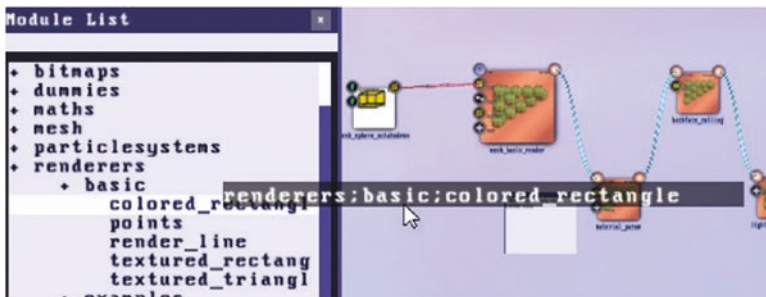


Figure 7-8. Module list, choosing a module

Between the title bar and the thicker black bar, you see the filter in Figure 7-8. Click on it and enter some characters, e.g. rect. The complete hierarchy is filtered to show only items with rect in the label; see Figure 7-9.

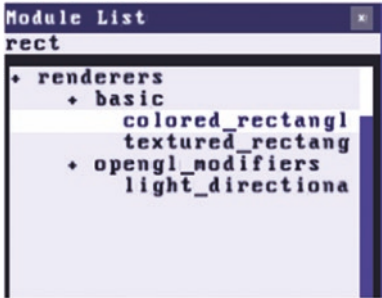


Figure 7-9. Module list, filtering

You can disable the filter to switch back to the complete view by removing the `rect` using Backspace or Del, or all at once by pressing Ctrl+Del.

The module list can be closed by clicking on the symbol at the right edge of the title. To reopen it, use the Module List... menu.

The Graphical Module Chooser

The graphical module chooser presents the module hierarchy in a hyperbolic tree. It can be started by double-clicking on an empty spot of the canvas. See Figure 7-10.

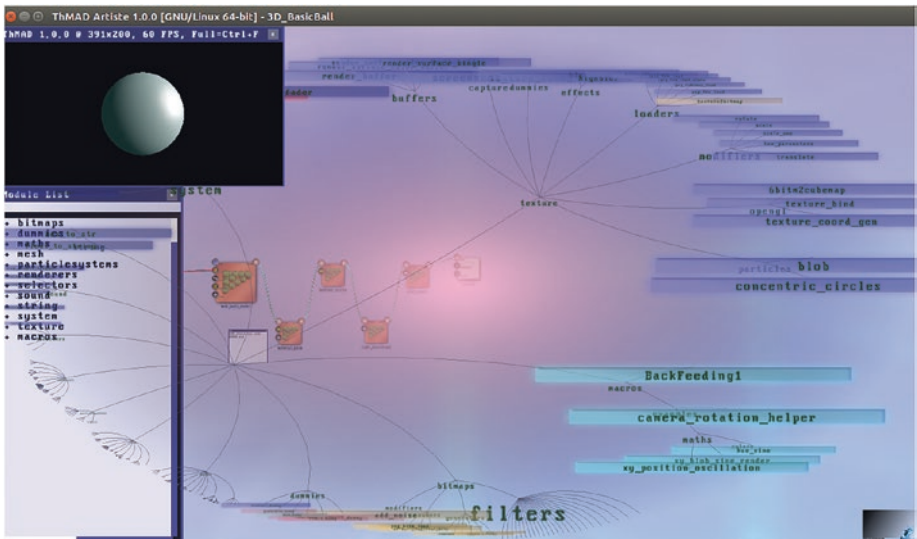


Figure 7-10. Graphical module chooser

You can navigate inside the module chooser by clicking and dragging it. If you hover over a module, the module description will show up in a tooltip; see Figure 7-11.



Figure 7-11. Graphical module chooser, tooltip pop-up

Once you found a module you need, drag it to an empty spot of the canvas. In most cases, clicking on a module, holding the mouse key, moving the mouse a little bit and then releasing the mouse key will do.

If you're doing this with the module chooser, leave it by right-clicking it.

The Assistant

The Assistant helps in understanding what a module does, and it presents a couple of small introductory courses and an overview of mouse and keyboard actions you can use.

The most important thing to know about the Assistant is that you can cycle through different sizes by pressing the Tab key.

Once in the size you like, right-click it to choose the operating mode; see Figure 7-12.



Figure 7-12. The Assistant's operating modes

See Table 7-5 for details.

Table 7-5. Assistant modes

| | |
|----------------------|--|
| Automatic Mode | Click on any module or module anchor to see information about it inside the Assistant. |
| Courses | Choose one of: <ul style="list-style-type: none"> – Introduction course: A basic manual on how to use ThMAD – Performance mode: A few words about the performance mode – Macros: Information about macros |
| Keys/Mouse Reference | Presents a concise list of most of the keyboard and mouse commands. |

If the Assistant shows more than one page in any mode, you can click on any of the page controls at its bottom: , or you can click anywhere inside the Assistant to page forward.

The Assistant cannot be disabled; if you do not need it, just press Tab a couple of times to make it appear in its smallest size.

The Object Inspector

The Object Inspector gives a more technical view of modules and anchors, and it allows you to rename module aliases to improve state readability; see Figure 7-13.

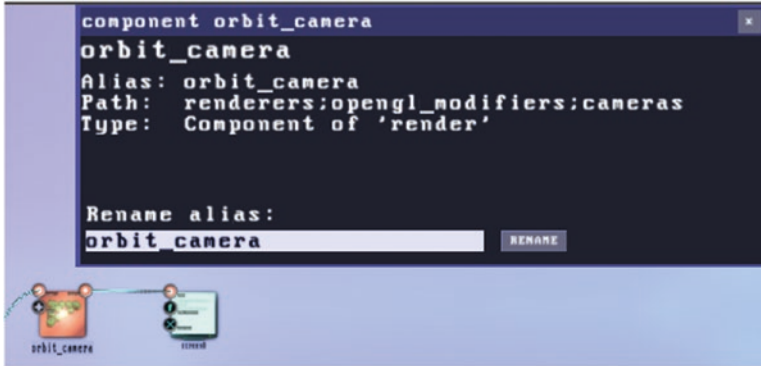


Figure 7-13. *The Object Inspector*

You can toggle the Object Inspector's visibility by pressing Ctrl+O, and you can by default enable it inside the configuration. For the latter, choose the configuration menu.

To rename something, enter a unique name in the editor field and press the Rename button.

Saving and Loading States

Unless you changed the configuration, upon startup ThMAD Artiste will load a state named `_default`, but you can change that to have the last saved state loaded inside the configuration menu.

Once inside Artiste, you can load any state, visual, or fader by Ctrl+double-clicking on an empty spot of the canvas, or by using the Open... menu after you open the main menu (right-click on an empty spot of the canvas).

The graphical file chooser will show up, and there you can choose a file by double-clicking it. To navigate, click and drag inside it, and, to cancel the operation, right-click. See Figure 7-14.

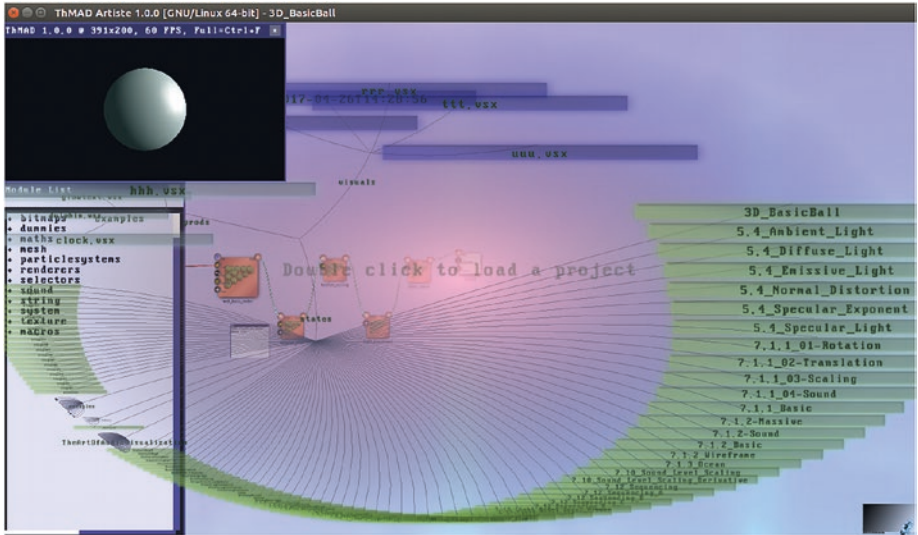


Figure 7-14. The file chooser

Loading a state from inside the States folder will give you the full control over it. Loading a compiled visual is different—once you loaded one you will get the message that ThMAD continues to operate in Detached mode. Just looking at it will be fine, but if you change it, you should save it as a state with Save As... to not risk losing changes.

To save a state under its current name, press Ctrl+S or use the Save menu. To save a new state or save the current state under a different name, use the Save As menu.

After you click Save As, the dialog shown in Figure 7-15 will appear. There you must at least enter a file name; all other entries will be saved along with the state and are mandatory. Note that states in ThMAD are considered to be internal and hence usually do not have a file suffix.

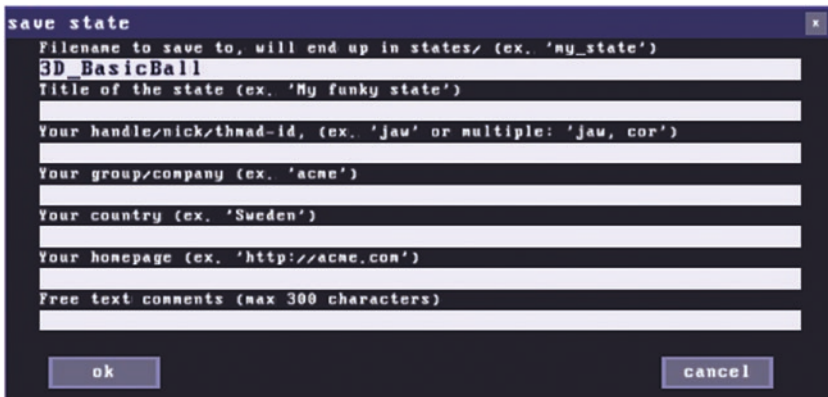


Figure 7-15. The Save As dialog

To save a state as a visual instead, use the Compile > menu.

Modules

Modules tell ThMAD what to show, how to show it, and how it gets controlled.

Module Types

ThMAD has the modules types listed in Table 7-6.

Table 7-6. *ThMAD Module Types*



This represents the physical screen. There is always exactly one instance of this module type and it cannot be deleted.



A renderer type module. Any module that produces renderable output, that means produces image data and could be connected to the screen, will be of this type.



A string type module. String manipulation—you will not use it too often.



An audio related module.



Modules related to textures. Textures are image data living on the graphics hardware. This makes them powerful and fast.



A system type module. Includes time control, sub-pipeline control, file system access, joystick control, and programmatic shutdown.



Modules related to particle systems use this symbol.

(continued)

Table 7-6. (continued)

Maths and parameter control modules use this icon.



Mesh related modules. Includes mesh generators and modifiers.



A macro. Used for grouping modules.



Signifies a bitmap related module.

Modules have anchors where parameters can be specified or other modules can be connected. Anchors are described later in this chapter.

Placing and Deleting Modules

Working with Artiste means placing, or instantiating, modules, deleting them, and connecting them.

To place modules on the canvas, use the module list on the left of ThMAD's window or the graphical module chooser, which opens after Ctrl+double-clicking an empty spot of the canvas. Or you can use the Module List or Module Browser menu entries.

Modules can be shifted freely around on the canvas, but of course it makes sense to place modules that belong to a logical grouping or need to be connected to each other nearby. To shift a module, click and drag on it. To shift several modules at once, proceed as follows: with Ctrl pressed, click at the top-left corner of a rectangle containing the module to displace, then move the mouse to the bottom-right corner. Release the click and the keyboard keys, then click and drag any of the modules to choose a new position for the module group.

To delete a module, click on it and then press the Del key. Or right-click on it and select the Delete menu item. To delete several modules at once, you can draw a rectangle around them. With Ctrl pressed, click at the top-left corner of a rectangle containing the module to delete, then keep them pressed and move the mouse to the bottom-right corner. Release the keys, then press Del to delete them all.

Connecting Modules

Modules need to be connected one or the other way to allow for data moving between them. The connectors or anchors themselves are described later in this chapter, but if they are seen from a state composition point of view, ThMAD imposes two simple rules—only anchors of the same type can have connections between them, and incoming connections are always at the left side of the module, outgoing connections at the right side.

This makes composing states easier. Look for example at Figure 7-16: the screen module has three inputs of three different types. The upper one is of type *renderer* and it must be connected to a module that has an output of type *renderer*.

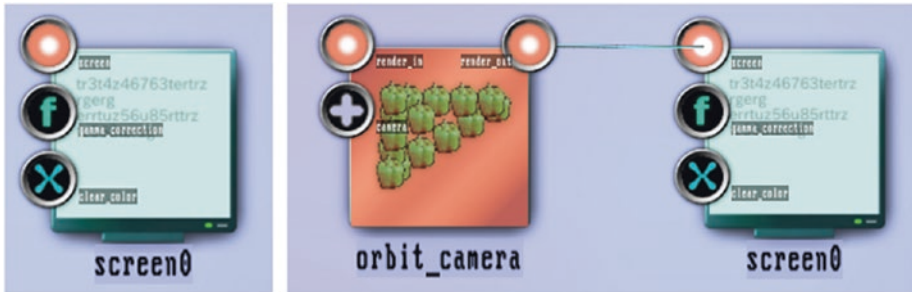


Figure 7-16. Connecting modules

The same principle holds for the other anchors. In this case, they can be connected to other module's output anchors of the same type, but it is not necessary to connect them to other modules.

Cloning Modules

Modules can be cloned, i.e. copied with their properties, by holding the Ctrl+Alt keys and dragging the module to an empty space of the canvas.













Module Anchors: Parameters and Connectors

Module anchors are the eyes, ears, and the voice of modules. In order for modules to talk to other modules, anchors need to be connected to other anchors. But they also serve as parameter knobs to control the functioning of modules. It is one of the main principles of ThMAD that many anchors can do both, depending on whether they are connected.

Look for example at the middle input anchor of the screen module in Figure 7-16. It is called *gamma_correction*, and if it's used as a parameter, it will allow you to configure exactly that property of the screen, the gamma correction. However, you can also connect it to an output anchor of the same type of another module, then you allow another module to control the gamma correction of the screen. As for the screen module, it is unimportant if this control happens dynamically, i.e., it changes over time, or statically and does not change.







ThMAD's modules have anchors of the types shown in Table 7-7.

Table 7-7. Anchor types

| | |
|---|---|
|  | A complex anchor. This is a container holding other anchors as children. It is an exception to what was said previously, for it can never be connected to other anchors directly nor serve as a module parameter holder by itself. To see or hide the child anchors, click on them. Of course the child anchors then can be controlled or connected to other anchors, unless they are complex anchors themselves. |
|  | A bitmap anchor. Used for exchanging bitmap data between modules. |
|  | An enumeration anchor. Used to control enumeration entries like, for example, "yes" and "no". As of ThMAD 1.0 cannot be connected to other modules. |
|  | Type float; a float number like 0.0 or 3.14 or -0.0002. |
|  | Type float3; a group of three float numbers, used very often for point or vector coordinates in three dimensions. |
|  | An array of float or float3 values. |
|  | Type float4; a group of four float numbers used very often in conjunction with color values: RED, GREEN, BLUE, and ALPHA. |
|  | An integer value. ThMAD mostly everywhere uses float numbers, so you will not encounter this type. |
|  | An array of float values. |
|  | A mesh describing 3D objects. |
|  | A particle system. |
|  | A quaternion describing rotations in space. |

(continued)

Table 7-7. (continued)

| | |
|---|--|
|  | Signifies rendering data. You will find this very often, as it is the backbone of rendering pipelines. |
|  | Describes a resource, like a file. Mostly used for image files like PNG or JPEG files, but also for fonts. |
|  | A segmesh. This is considered a highly special feature in ThMAD. You find some modules using it, but it is not subject of this book. |
|  | A sequence of numbers. There are currently no outputs of this type so this is an anchor of parameter controller type only. |
|  | A string. Used for drawing text, but also for providing shader programs and alike. |
|  | A texture; represents bitmap data on the graphics hardware. |

All complex anchors can be folded at once by using the Ctrl+D key; all open controllers can be closed by pressing Ctrl+C key with the mouse over an empty spot of the canvas.

Drawing Connections Between Anchors

To connect an anchor of one module to an anchor of the same type of another module, just click on either of the anchors, move the mouse to the other partner, and then release the mouse button when over it.

When you connected a child of a complex anchor and folded it, it will look as if you drew a connection to the complex anchor; see Figure 7-17.

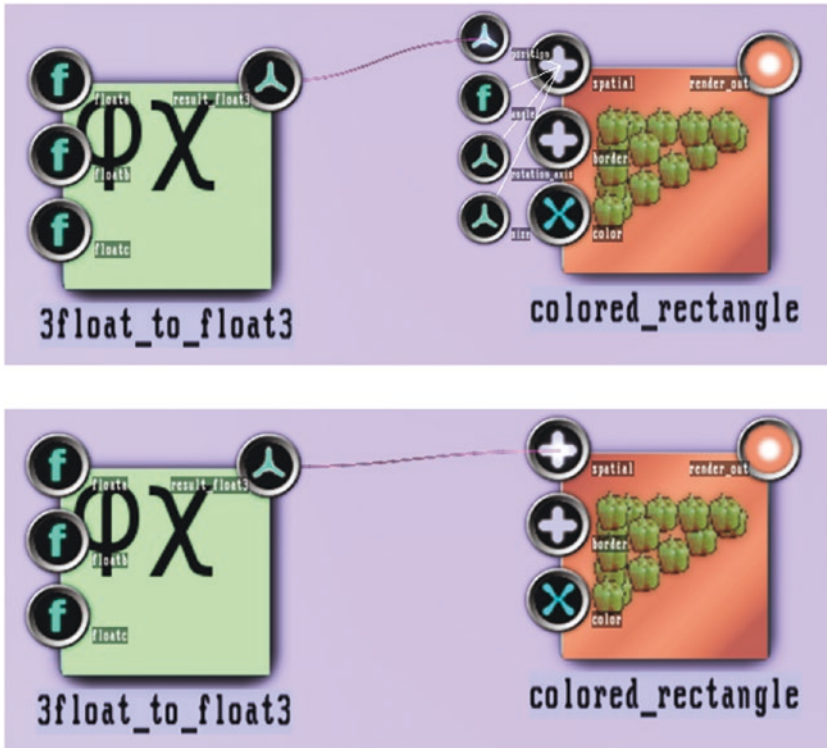


Figure 7-17. Anchor connections with complex anchors

However, this is *not* the case, complex anchors cannot be connected directly. To see where the connection really goes, you have to unfold the complex anchor by clicking on it.

To delete a connection, move the mouse close to the line, then right-click and select the Disconnect pop-up menu. See Figure 7-18.

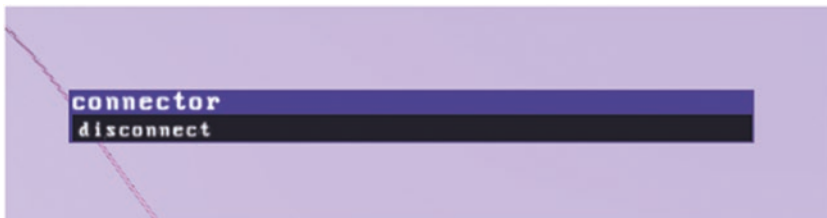



Figure 7-18. Removing an anchor connection


■ **Caution** To delete a connection going to an anchor inside a complex anchor, you have to unfold the complex anchor first by clicking on it.

Enumeration Input as Module Parameter

If not connected to another module, an enumeration type input anchor  can have its value set by clicking on it and selecting the appropriate entry in the pop-up menu.


Some enumerations are misused as actions to reset something or do something else once. Setting them to the Action value will then automatically be reverted once the action has been performed. It is up to the module whether it uses such enumerations; details are included in the module reference in Chapter 8.

Float Input as Module Parameter

If not connected to another module, a float type input anchor  can have its value set by double-clicking on it and then using the controller.

Float anchors are assigned a default controller, which is usually but not necessarily a single knob. To see the available controllers and choose an appropriate one, right-click on the anchor and select one of the options listed in Table 7-8.

Table 7-8. *Float number input*

| | | |
|------|---|---|
| Knob |  | <p>A knob controller. To use the mouse for adjusting the value, click somewhere near the center and drag (move the mouse while holding down the mouse button) up/down for coarse adjustments and left/right for fine adjustments. Additionally press the Shift key to snap values at multiples of 0.1.</p> <p>To change the value directly, click on the ciphers and use the keyboard to enter a value.</p> <p>Clear the field by pressing Ctrl+Del. Press Enter/Return when done.</p> <p>To move the knob on the canvas, click and drag on the black area.</p> <p>To close the knob, double-click on it.</p> |
|------|---|---|

(continued)

Table 7-8. (continued)

Slider



A slider controller. Use this to define a slider operating inside a range of float values.

The upper knob defines the extent of float values, the lower knob is the lowest value. Each knob can be controlled exactly in the way as described for the single knob controller.

The slider value can be adjusted by dragging the slider handle. Or you can enter the value using the keyboard by clicking on the ciphers above the scale and entering values as described for the single knob controller.

To move the slider on the canvas, click and drag on one of the black areas. Double-click anywhere on the slider to close it.


Direct



Direct value input using the keyboard. Click on the ciphers and use the keyboard to enter a new value and then confirm by pressing Enter or Return. Enter Ctrl+Del to clear the field first.






Move it by dragging the black area. Close it by double-clicking on it.

Float3 Input as Module Parameter


If not connected to another module, a float3 type input anchor  usually denoting a coordinate set in 3D can be adjusted by double-clicking on it. A couple of controllers are available, but they usually show up as a default controller.

All controller types available after right-clicking are listed in Table 7-9.

Table 7-9. *Float3 input*


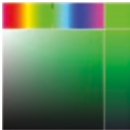

| | | |
|-----------|---|--|
| Sliders |  | <p>A triple slider. If the anchor is used for coordinate setting, the three sliders are for the x, y, and z coordinates, respectively.</p> <p>Each of the upper knobs defines the extent of float values, the corresponding lower knob is the lowest value. Each knob can be controlled exactly in the way a single knob controller is controlled.</p> <p>The slider value can be adjusted by dragging the slider handle. Or you can enter the value using the keyboard by clicking on the ciphers above the scale and entering values. To move the slider on the canvas, click and drag on one of the black areas. Double-click anywhere on the slider to close it.</p> |
| Axes View |  | <p>A graphical representation of a rotation, seen as a point on the unit sphere. The x-axis is white, the y-axis violet, the z-axis green, and positive directions thicker.</p> <p>This control is extremely powerful for defining orientations where the norm of the vector is supposed to be 1.0, but it can be dangerous in other cases. Of course, you can try it to find out whether it makes sense to use this control.</p> <p>To adjust the orientation, click and drag anywhere near the center and move the mouse in any direction. Move it on the canvas by dragging the black area near the corners, and close it by double-clicking on it.</p> |
| Color |  | <p>This is of limited use here, since the fourth color value, the ALPHA, cannot be set here and hence usually float3 anchors do not correspond to a color value. You can use it nevertheless if you want to control the three values of the float3 type inside the range [0;1.0].</p> <p>To close it, double-click the box in the top-right corner.</p> |
| Pad |  | <p>Using the pad you can only control the first two values of the float3 type; the third one will always be set to 0.0. Use the knobs to define the limits (set them like any other knob). Click with the mouse at any point to set the two coordinates.</p> <p>Move it on the canvas by dragging the black area; close it by double-clicking on it.</p> |
| Direct |  | <p>Use this if you want to enter all three values directly. Click on each of the numbers and use the keyboard to enter a new value, confirm by pressing Enter or Return. Enter Ctrl+Del to clear the field first.</p> <p>Move it by dragging the black area. Close it by double-clicking on it.</p> |

Float4 Input as Module Parameter


Float4 parameters usually define color values. Unless connected to another module, a float4 type input anchor  can be adjusted after double-clicking on it. A couple of controllers are available, but the color slider will typically show up as the default controller.

Right-clicking will allow you to use any of the options in Table 7-10.

Table 7-10. *Float4 input*




| | | |
|---------|---|--|
| Sliders |  | <p>A triple slider. If the anchor is used for coordinate setting, the three sliders are for the x, y and z coordinates, respectively.</p> <p>Each of the upper knobs defines the extent of float values; the corresponding lower knob is the lowest value. Each knob can be controlled the same way as the single knob controller.</p> <p>The slider value can be adjusted by dragging the slider handle. Or you can enter the value using the keyboard by clicking on the ciphers right the scale and entering values. To move the slider on the canvas, click and drag on one of the black areas. Double-click anywhere on the slider to close it.</p> |
| Color |  | <p>Very useful for entering color values and most of the time the default controller for float4 values. Any of the four values is constrained to lie inside the range [0;1.0].</p> <p>To set a value, first click and drag inside the top HUE chooser, then click anywhere inside the big SATURATION/VALUE box underneath it to finish choosing RGB (or HUE-SATURATION-VALUE). To choose the ALPHA value, click and drag inside the bottom-right rectangle. Highest means an ALPHA of 1.0; lowest an ALPHA of 0.0.</p> <p>Any drag operation is fully smooth, meaning you see immediate changes in the output while moving the mouse. To move the control on the canvas, click and drag on the top-right box. Double-click on the top-right box to close it.</p> |
| Direct |  | <p>Use this if you want to enter all four values directly. Click on each of the numbers and use the keyboard to enter a new value. Confirm by pressing Enter or Return. Enter Ctrl+Del to clear the field first.</p> <p>Move it by dragging the black area. Close it by double-clicking on it.</p> |

Quaternion Input as Module Parameter

Unless connected to another module, a quaternion type input anchor  can be adjusted after double-clicking on it. A couple of controllers are available, but the quadruple slider will typically show up as the default controller.

Right-clicking will show any the options in Table 7-11.

Table 7-11. Quaternion input

| | | |
|-----------|---|--|
| Sliders |  | <p>A quadruple slider. Represents the x, y, z, w values. Each of the upper knobs defines the extent of float values, the corresponding lower knob is the lowest value. Each knob can be controlled exactly in the way as the single knob controller is.</p> <p>The slider value can be adjusted by dragging the slider handle. Or you can enter the value using the keyboard by clicking on the ciphers above the scale and entering values.</p> <p>To move the slider on the canvas, click and drag on one of the black areas. Double-click anywhere on the slider to close it.</p> |
| axes view |  | <p>Quaternions inside ThMAD usually represent rotations. So does the axes view control.</p> <p>The component order is: white, violet, green. To adjust the rotation or quaternion representing the rotation, click and drag anywhere near the center and move the mouse in any direction.</p> <p>Move it on the canvas by dragging the black area near a corner, and close it by double-clicking on it.</p> |
| direct |  | <p>Use this if you want to enter all four values x, y, z, and w, directly. Click on each of the numbers and use the keyboard to enter a new value. Confirm by pressing Enter or Return. Enter Ctrl+Del to clear the field first.</p> <p>Move it by dragging the black area. Close it by double-clicking on it.</p> |

String Input as Module Parameter

If not connected to another module, a string type input anchor  can be adjusted by double-clicking on it.

A text editor appears; see Figure 7-19.



Figure 7-19. String input text editor

You can enter any text there; however, the editing capabilities are limited. You cannot enter Tabs and select text ranges for cut and paste or copy and paste. The To Clipboard and From Clipboard buttons at least allow you to copy the text from your favorite editor back and forth.

When done you must use the Save button to make changes permanent, then click Close to close the editor. Clicking only on Close instead will discard any changes, so be careful with that.

Resource as Module Parameter

A resource type input  can be set by double-clicking on it.

The resource viewer will show up, as shown in Figure 7-20.

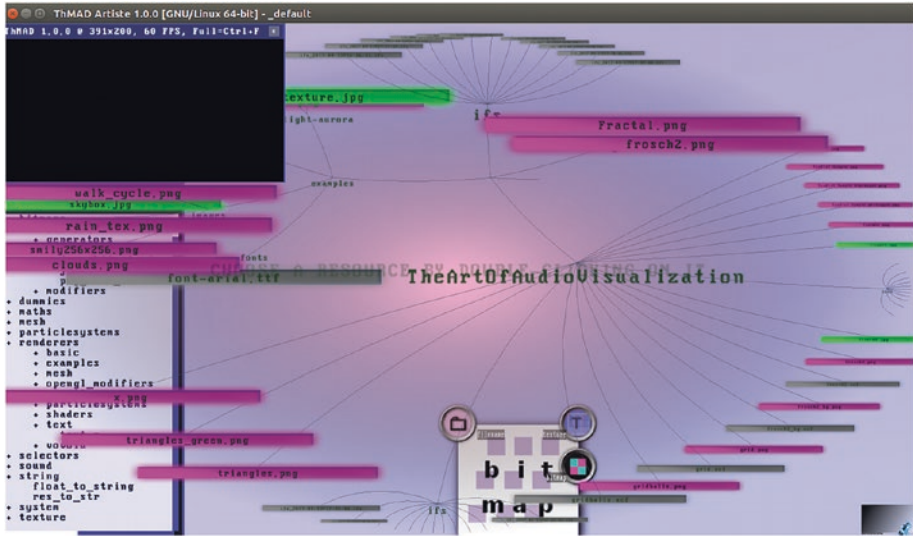


Figure 7-20. Resource selector

Navigate inside the resource viewer by click and dragging starting at any spot away from the colored boxes. Choose a file by double-clicking on it, or cancel the selection operation by right-clicking.

Note that if you changed resources from outside ThMAD, you need to tell ThMAD to refresh its file lists. You do so by choosing Refresh File Lists from the main menu.

Sequence Input as Module Parameter

This control is used by modules that need an x-y-sequence, say float value versus time or similar. A sample is the input parameter float_sequence module, Maths ► oscillators ► float_sequencer, where the sequence of this oscillator’s output values is defined.

Once the corresponding parameter editor is opened, either by double-clicking on the anchor or by right-clicking the context menu, the image in Figure 7-21 appears.

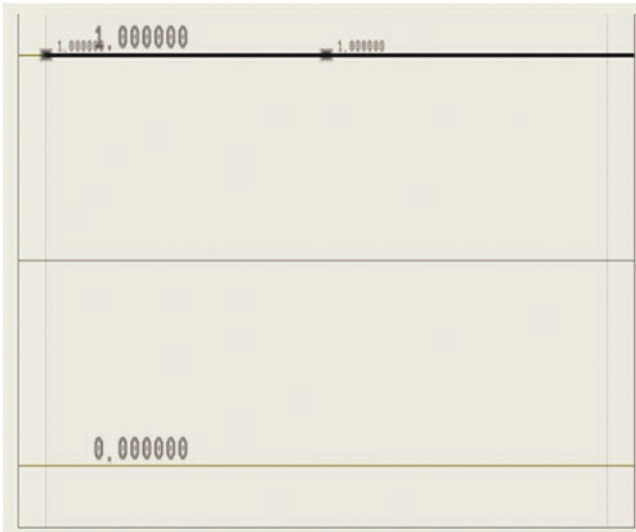


Figure 7-21. Sequence editor

This is a sequence with all values equaling 1.0. The anchors of the sequence are the small rectangles at the start and middle of the line. At the beginning we just have two anchors. You can freely move anchors by left-clicking on them and then dragging.

To add new anchors, just click anywhere on the line and hold down the Shift key. To remove an anchor, right-click it with the Shift key pressed. The result might look like Figure 7-22.

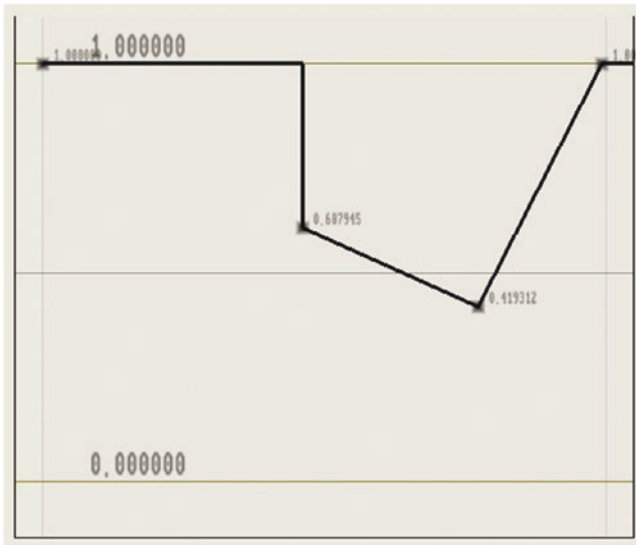


Figure 7-22. Sequence editor with anchors shifted and added

At each point, the *interpolation type* determines how it is connected to the neighbor point to the right. The default is no interpolation, meaning an anchor will abruptly force a change to its value exactly at its position, not earlier. This can be seen at the second anchor in Figure 7-22. Because we up to now added anchors via Shift+click, those anchors will show a linear interpolation to their right neighbor, which means the values change linearly from one point to the next. You can change the interpolation type by right-clicking on an anchor and choosing what you want.

Interpolation types are:

- *No interpolation:* The value stays constant until the x coordinate of the next point is reached and then abruptly changes its value. See Figure 7-23.

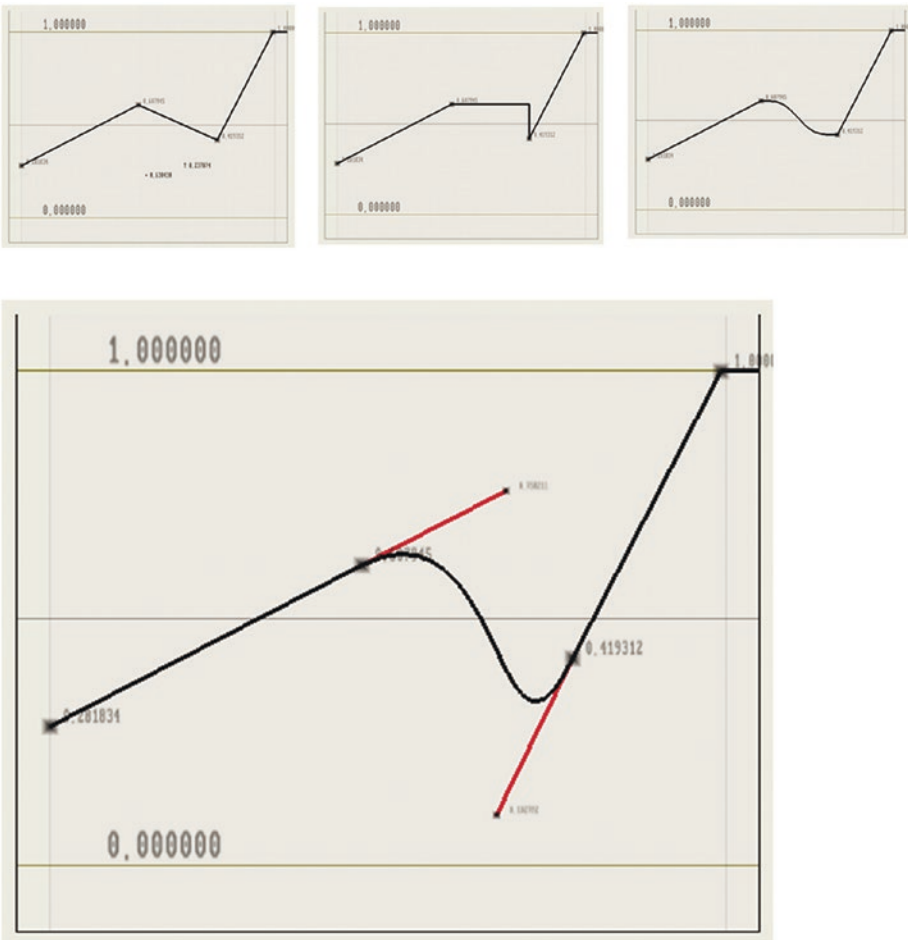


Figure 7-23. Different sequence editor anchor interpolations
 Linear, none, cosine, and Bezier. The Bezier anchors at the end of the line extension have been chosen to make the segment match smoothly.

- *Linear interpolation*: The default. Linearly changes from one point to the next. See Figure 7-23.
- *Cosine interpolation*: A smooth line with a cosine for function arguments 0 or π , up to π or 2π , depending on whether the value decreases or increases. Note that the slope at the segment starts and ends at zero. See Figure 7-23.
- *Bezier interpolation*: A Bezier curve, allowing for smooth inter-segment connections. Once chosen the Bezier segment will have two more anchors added for the Bezier parameters, which you can select and click and drag. Those extra handles are small, so make sure you zoom in to see them. See Figure 7-23.

Internally, for performance reasons, all points of a sequence are stored in an array of 8192 x-y pairs.

Exporting States

Once you are finished with a state, you can save it into an unmodifiable file unit called a *visual*. If state files live inside this folder:

```
/home/[USER]/thmad/[VERSION]/
data/states/
```

The visuals as for ThMAD Artist’s matters live in one of these folders:

- /home/[USER]/thmad/[VERSION]/data/visuals/
- /home/[USER]/thmad/[VERSION]/data/prods/
- /home/[USER]/thmad/[VERSION]/data/faders/

Between the `visuals/` and the `prods/` folder, there is just a logical distinction; into `visuals/` you export files that you later want to use for ThMAD Player, while `prods/` is more a general purpose export folder. The `faders/` folder is for special fader state exports you are going to use for transitions inside the ThMAD Player. The internal file structure is the same.

The procedure to perform the export is described in the “Main Menu” section under **Compile** ►.

Macros

Macros are containers for grouping modules. Unless expanded, they appear as a single symbol on the canvas and you thus can use them to hide details in your state, improving readability. In addition, macros can be saved separately from the state and you can reuse them in other states.

To create a macro, use the main menu that pops up after right-click an empty spot of the canvas. Click on the **Create Macro** item.

Activities are listed in Table 7-12.

Table 7-12. *Macro actions*

| | |
|---|---|
| Open the macro | Right-click on the macro, then choose Open/Close from the pop-up menu. |
| Close the macro | Same as open the macro. |
| Move the macro | If the macro is closed, simply click and drag it, like any other module. If the macro is open, clicking an empty spot of the macro. |
| Delete a macro | Click the macro, then press the Del key. Or right-click the macro and select the Delete [del] item. |
| Rename a macro | Make sure the Object Inspector is enabled (you can toggle visibility by pressing Ctrl+O while over an empty spot of the canvas). Click on the macro and then use the Rename functionality in the Object Inspector. |
| Clone a macro | Just as for modules, press and hold Ctrl+Alt and drag the macro to an empty space of the canvas. Cloning a macro will clone all its constituent modules as well! |
| Resize a macro | Right-click and choose Double Size or Half Size from the menu that appears. This only applies to an opened macro (you can resize with the macro closed, but the symbol for the closed macro will not change the size). |
| Modify/control modules inside the macro | Open the macro, then do the same as you'd do if the component was outside the macro. |
| Move a single module to the macro | Click Ctrl+Shift on the keyboard, then drag the module using the left mouse button to the opened macro (will not work for closed macros!). Once inside the macro, a module can only be removed from the macro by deleting it. You cannot move it outside (pressing Ctrl+Z for Undo will work though). |
| Move several modules at once to the macro | Select several modules. Either click while holding the Ctrl key at the same time, or draw a rectangle around the modules while pressing the Ctrl key (press Ctrl, then click and hold the left-top corner, move to the bottom-right corner, then release the mouse button, then release Ctrl). Once selected, press Ctrl+Shift and drag one of the selected modules to the opened macro (will not work for closed macros!). Once inside the macro, modules can only be removed from the macro by deleting them; you cannot move them outside (pressing Ctrl+Z for Undo will work though). |

No matter whether you moved a single module to the macro, or several modules at once, incoming and outgoing connections will show up as artificial anchors at the edge of the macro. See Figure 7-24.

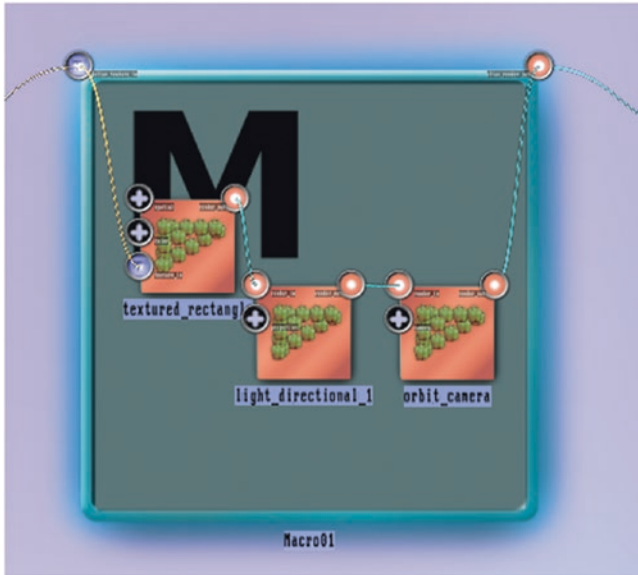


Figure 7-24. Macro anchors

Further actions are listed in Table 7-13.

Table 7-13. Extended macro actions

| | |
|---|---|
| Expose an anchor from a module inside to the external world | To expose internal anchors (e.g., you have a module inside with input anchor xyz and you want to control it from outside the macro), drag the anchor to an empty spot inside the macro. |
| Remove macro anchors | Right-click on a macro anchor, then select Unalias from the menu. |
| Save a macro | Macros can be saved for later reuse by right-clicking on the macro, then selecting Save Macro... from the pop-up menu. The saved macro will appear in the module lister after the next ThMAD restart. |
| Insert a saved macro into the current state | Drag from the Macros section of the module lister as you'd do for normal components. |

Saved macros can only be deleted from outside ThMAD. Navigate to the following folder and remove the appropriate files here:

```
/home/[USER]/thmad/  
[VERSION]/data/macros/
```

Notes

Notes help to document your states or parts of them. To create a note, click mouse-right at an empty spot of the canvas, and in the pop-up menu, select Create Note. Add any text there. See Figure 7-25.

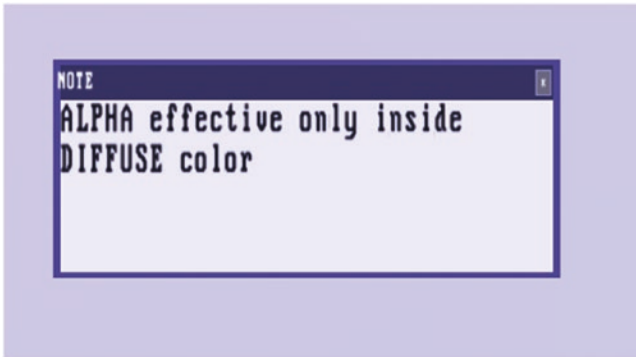


Figure 7-25. A note

To move a note, drag its title bar. To remove a note, click on its close box in the top-right corner. To change its size, click and drag any edge or corner. (Unfortunately, there is no user interface feedback for finding the right mouse position; you just have to try.)

Note that you cannot add notes to macros, and that notes do not take part in a multi-select (Ctrl-click to draw a rectangle). Also, nodes cannot be cloned like modules.

ThMAD Player

Starting and Stopping the GUI

For information on starting and stopping the ThMAD Player, see Chapter 3. The rest of this chapter assumes you have the ThMAD Player running.

Player GUI Operations

The Player does not react to mouse movements, but you can use a couple of keys on your keyboard to control it.

The keyboard operations listed in Table 7-14 are available while the Player is running.

Table 7-14. *ThMAD Player Controls*

| | |
|-------------------|---|
| O | Toggle text overlay's visibility. If enabled, an info is printed shortly when visuals switch. The default is on, unless disabled by program invocation. |
| F1 | Show help. Automatically enables text overlay; press O later to disable it. |
| F | Show status information. Automatically enables text overlay; press O later to disable it. |
| Cursor left/right | Back to previous or forward to next visual in the list. |
| Cursor up/down | Increase or decrease sensitivity to sound input. |
| Page up/down | Make faster/slower. Controls time-scaling factor. |
| R | Enable/disable randomization. If disabled, stay at one visual. If enabled, randomly switch between the visuals after random time intervals. |
| Ctrl+R | Randomly choose another visual from the list. |
| Escape | Quit the Player. |

Summary

In this chapter, you learned about all GUI functionalities of ThMAD Artiste and ThMAD Player. You saw how to use the keyboard to control everything, how to identify, place, and connect elements, and how to use the parameter controls, which can be used to adjust module parameters of different types. You also investigated different window modes and helper widgets inside Artiste. You learned about the file handling, and how Artiste and Player cooperate.

The next chapter provides a complete reference of all modules available for creating visualizations.

CHAPTER 8



ThMAD Module Reference

In ThMAD, the modules are the building blocks for visualizations. In this chapter, we provide a complete module reference, with a description of what each module does and which parameters it has. By its structure it also describes what you can do using the software. Scanning over the chapter and reading it thoroughly will help you to get some impressions and new ideas. The module category names and the module names closely relate to the GUI and the implementation code, with an emphasis on the GUI so you can easily make your own experiments using the GUI.

All modules have zero or more output anchors representing data of various types, and zero or more input anchors representing data of various types. Some modules have side effects, i.e., they store something or do something that is not reflected in the output data. Data types are:

- `int`: An integer number.
- `float`: A floating point number.
- `double`: A double precision float. All floating point numbers in ThMAD are of type single precision float, so this double type, although defined, is not used.
- `float3`: A vector of three floating point numbers. For example, 3D coordinates.
- `float4`: A vector of four floating point numbers. For example, the RED, GREEN, BLUE, ALPHA color values.
- `quaternion`: Also a vector of four floating point numbers, but with *quaternion* calculation rules. Quaternions represent a 3D rotation operation accurately.
- `string`: A string of characters.
- `matrix`: a 4x4 matrix of floats. Matrices are used to describe spatial operations like rotation, scaling, and translation.
- `float_array`: An array of a variable number of float values.
- `float3_array`: An array of a variable number of float3 vectors.

- `quaternion_array`: An array of a variable number of quaternion data.
- `resource`: Points to a resource file, maybe a picture or some text file a module can read from or write to.
- `enum`: An enumeration. Used at places, where a string value from a fixed collection has to be chosen. Enums also are used for one-time actions like resetting or triggering something.
- `sequence`: An array of 8192 float values. Used by modules that need to have some development of a float value specified, usually in the time domain. Differs from a `float_array` by having a special graphical editor in the ThMAD Artiste GUI.
- `render`: An internal queue that represents drawable items.
- `texture`: Represents a bitmap of colored points inside the graphics hardware.
- `mesh`: A collection of points defining the surface of a 3D object. May represent faces, i.e., two dimensional surface atoms with connected edges, but this is not necessarily the case.
- `bitmap`: A number of colored points under custody of the CPU, which means not directly handled by the graphics hardware. Only by a suitable renderer module sent to the graphics hardware.
- `particlesystem`: A possibly large number of equal or similar 3D objects. Handled internally and only via some renderer sent to the graphics hardware.
- `segment_mesh`: A special form of a mesh with unconnected edges. Usually converted to a mesh before handled further.

In the module reference tables of this chapter, all nodule control anchor specifications will be written as:

Parameter-Name : Parameter-Type

Where the parameter names do not contain spaces or hyphens. Longer names inside the tables will break at an underscore, so if you see something like `gamma_correction:float`. What is actually meant is `gamma_correction:float`.

Each module belongs to a category, which will be reflected by the position of each module inside the module lister and module browser. The module categories are as follows:

- **Screen**: The output to the graphics hardware.
- **Bitmaps**: Colored pixel stores residing in the main memory and handled by the CPU.

- **Dummies:** Modules usually passing through values unaltered. Used for example if you want to control one float value connected to several input anchors at once. This category actually only contains aliases to various other categories' dummies.
- **Maths:** Mathematical operations like operators, with all input and output types the same, and functions with different input and output types. Here you'll also find the oscillators.
- **Mesh:** Comprise a collection of points defining surfaces or surface parts of 3D objects, whether connected or unconnected.
- **Particlesystems:** A possibly large number of equal or similar 3D objects. Used for rain, fog, and a lot of special effects.
- **Renderers:** Convert different kinds of input to an output anchor that may directly connect to the screen. Also contain modules that alter other modules' renderer output.
- **Selectors:** Used to fetch a single element or a subrange of elements from a collection of some data type.
- **Sound:** Handles sound input.
- **String:** Handles strings.
- **System:** A collection of system relevant modules like time, system status, or file chooser related modules.
- **Texture:** Handles bitmaps that reside on the graphics hardware. Using textures instead of bitmaps yields a huge performance boost. Textures are also important for back propagating pixel data, i.e., feeding a sub-pipeline output back to one of its input constituents.
- **Macros:** Reusable sub-pipelines.

Screen

This actually contains a single built-in module `screen0`, obligatory for graphics output. Any module must somehow eventually be connected to the screen; otherwise, it will not be active. So it is the last module in the pipeline or set of pipelines of a visualization setup and responsible for the visual output on the monitor.

screen0

Output to graphics hardware, graphics card, and monitor. See Table 8-1.

Table 8-1. *The screen0 module*

| screen0 | | The screen |
|----------------|------------------------|--|
| In | screen:render | Here any visualization pipeline must be connected to. Nothing can ever be seen if this is unconnected. If several pipelines connect to this anchor, the rendering order may be quite important. To change the order in the ThMAD Artiste GUI, double-click on the anchor and drag the anchor connectors. |
| | gamma_correction:float | Gamma correction. Defaults to 1.0. The gamma correction stems from the fact that human perception of visual brightness is nonlinear, and from old cathode ray tubes that show nonlinear behavior by themselves. Usually a gamma greater than 1 makes shadows appear darker and a gamma less than 1 makes shadows appear lighter. |
| Out | None | Will render to the graphics hardware. |

Bitmaps

Bitmaps are images living inside ThMAD, not inside the graphics hardware, so it is possible to perform operations on them before they are uploaded to the graphics hardware.

Filters

The bitmap filters all involve two bitmaps and all have the same structure. The path is Bitmaps → Filters, then see Table 8-2.

Table 8-2. *Bitmap filter modules*

| Name Dynamically Assigned | | |
|---------------------------|----------------------|---|
| In | bmp1:complex | The first bitmap |
| | in1:bitmap | Bitmap anchor; needs to connect to a bitmap producer |
| | bitm1_ofs: float3 | This is an offset to the bitmap pixel coordinates applied before the blending happens. The third coordinate is ignored |
| | bmp2:complex | The second bitmap |
| | in2:bitmap | Bitmap anchor; needs to connect to a bitmap producer |
| | bitm2_ofs: float3 | This is an offset to the bitmap pixel coordinates applied before the blending happens. The third coordinate is ignored |
| | bitm2_opacity: float | Defines a mixing factor, values are from [0;1]. With 0, the output will be B1. With 1, the output will be the blending result. All other values interpolate linearly. |
| | target_size: float3 | The output bitmap size. The third coordinate is ignored. |
| | set_target_size:enum | Use this to set the size to a square like one of: 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, and 2048x2048. |
| | blend_type: enum | The blending type—determines the formula to be applied when doing the blending. The available modes are listed in this chapter. Note that all the different bitmap filters will just use different settings at this anchor. |
| | bitmap_type: enum | One of integer or float. Unused. |
| Out | bitmap:bitmap | The output bitmap. You can add more filters here or send this to any module that needs a bitmap as input. |

All available blending modes are listed in Table 8-3. If sample images are given, we use







transparent), and





as bitmap B1 (the white parts at the edge as bitmap B2 (an ALPHA channel is added, but reads 1.0 everywhere).

Table 8-3. *Bitmap filter blending modes*

| Name of the Filter Module | Corresponding “blend_type” Anchor setting | Description |
|---------------------------|---|--|
| bitm_blend_add | BLEND_ADD | <p>For each RGBA value, add B1 and B2.</p>  |
| bitm_blend_average | BLEND_AVERAGE | <p>Clamp to max. 1.0. Blending the images will yield the image shown here. The white color on top-left comes from the ALPHA there <i>not</i> applied before doing the blending! So even though ALPHA is 0 there, the hidden RGB values (white here) will be taken into account!</p> <p>For each RGBA value, take the average $(B1+B2) / 2$.</p>  |
| bitm_blend_color_burn | BLEND_COLOR_BURN | <p>For each RGBA value, divide inverted B1 by B2, invert the result, and clamp to [0.0;1.0]. Burns in the color of B2 to B1. The result for the sample images is shown here; the darker B1, the more its color is used.</p>  |
| bitm_blend_color_dodge | BLEND_COLOR_DODGE | <p>For each RGBA value, divides B1 by inverted B2. The brighter B2, the more it affects the color of the result. No part of B1 will be darkened in the result.</p>  |





(continued)

Table 8-3. (continued)

| Name of the Filter Module | Corresponding “blend_type” Anchor setting | Description |
|---------------------------|---|--|
| bitm_blend_darken | BLEND_DARKEN | For each RGBA value, take the smaller of B1, B2.  |
| bitm_blend_difference | BLEND_DIFFERENCE | For each RGBA value, take the absolute difference $ B1 - B2 $. In our example, we will only get ALPHA values where B1 has an ALPHA $\neq 1.0$, i.e., in its surroundings. With B2’s ALPHA not 1.0 everywhere, you get more interesting results.  |
| bitm_blend_exclusion | BLEND_EXCLUSION | The formula for each of the RGBA values is: $B1 + B2 - 2 \cdot B1 \cdot B2$ |
| bitm_blend_glow | BLEND_GLOW | For each of the RGBA values, take the square of B2 and divide by the inverse of B1. The square will emphasize lighter parts of B2, but for darker parts the light parts of B1 gain influence.  |
| bitm_blend_hard_light | BLEND_HARD_LIGHT | Same as BLEND_OVERLAY, but with B1 and B2 exchanged.  |

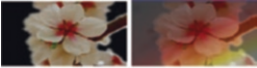





(continued)

Table 8-3. (continued)

| Name of the Filter Module | Corresponding “blend_type” Anchor setting | Description |
|---------------------------|---|---|
| bitm_blend_hard_mix | BLEND_HARD_MIX | For each RGBA value, where B1 is darker than B2 is lighter, set 0. Otherwise, set 1.  |
| bitm_blend_lighten | BLEND_LIGHTEN | For each RGBA channel, take the bigger of B1, B2.  |
| bitm_blend_linear_burn | BLEND_LINEAR_BURN | Same as BLEND_SUBTRACT. |
| bitm_blend_linear_dodge | BLEND_LINEAR_DODGE | Same as BLEND_ADD. |
| bitm_blend_linear_light | BLEND_LINEAR_LIGHT | For each RGBA value, if $B1 < 0.5$, do a linear burn with $2 \cdot B1$ and $B2$; otherwise, a linear dodge with $2 \cdot (1.0 - B1)$ and $B2$.  |
| bitm_blend_multiply | BLEND_MULTIPLY | For each RGBA value, take $B1 \cdot B2$.  |
| bitm_blend_negation | BLEND_NEGATION | Because of the multiplication, where parts of both B1 and B2 are dark, the result will be even darker. For each RGBA value, take the inverse of: the absolute value of: $(1.0 - B1) - B2$ |



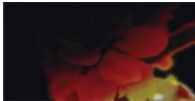

(continued)

Table 8-3. (continued)

| Name of the Filter Module | Corresponding “blend_type” Anchor setting | Description |
|---------------------------|---|--|
| bitm_blend_normal | BLEND_ NORMAL | For each RGBA value, mix B2 to B1 according to the bitm2_opacity anchor value. Following 0.0, 0.5 and 1.0:   |
| bitm_blend_overlay | BLEND_ OVERLAY | For each RGBA value, where $B2 < 0.5$, take twice the multiplied value, $2 \cdot B1 \cdot B2$. Otherwise, take the inverse of twice the inverse multiplied value, $1.0 - 2 \cdot (1.0 - B1) \cdot (1.0 - B2)$.  |
| bitm_blend_phoenix | BLEND_ PHOENIX | For each RGBA value, take the lower of B1, B2, and add the inverse of the higher of B1, B2.  |
| bitm_blend_pin_light | BLEND_ PIN_LIGHT | For each RGBA value, if $a < 0.5$, do a blend darken of $2 \cdot B1$ and $B2$; otherwise, do a blend lighten of $2 \cdot (1.0 - B1)$ and $B2$.  |
| bitm_blend_reflect | BLEND_ REFLECT | Same as BLEND_ GLOW, but with B1 and B2 exchanged. The sample shows some artifacts because of the division.  |

(continued)

Table 8-3. (continued)

| Name of the Filter Module | Corresponding “blend_type” Anchor setting | Description |
|---------------------------|---|---|
| bitm_blend_screen | BLEND_SCREEN | Same as blend multiply, but take the inverses of B1 and B2, and from the result take the inverse again. Where a normal multiply intensifies dark regions in both bitmaps, this mode intensifies light regions in both bitmaps.  |
| bitm_blend_soft_light | BLEND_SOFT_LIGHT | A softened version of BLEND_HARD_LIGHT.  |
| bitm_blend_subtract | BLEND_SUBTRACT | Like addition, but lets RGBA values refer to 0.5 as a zero point. You can thus achieve a subtraction if values are below 0.5. Clamps the result to min. 0.0.  |
| bitm_blend_vivid_light | BLEND_VIVID_LIGHT | For each RGBA value, if $B1 < 0.5$, take a color burn of $2 \cdot B1$ and $B2$; otherwise take a color dodge of $2 \cdot (1.0 - B1)$ and $B2$.  |

Both images are PNGs with WHITE as the assigned background color. Note that all blend filters treat the color components RED, GREEN, BLUE, and ALPHA independently, and that they all act pixel by pixel without mixing different pixels.

The resulting bitmap will be sent to the graphics hardware as a texture acting on a quad with vertex colors all white.

Note that you can use the same bitmap for B1 and B2 to blend an image with itself.

Generators

Bitmap generators serve as a source for generating bitmaps without loading external image files. The path is Bitmaps → Generators, then see Tables 8-4 to 8-9.

Table 8-4. *The blob module*

| blob | Generates a blob |
|--------------------|---|
| In settings | |
| arms:float | If you want to make a star or flower, set a value > 0. The value 0 means: no arms. |
| attenuation: float | Controls the sharpness of the shape. The less, the blurrier. |
| star_flower: float | Controls the density of the center part. If > 0, makes arms thinner near the center (as for a flower). |
| angle:float | Use this to rotate the shape (makes sense only if it has arms). |
| color:float4 | Defines the color of the blob. |
| alpha:enum | Set to yes or no. If yes, use the ALPHA channel for the density transition; otherwise apply ALPHA already here. In the latter case the bitmap will have ALPHA = 1.0 everywhere. |
| size:enum | One of 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048. Defines the pixel size of the bitmap. |
| Out bitmap:bitmap | The resulting bitmap. |

Table 8-5. *The concentric_circles module*

| concentric_circles | Creates concentric circles |
|---------------------------|--|
| In settings | |
| frequency: float | Controls the density of circles to generate |
| attenuation: float | Controls the sharpness of the shape. The less, the blurrier |
| color:float4 | Defines the color |
| alpha:enum | Set to yes or no. If yes, use the ALPHA channel for the density transition; otherwise apply ALPHA already here. In the latter case, the bitmap will have ALPHA = 1.0 everywhere. |

Table 8-6. *The perlin_noise module*

| perlin_noise | A perlin noise bitmap |
|-----------------------|--|
| In | |
| perlin_options | |
| rand_seed: float | With the integer part changing, create noise of another shape. |
| perlin_strength:float | The intensity of the noise. |
| size:enum | One of 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048. Defines the pixel size of the bitmap. |
| octave:enum | The blurriness of the noise. The higher, the less blurry. Values range from 1 to 16. |
| frequency: enum | Defines the grain. Values range from 1 to 8-. |
| bitmap_type: enum | One of integer or float. Unused. |
| blob_settings | |
| enable_blob: enum | Set to yes or no, whether to blend into a blob. |
| arms:float | If you want to make a star or flower, set a value > 0. |
| attenuation: float | Controls the sharpness of the shape. The less, the blurrier. |
| star_flower: float | Controls the density of the center part. If > 0, makes arms thinner near the center (as for a flower). |
| angle: float | Use this to rotate the shape (makes sense only if it has arms). |
| color:float4 | The color of the noise. |
| alpha:enum | Set to yes or no” If yes” use the ALPHA channel for the density transition; otherwise apply ALPHA already here. In the latter case, the bitmap will have ALPHA = 1.0 everywhere. |
| Out | |
| bitmap:bitmap | The resulting bitmap. |

Table 8-7. *The plasma module*

| | | |
|---------------|-----------------|---|
| plasma | | Generates a deterministic, non-random regular plasma. Color values are calculated VIA Linear function of: $\sin(\text{linear function of } x) \cdot \sin(\text{linear function of } y)$ |
| In | settings | |
| | col_amp: float4 | Color-component-wise multiplier of the outer linear function |
| | col_ofs: float4 | Color-component-wise additive of the outer linear function |
| | period: complex | Contains color-component-wise float3 anchors for the multiplier of the linear functions inside the $\sin()$ terms (the third component of each float3 is unused, since we need only x and y coordinates) |
| | ofs: complex | Contains color-component-wise float3 anchors for the additives of the linear functions inside the $\sin()$ terms (the third component of each float3 is unused, since we need only x and y coordinates) |
| | size: enum | Bitmap size, one of: 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, or 1024x1024. |
| Out | bitmap: bitmap | The result bitmap |

Table 8-8. *The solid module*

| | | |
|--------------|----------------|--|
| solid | | Generate a monochrome bitmap with the same color for every pixel. You can use this for filters for colorizing bitmaps |
| In | color: float4 | The color |
| | size: enum | Bitmap size one of: 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 |
| Out | bitmap: bitmap | The result bitmap |

Table 8-9. *The subplasma module*

| subplasma | | Generates a random plasma. Much less artificial compared to the plasma module |
|------------------|-----------------|--|
| In | rand_seed:float | With the integer part changing, create a subplasma of another shape |
| | size:enum | Bitmap size, one of: 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, or 1024x1024 |
| | amplitude: enum | Defines the grain of the subplasma. the higher, the finer. Available values are 2, 4, 8, 16, 32, 64, 128, 256, and 512 |
| | color:float4 | The color of the subplasma |
| Out | bitmap:bitmap | The result bitmap |

Loaders

Loads bitmaps from files. The path is Bitmaps → Loaders. Then see Tables 8-10 to 8-12.

Table 8-10. *The jpeg_bitm_load module*

| jpeg_bitm_load | | Load a JPG file as bitmap. The file must be inside the /home/[USER]/thmad/ [VERSION]/data/resources folder |
|-----------------------|--------------------|---|
| In | filename: resource | Points to the file inside the resources folder. |
| Out | bitmap:bitmap | Output bitmap. |
| | texture:texture | Directly sends a texture to the graphics hardware after loaded. |

Table 8-11. *The jpeg_bitm_load_alpha module*

| jpeg_bitm_load_alpha | | Loads two JPG files. The second one defines the ALPHA channel. Both files must be inside the /home/[USER]/thmad/ [VERSION]/data/resources folder |
|-----------------------------|-------------------------|---|
| In | filename_rgb: resource | The RGB data |
| | filename_alpha:resource | The ALPHA data; use a gray-value bitmap to specify the ALPHA value |
| Out | bitmap:bitmap | Output bitmap |
| | texture:texture | Directly send as texture to the graphics hardware after loaded |

Table 8-12. *The png_bitm_load module*

| png_bitm_load | | Load a PNG file as bitmap. The file must be inside the /home/[USER]/thmad/ [VERSION]/data/resources folder. |
|----------------------|-----------------------------------|--|
| In | Filename: resource reload:enum | Points to the file inside the resources folder. An action enum. Click on yes to tell ThMAD to reload the file. |
| Out | bitmap:bitmap texture:texture | Output bitmap Send as texture to the graphics hardware after loaded |

Modifiers

Modification of bitmaps. The path is Bitmaps → Modifiers → add_noise. See Table 8-13.

Table 8-13. *The add_noise module*

| add_noise | | Add noise to a bitmap. Note that this is an expensive operation, because the bitmap needs to be recalculated often |
|------------------|--|---|
| In | bitmap_in:bitmap time_rate:float noise_amount: float | Input bitmap Time interval in milliseconds for applying noise. How much noise to add: 1.0 maximum amount, 0.0 nothing |
| Out | bitmap:bitmap | Output bitmap |

Dummies

Dummies either do nothing or pass through values without changing them. They are used for technical reasons or for multiplexing controllers. For example, modules A and B need a float input, and you want the corresponding anchors to be subject only to manual changes, i.e., using the GUI, and in addition synchronized. They should always receive the same controller value. Without dummies, you'd have to change them one after the other. To avoid this monotonous work, you can use a dummy and connect it to A and B; see Figure 8-1.

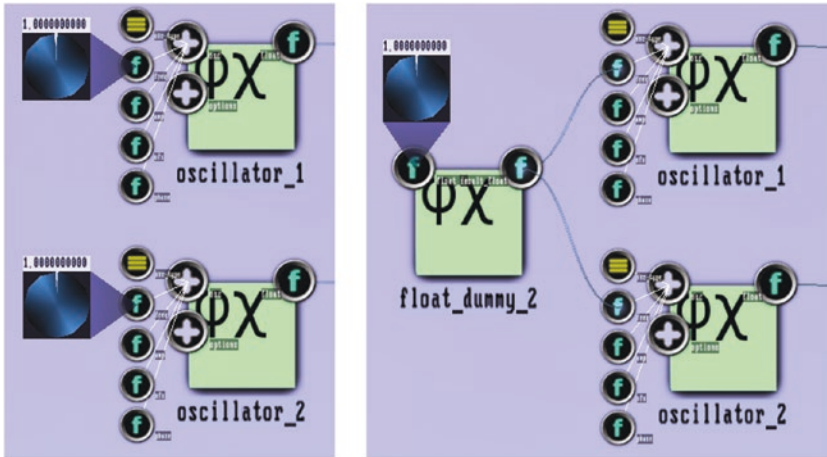


Figure 8-1. Using dummies to simplify manual controlling

On the left side of the figure, you need to adjust two controllers if you want to synchronize the input. Using the dummy shown on the right allows you to do it in one step.

Dummies are situated at various places in the module hierarchy—the Dummies module section provides aliases for dummies from different other module sections; see Table 8-14.

Table 8-14. Dummy module aliases

| Name | Description | Is An Alias To |
|------------------|--|------------------------------------|
| float_dummy | Multiplexes a float value | Maths → Dummies → float_dummy |
| float3_dummy | Multiplexes a float3 vector | Maths → Dummies → float3_dummy |
| float4_dummy | Multiplexes a float4 vector, mostly color values | Maths → Dummies → float4_dummy |
| quaternion_dummy | Multiplexes a quaternion | Maths → Dummies → quaternion_dummy |
| mesh_dummy | Multiplexes a mesh input | Mesh → Dummies → mesh_dummy |
| texture_dummy | Multiplexes a texture input | Texture → Dummies → texture_dummy |

Math Modules

Math modules are important for controlling other modules. They do not produce any graphical output.

Accumulators

Accumulators accumulate input values, i.e., they hold an internal value that is updated each frame by means of an input value provided. The initial value of each is the neutral element of the operation, for addition 0.0 or an array with 0.0 valued elements, for a quaternion multiplication (0.0; 0.0; 0.0; 1.0).

If a `reset:enum` input anchor is used, this means that setting its value to `ok` will reset the accumulator once to its initial state. This anchor will be deactivated automatically the next frame, so you have to set it to `ok` again if you want to repeat the action.

The path is `Maths` → `Arithmetic` → `Accumulators`, then see [Table 8-15](#).

Table 8-15. *Accumulator modules*

| Name | Type | Parameter Names | Description |
|---------------------------------------|---|--|--|
| <code>float_accumulator</code> | <code>float</code> → <code>float</code> | <code>float_in</code> → <code>result_float</code> | Adds the input value to the accumulator each frame. |
| <code>float_accumulator_limits</code> | <code>float</code> → <code>float</code> | <code>float_in</code> , <code>limit_lower</code> , <code>limit_upper</code> → <code>result_float</code> | Adds the input value to the accumulator each frame. If the accumulator value ever is beyond one of the limits, it is clamped to the corresponding limit. |
| <code>float3_accumulator</code> | <code>float3</code> → <code>float3</code> | <code>float3_in</code> → <code>result_float3</code> | Adds the input <code>float3</code> value component-wise to the accumulator each frame. |
| <code>float4_accumulator</code> | <code>float4</code> → <code>float4</code> | <code>float4_in</code> → <code>result_float4</code> | Adds the input <code>float4</code> value component-wise to the accumulator each frame. |

(continued)

Table 8-15. (continued)

| Name | Type | Parameter Names | Description |
|--------------------------------|-------------------------------|-----------------------------------|---|
| quat_ rotation_ accum_2d | float x float → quaternion | param_x, param_y → result_quat | <p>In ThMAD quaternions are represented by (x, y, z; w), meaning its special real part sits at the end of the array.</p> <p>The param_x and param_y build multiplier quaternions (0,x,0; 1) resp. (0,0,y; 1) for the first and second multiplication or rotation to be applied each frame:</p> $\text{accum} = \text{accum} \cdot (0,x,0; 1)$ $\text{accum} = \text{accum} \cdot (0,0,y; 1)$ <p>Because of the special properties of quaternions a multiplication of this form can describe any rotation.</p> |

Arithmetic

This category is about basic arithmetical operations.

Unary Operators

An operation in → out with all types the same.

In case of Boolean operators, input and output are mapped floats. An input inside [-0.5;0.5] means FALSE, an input less than -0.5 or greater than +0.5 means TRUE, an output of 1.0 means TRUE and an output of 0.0 means FALSE. The path is Maths → Arithmetic → Unary. Then Table 8-16 gives an overview.

Table 8-16. *Unary operators*

| Name | Type | Parameter Names | Description |
|------------------|--------|--|---|
| cos | float | float_in → result_float | Cosine function out = cos(in) |
| sin | float | float_in → result_float | Sine function out = sin(in) |
| tan | float | float_in → result_float | Tangent function out = tan(in) |
| acos | float | float_in → result_float | Arcus cosine function out = arccos(in) |
| atan | float | float_in → result_float | Arcus tangent function out = arctan(in) |
| abs | float | float_in → result_float | Absolute value, removes the sign out = abs(in) |
| not | float | a → result_float | Boolean NOT out = not(in) |
| derivative | float | float_in → result_float | Derivative function out = in - last_in Since the module will be called each 1/60 second, that means this is an approximate derivative in time, $1/60 \cdot d / dt$ |
| ifinside | float | float_in → result_float parameters: low, high val_inside, val_outside | Checks whether the input lies inside the range [low;high], and if so, outputs val_inside. Otherwise outputs val_outside. |
| vector_normalize | float3 | param1 → result_float3 | Normalizes a float3 vector, i.e. changes its length to 1.0 while keeping the direction: $v \rightarrow v / \ v\ $ |

Binary Operators

An operation $in1 \circ in2 \rightarrow out$ with all types the same. The path is Maths → Arithmetic → Binary → * → *.

Table 8-17 gives a list; note that in case of Booleans, input and output are mapped floats: an input inside [-0.5;0.5] means FALSE, an input less than -0.5 or greater than +0.5 means TRUE, an output of 1.0 means TRUE and an output of 0.0 means FALSE.

Table 8-17. Binary operators

| Name | Type | Parameter Names | Description |
|-------|-------|-----------------------|---|
| mod | float | param1 param2 → mod | Modulus of division out = $in1 - \text{int}(in1 / in2) \cdot in2$ where int() removes the fractional part |
| add | float | param1 param2 → sum | Addition out = $in1 + in2$ |
| sub | float | param1 param2 → diff | Subtraction out = $in1 - in2$ |
| mult | float | param1 param2 → mult | Multiplication out = $in1 \cdot in2$ |
| div | float | param1 param2 → div | Division out = $in1 / in2$ |
| max | float | param1 param2 → max | Maximum out = $in1 > in2 ? in1 : else: in2$ |
| min | float | param1 param2 → min | Minimum out = $in1 < in2 ? in1 : else: in2$ |
| pow | float | param1 param2 → pow | Power out = $in1 ^ in2$ |
| round | float | param1 param2 → round | Rounds out = $\text{round0}(in1 / in2) \cdot in2$ where round0() gives the closest integer (half up for positives, half down for negatives) Example: $in1 = 25.6382,$ $in2 = 100 \rightarrow 25.64$ Example: $in1 = 25.635,$ $in2 = 100 \rightarrow 25.64$ Example: $in1 = -25.635,$ $in2 = 100 \rightarrow -25.64$ |
| floor | float | param1 param2 → floor | Gives the scaled floor out = $\text{floor0}(in1 / in2) \cdot in2$ where floor0() gives the next lower integer Example: $in1 = 25.6382,$ $in2 = 100 \rightarrow 25.63$ |
| ceil | float | param1 param2 → ceil | Gives the scaled ceil out = $\text{ceil0}(in1 / in2) \cdot in2$ where ceil0() gives the next upper integer Example: $in1 = 25.6312,$ $in2 = 100 \rightarrow 25.64$ |

(continued)

Table 8-17. (continued)

| Name | Type | Parameter Names | Description |
|------------------------------|--------------------|----------------------------------|---|
| and | float (boolean) | param1 param2 → and | Boolean AND out = in1 && in2 0,0 → 0 1,0 → 0 0,1 → 0 1,1 → 1 |
| or | float (boolean) | param1 param2 → or | Boolean OR out = in1 in2 0,0 → 0 1,0 → 1 0,1 → 1 1,1 → 1 |
| nand | float (boolean) | param1 param2 → nand | Boolean NAND out = not(in1 && in2) 0,0 → 1 1,0 → 1 0,1 → 1 1,1 → 0 |
| xor | float (boolean) | param1 param2 → xor | Boolean XOR out = (in1 in2) && not(in1 && in2) 0,0 → 0 1,0 → 1 0,1 → 1 1,1 → 1 |
| nor | float (boolean) | param1 param2 → nor | Boolean NOR out = not(in1 in2) 0,0 → 1 1,0 → 0 0,1 → 0 1,1 → 0 |
| float4_ add | float4 | param1 param2 → result | Component-wise addition of two float4 values out _i = in _{1i} + in _{2i} |
| quat_ mul | quaternion | param1 param2 → result | Quaternion multiplication out = in1 † in2 If seen as a rotation in space, the multiplication represents two subsequent rotations. |
| vector_ add | float3 | param1 param2 → result_float3 | Vector addition out _i = in _{1i} + in _{2i} |
| vector_ cross_ product | float3 | param1 param2 → result_float3 | Vector cross product out = in1 x in2 |
| vector_ from_ points | float3 | param1 param2 → result_float3 | Vector from point A to point b, essentially a subtraction of their position vectors out _i = in _{2i} - in _{1i} |

Ternary Operators

An operation $in1 \circ in2 \circ in3 \rightarrow out$ with all types the same. The path is Maths \rightarrow Arithmetic \rightarrow Ternary \rightarrow Float \rightarrow mult_add. There is just one, as shown in Table 8-18.

Table 8-18. Ternary Operators

| Name | Type | Parameter Names | Description |
|----------|-------|---|--|
| mult_add | float | float_in first_mult then_add \rightarrow result_float | First multiply, then add out = (float_in · first_mult) + then_add |

Functions

Functions have one or more input values of one or more types, and one or more output values of possibly different types. The path is Maths \rightarrow Arithmetic \rightarrow Functions. See Tables 8-19 to 8-26.

Table 8-19.

| compare | | Compares two values, fuzzily if checked for equality |
|---------|---------------|---|
| In | float_a:float | First float. |
| | float_b:float | Second float. |
| | operator:enum | One of equals, larger_than or smaller_than. |
| Out | result:float | For equals: Will return 1.0 if $abs(a-b) < 0.00001$, else 0.0 For larger_than: Will return 1.0 if $a > b$, else 0.0 For smaller_than: Will return 1.0 if $a < b$, else 0.0 |

Table 8-20.

| float4_mul_float | | Multiplies all components of a float4 with a given other float |
|------------------|-----------------------|--|
| In | param1:float4 | float4 value |
| | param2:float | float multiplicand |
| Out | result_float4: float4 | The new float4 value: (param1 _i · param2) |

Table 8-21.

| quaternion_to_axis_angle | | Converts a rotation representing quaternion to an axis and angle value describing the same rotation |
|---------------------------------|--|--|
| In | source_quaternion: quaternion | The input quaternion |
| Out | result_axis: float3 result_angle: float | The output axis Rotation angle around the output axis |

Table 8-22.

| axis_angle_to_quaternion | | Converts a rotation around an axis by an angle to a quaternion representation |
|---------------------------------|----------------------------|--|
| In | axis:float3 angle:float | The axis The rotation angle around the axis |
| Out | Result: quaternion | The quaternion representing the same operation |

Table 8-23.

| vector_add_float | | Adds one float to each vector component out_i = in1_i + in2 |
|-------------------------|-------------------------------|--|
| In | param1:float3 param2:float | Vector Single scalar to add to each component |
| Out | result_float3:float3 | The result vector |

Table 8-24.

| vector_dot_product | | Vector dot product out = $\sum_i in1_i \cdot in2_i$ |
|---------------------------|--------------------------------|---|
| In | param1:float3 param2:float3 | First vector Second vector |
| Out | result_float: float | Dot product |

Table 8-25.

| vector_mul_float | | Multiplies a vector by a scalar out_i = in1_i · in2 |
|-------------------------|-----------------------|--|
| In | param1:float3 | The vector to scale |
| | param2:float | The scalar multiplier |
| Out | result_float3: float3 | The result of the multiplication |

Table 8-26.

| float3_rotate_by_quat | | Lets a point vector rotate around a center point with a rotation defined by a quaternion |
|------------------------------|-----------------------|---|
| In | point:float3 | The (position) vector we want to rotate |
| | center:float3 | Rotate around this center point |
| Out | result_float3: float3 | The result of the rotation |

Array

Array related functions. The path is Maths → Array. See Tables 8-27 to 8-29.

Table 8-27.

| float_array_memory_buffer | | A buffer of variable size. Adds the input value according to the mode |
|----------------------------------|---------------------------|---|
| In | float_in:float | The next value to be added the next frame. |
| | size:enum | The size of the buffer. One of 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, or 4096. |
| | mode:enum | If <code>rewind</code> , starts adding at the left array boundary, then increases the position by one each step. When the maximum is exceeded, starts at the left boundary again. If <code>shift</code> , starts adding left, and when the maximum is exceeded, shifts the complete array left and henceforth only adds at the right boundary. If <code>insert_right</code> , always shifts left and always adds at the right array boundary. |
| | ema:float | Can be used for a smoothing of input values. 0.0 means no smoothing, 0.9 means normal, 0.99 means slow smoothing. You do not want to let that value be ≥ 1.0 . |
| Out | result_array: float_array | The current buffer. |
| | cur_index:float | The index of the next float that will be added. |

Table 8-28.

| float_array_pick | | Picks a single value from a float_array |
|-------------------------|---------------------------------------|--|
| In | float_in: float_array which: float | The array to get the value from The index. If the index is out of bounds, the last value returned will be returned again. |
| Out | result_float: float | The value picked. |

Table 8-29.

| float_array_average | | Calculates the average of the array given |
|----------------------------|---|--|
| In | float_in: float_array start: float end: float | The array The start index inside the array The end index inside the array |
| Out | result_float: float | The average. If start and/or en index are fractional, at the edges a weighted value of the adjacent members will be taken into account |

Color

A color is a four dimensional value, or float4, comprising the following:

- RED
- GREEN
- BLUE
- ALPHA, opacity or inverse transparency

with all values being inside [0.0;1.0].

The path is Maths → Color. Functions useful especially for colors are shown in Tables 8-30 and 8-31.

Table 8-30.

| converters → 4f_hsv_to_f4_rgb | | Converts a HSV+A color from the HSV space to an RGBA float4 |
|--------------------------------------|-----------------------|--|
| In | h:float | HUE |
| | s:float | SATURATION |
| | v:float | VALUE |
| | a:float | Alpha |
| Out | result_float4: float4 | The RGBA color |

Table 8-31.

| converters → f4_hsl_to_f4_rgb | | Converts a HSL+A color from the HSL color space to a RGBA float4 |
|--------------------------------------|-----------------------|---|
| In | hsl:float4 | HUE, SATURATION, LIGHTNESS, Alpha |
| Out | result_float4: float4 | The RGBA color |

Converters

Various converters from one data type to another. Converters for color types are not here; find them later in this chapter. The path is Maths → Converters. See Tables 8-32 to 8-39.

Table 8-32.

| float_to_float3 | | Multiplexes a single input float to all components of a float3 |
|------------------------|-----------------------|---|
| In | param1:float | The input |
| Out | result_float3: float3 | A float3 (param1;param1;param1) |

Table 8-33.

| 3float_to_float3 | | Constructs a float3 given its constituents |
|-------------------------|-----------------------|---|
| In | floata:float | First float |
| | floatb:float | Second float |
| | floatc:float | Third float |
| Out | result_float3: float3 | The float3 output (floata; floatb; floatc) |

Table 8-34.

| 4float_to_float4 | | Constructs a float4 given its constituents |
|-------------------------|-----------------------|--|
| In | floata:float | First float |
| | floatb:float | Second float |
| | floatc:float | Third float |
| | floatd:float | Fourth float |
| Out | result_float4: float4 | The float4 output (floata; floatb; floatc; floatd) |

Table 8-35.

| float3_to_float | | Inflates a float3 value |
|------------------------|------------------|--------------------------------|
| In | float3_in:float3 | Input float3 |
| Out | a:float | First float |
| | b:float | Second float |
| | c:float | Third float |

Table 8-36.

| float4_to_4float | | Inflates a float4 value |
|-------------------------|------------------|--------------------------------|
| In | in_float4:float4 | Input float4 |
| Out | param1:float | First float |
| | param2:float | Second float |
| | param3:float | Third float |
| | param4:float | Fourth float |

Table 8-37.

| quaternion_to_4float | | Inflates a quaternion |
|-----------------------------|---------------------|------------------------------|
| In | in_quat: quaternion | Input quaternion |
| Out | param1:float | i value = x in ThMAD |
| | param2:float | j value = y in ThMAD |
| | param3:float | k value = z in ThMAD |
| | param4:float | Real part = w in ThMAD |

Table 8-38.

| 4float_to_quaternion | | Constructs a quaternion given its constituents |
|-----------------------------|-------------------------|---|
| In | param1:float | i value = x in ThMAD |
| | param2:float | j value = y in ThMAD |
| | param3:float | k value = z in ThMAD |
| | param4:float | Real part = w in ThMAD |
| Out | result_quat: quaternion | The output quaternion |

Table 8-39.

| matrix_to_quaternion | | Converts a suitable matrix to a corresponding quaternion |
|-----------------------------|-------------------------|---|
| In | source_matrix: matrix | The 4x4 matrix |
| Out | result_quat: quaternion | The output quaternion |

Dummies

Because of modules being able to provide more than one outgoing data flow per anchor, it might be helpful to multiplex values in such a way that one controller controls several components at the same time, providing the same value to all of them. That is what dummies are for. The path is Maths → Dummies. See Tables 8-40 to 8-43.

Table 8-40.

| float_dummy | | Multiplexes a single input float |
|--------------------|---------------------|---|
| In | float_in:float | The input |
| Out | result_float: float | The input copied to the output. Several other modules can connect to. |

Table 8-41.

| float3_dummy | | Multiplexes a single input float3 |
|---------------------|--------------------|---|
| In | float3_in:float3 | The input |
| Out | out_float3: float3 | The input copied to the output. Several other modules can connect to. |

Table 8-42.

| float4_dummy | | Multiplexes a single input float4 |
|---------------------|--------------------|--|
| In | float4_in:float4 | The input |
| Out | out_float4: float4 | The input copied to the output. Here several other modules can connect to. |

Table 8-43.

| quaternion_dummy | | Multiplexes a single input quaternion |
|-------------------------|----------------------|--|
| In | quat_in: quaternion | The input |
| Out | out_quat: quaternion | The input copied to the output. Here several other modules can connect to. |

Interpolation

Interpolation between values of different types. The path is Maths → Interpolation. See Tables 8-44 to 8-49.

Table 8-44.

| float_interpolate | | Linearly interpolates between two anchor values |
|--------------------------|---------------------|--|
| In | float_in_a:float | The first value |
| | float_in_b:float | The second value |
| | pos:float | The position, range is [0.0;1.0] |
| Out | result_float: float | The interpolation: $\text{float_in_a} + \text{pos} \cdot (\text{float_in_b} - \text{float_in_a})$ |

Table 8-45.

| float3_interpolate | | Linearly interpolates between two anchor values. |
|---------------------------|-----------------------|---|
| In | float3_in_a: float3 | The first value |
| | float3_in_b: float3 | The second value |
| | pos:float | The position, range is [0.0;1.0] |
| Out | result_float3: float3 | The interpolation: $\text{float3_in_a} + \text{pos} \cdot (\text{float3_in_b} - \text{float3_in_a})$ |

Table 8-46.

| float4_interpolate | | Linearly interpolates between two anchor values |
|---------------------------|-----------------------|---|
| In | float4_in_a: float4 | The first value |
| | float4_in_b: float4 | The second value |
| | pos:float | The position, range is [0.0;1.0] |
| Out | result_float4: float4 | The interpolation: float3_in_a + pos · (float3_in_b - float3_in_a) |

Table 8-47.

| float_smoother | | Gradually in time changes from an internally stored value to a given end value. Say the internal value reads 5.0, and the end value 1.0, then smoothly changes the internal value from 5.0 with decreasing steps to 1.0 |
|-----------------------|---------------------|--|
| In | value_in:float | The end value to change to |
| | speed:float | The speed |
| Out | result_float: float | The current developing internal value |

Table 8-48.

| quat_slerp_2p | | Linearly interpolates between two quaternions. You use this for a smooth rotation |
|----------------------|-------------------------|--|
| In | quat_a: quaternion | The first value |
| | quat_b: quaternion | The second value |
| | pos:float | The position, range is [0.0;1.0] |
| Out | result_quat: quaternion | The interpolation: quat_a + pos · (quat_b - quat_a) |

Table 8-49.

| quat_slerp_3p | | Linearly interpolates between three quaternions, either from first to second or from second to third, depending on whether the pos parameter is less than or greater than 0.5 |
|----------------------|-------------------------|---|
| In | quat_a: quaternion | The first value |
| | quat_b: quaternion | The second value |
| | quat_c: quaternion | The third value |
| | pos:float | The position, range is [0.0;1.0] |
| Out | result_quat: quaternion | The interpolation: $pos < 0.5: \text{quat_a} + 2 \cdot pos \cdot (\text{quat_b} - \text{quat_a})$ $pos \geq 0.5: \text{quat_b} + 2 \cdot (pos-0.5) \cdot (\text{quat_c} - \text{quat_b})$ |

Limiters

Limiters for float values. The path is Maths → Limiters. See Tables 8-50 and 8-51.

Table 8-50.

| float_limiter | | Limits either to a max or a min value |
|----------------------|---------------------|--|
| In | value_in:float | The input float |
| | limit_value: float | Where to set the limit |
| | type:enum | One of: max, min. Whether to set an upper or a lower limit. |
| Out | result_float: float | The limited value: $type = \text{max} \rightarrow \text{result} = \min(\text{value}, \text{limit})$ $type = \text{min} \rightarrow \text{result} = \max(\text{value}, \text{limit})$ |

Table 8-51.

| float_clamp | | Limits value to stay inside a range |
|--------------------|---------------------|---|
| In | value:float | The input float |
| | low:float | The lower limit |
| | high:float | The upper limit |
| Out | result_float: float | The clamped value: $low \leq value \leq max \rightarrow value$ $value < low \rightarrow low$ $value > high \rightarrow high$ |

Oscillators

Oscillators produce a periodically reoccurring float value, given some frequency f . More formally, $function(t + n \cdot 1/f) = function(t)$, $n = 0, +/- 1, +/- 2, \dots$

float_sequencer

The `float_sequencer` allows for repeatedly producing float numbers from a given sequence. The path is Maths \rightarrow Oscillators \rightarrow `float_sequencer`.

The sequence is defined from inside the Artiste GUI using a graphical `float_sequence` controller. The values are repeated unconditionally or triggered, depending on the chosen type; see Table 8-52.

Table 8-52. *Sequencer types*

| Type | Description |
|------------------|---|
| oscillating | Repeats unconditionally and endlessly. |
| trigger | Only starts and runs once when the trigger changes from: $trigger < 0 \rightarrow trigger > 0$ At the end of the sequence, stops with emitting the last value. Starts again when the same trigger $trigger < 0 \rightarrow trigger > 0$ got fired again. |
| trigger_pingpong | Same as trigger, but alternately runs the sequence in forward and reverse order when the trigger got fired again and again. |
| trigger_sync | Same as trigger_pingpong, but the forward sequence runs when triggering $trigger < 0 \rightarrow trigger > 0$ and the reverse sequence runs when $trigger > 0 \rightarrow trigger < 0$ |

The anchors are shown in Table 8-53.

Table 8-53.

| float_sequencer | A float sequencer |
|-----------------------------|---|
| In float_sequence: sequence | The float sequence. |
| length:float | The number of seconds one sequence run will take. |
| options:complex | |
| behavior: enum | The oscillator behavior, one of oscillating, trigger, trigger_pingpong, trigger_sync. |
| time_source: enum | Time (hidden input parameter) source. One of: <ul style="list-style-type: none"> - operating_system: Will use the operating system's timer - sequence: Will use the sequence timer, i.e., may disregard sequencing gaps |
| trigger:float | The trigger value. Unused for behavior = oscillating. |
| drive_type: enum | Time (hidden input parameter) mode. One of: <ul style="list-style-type: none"> - time_internal_relative: Pass the time that elapsed for rendering work since the engine started. - external: Use the drive parameter. |
| drive:float | Time parameter if external was used as drive_type. So you can use your own idea of time. |
| Out float:float | The oscillator value, one of the sequence |

inside_range

The path is Maths → Oscillators → inside_range See Table 8-54.

Table 8-54.

| inside_range | Tells whether or not a number lies inside a certain range |
|----------------------|---|
| In sound_in:float | The input float |
| range_low:float | Lower limit |
| range_high:float | Upper limit |
| randomness: float | Specifies the randomness for the random_beat anchor. 0 means never fire, 100 means always fire. |
| Out every_beat:float | Fires a one-frame pulse of 1.0 once the range is entered |
| random_beat: float | Same as every_beat, but fire only with a certain probability |
| in_range:float | 1.0 when we are inside the range. Otherwise 0.0. |

This is not an oscillator in the strictest sense, but an aggregator of input data. With input data showing periodic components, such as with music, it will look like an oscillator to components connected to its output, so it is in the oscillator section.

This pseudo-oscillator is useful for sound beat detection. It will even out unimportant side-beats to some extent. More in detail, it will act as follows: given an input of numbers $a_1, a_2, a_3, a_4, \dots$ and a range $[r_1, r_2]$, the module's `in_range` output anchor will be 1.0 for all numbers inside the range: $a_i \in [r_1, r_2]$, and 0.0 for all others. As an example, see Figure 8-2.

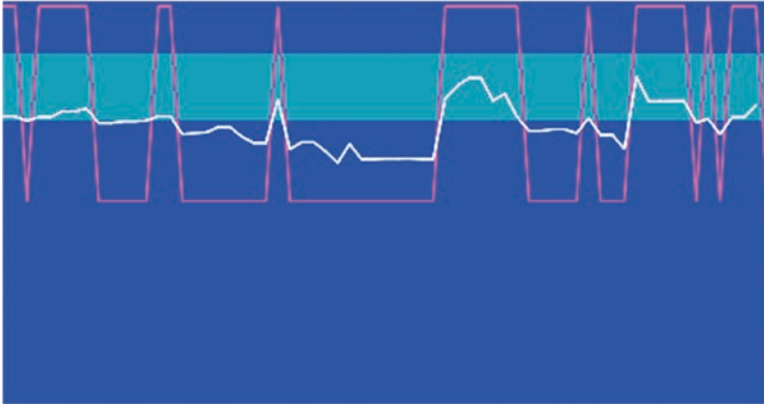


Figure 8-2. *In-range pseudo-oscillator, in-range output*

The white line denotes the input signal, green is the range, and the violet line is the `in_range` output.

In addition, the output anchor `every_beat` fires a single pulse every time the `in_range` output changes from 0.0 to 1.0. And as a variation to the latter, the output anchor `random_beat` will fire single pulses on a random basis when the `in_range` output changes from 0.0 to 1.0. The “randomness” input parameter describes the rate: only when some random number between 0 and 100 lies inside $[0; \text{randomness}]$, that pulse will be fired. So the probability of such random pulses will be “randomness” percent. An example for the `every_beat` output is shown in Figure 8-3.

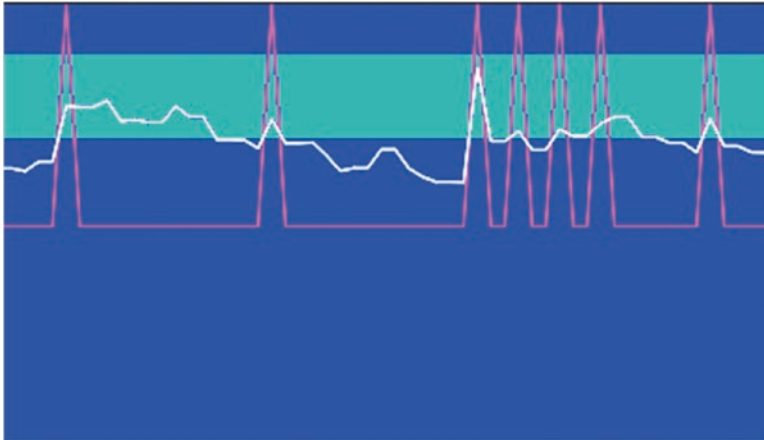


Figure 8-3. In-range pseudo-oscillator, beat output


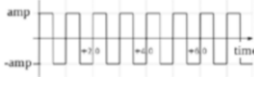
The white line denotes the input signal, green is the range, and the violet line is the `every_beat` output.

oscillator

This depends on the type: a noise, sine, saw, square, triangle, or quadratic oscillator. The path is Maths → Oscillators → Oscillator.



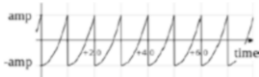
In detail, if `freq` is the frequency f , `amp` is the amplitude, and `ofs` is the offset; see Table 8-55.

Table 8-55. The oscillator module's types

| Name | Type | Description |
|-------|-------|---|
| noise | Noise | Will create a random value from inside $[-1;+1] \cdot \text{amp} + \text{ofs}$ each per frequency seconds |
| sin | Sine | For $f = 1$:  <code>ofs</code> will add to the values. NOTE: this is not mathematically $\sin(t)$, but $\sin(t \cdot 2\pi)$, so we have a full cycle each $t \rightarrow t + 1$ |
| saw | Saw | For $f = 1$:  <code>ofs</code> will add to the values |

(continued)

Table 8-55. (continued)

| Name | Type | Description |
|-----------|---------------------------------|--|
| square | Square | <p>For $f = 1$:</p>  <p>ofs will add to the values</p> |
| triangle | Triangle | <p>For $f = 1$:</p>  <p>ofs will add to the values</p> |
| quadratic | Half a parabolic curve repeated | <p>For $f = 1$:</p>  <p>ofs will add to the values</p> |

In addition, a parameter phase will add to the time, meaning shifting the curves horizontally. This is not used very often and left to its default value 0 you don't have to care about it if you don't need it. For all parameters, see Table 8-56.

Table 8-56.

| oscillator | An oscillator |
|------------|----------------|
| In | osc:complex |
| | osc_type: enum |
| | freq:float |
| | amp:float |
| | ofs:float |
| | phase.float |

(continued)





Table 8-56. (continued)

| oscillator | An oscillator |
|----------------------|---|
| options:complex | |
| time_source: enum | Time source (hidden input parameter). One of: <ul style="list-style-type: none"> – operating_system: Will use the operating system's timer – sequence: Will use the sequence timer, i.e. may subtract sequencing gaps |
| drive_type: enum | Time mode (hidden input parameter). One of: <ul style="list-style-type: none"> – time_internal_absolute: Pass the absolute time since the engine started – time_internal_relative: Pass the time which elapsed for rendering work since the engine started. – external: Use the drive parameter. |
| drive:float | Time parameter if external was used as drive_type. So you can have your own idea of time. |
| Out float:float | The oscillator value, always inside [-amp; +amp] + ofs |

pulse_oscillator

Repeatedly provides pulses of adjustable shape and length. Note that it only will start producing output when the trigger is activated. The path is Maths → Oscillators → pulse_oscillator. Possible shapes are shown in Table 8-57.

Table 8-57. Pulse types

| | |
|----------|---|
| Triangle |  |
| Square |  |
| Cosine |  |
| Gauss |  |

For the parameters, see Table 8-58.

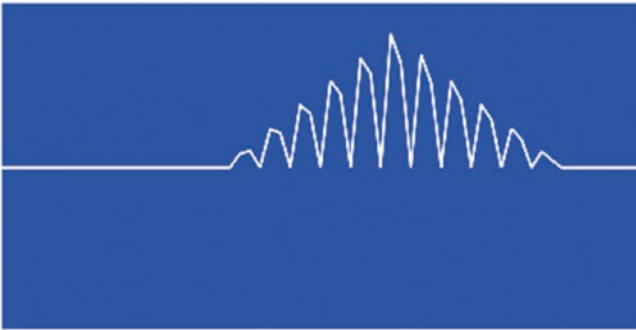
Table 8-58.

| | | |
|-------------------------|-------------------------------|---|
| pulse_oscillator | | |
| In | <code>in_signal:float</code> | Defaults to 1.0 You can use this to let the pulse envelope an input signal, see Figure 8-4. |
| | <code>trigger:float</code> | Defaults to 0.0 The pulse oscillator will run only when <code>trigger > 0</code> . |
| | <code>osc:complex</code> | |
| | <code>freq:float</code> | Frequency of the pulse, in 1/sec. |
| | <code>pulse_type: enum</code> | The pulse type. One of: <code>triangle</code> , <code>square</code> , <code>cosine</code> , or <code>gauss</code> . |

(continued)

Table 8-58. (continued)

| pulse_oscillator | |
|----------------------------------|--|
| <code>pulse_width</code> : float | The width of the pulse in seconds. For triangle, square and cosine shapes the width is obvious. For the gauss type, the width is -3σ up to 3σ (search for “Normal Distribution” in your favorite search engine). |
| <code>time_source</code> : enum | Time (hidden input parameter) source. One of: <ul style="list-style-type: none"> – <code>operating_system</code>: Will use the operating system’s timer – <code>sequence</code>: Will use the sequence timer, i.e., may subtract sequencing gaps |
| Out <code>result1</code> : float | The oscillator value, always inside <code>[0;in_signal]</code> |

**Figure 8-4.** Pulse oscillator of type “triangle” with enveloped input signal

Mesh

Meshes are collections of points that define surfaces, or faces, of objects in 3D.

Dummies

Dummies just forward their mesh input to the output. This is useful if you need a mesh to communicate to the interior of a macro and you have several consumers. The path is Mesh → Dummies → mesh_dummy. See Table 8-59.

Table 8-59.

| mesh_dummy | | Just pass its mesh input to the output |
|-------------------|---------------|---|
| In | mesh_in:mesh | Input mesh |
| Out | mesh_out:mesh | Output mesh |

Generators

Generators produce pixel data. The path is Mesh → Generators. See Tables 8-60 to 8-63.

Table 8-60.

| ocean | | An ocean in the x-z plane (y=0) |
|--------------|---------------------|---|
| In | time_speed: float | A time scaling for the surface motion |
| | lambda:float | Controls the wave speed |
| | wind_speed_x: float | Simulated wind speed x-coordinate |
| | wind_speed_y: float | Simulated wind speed y-coordinate |
| | normals_only: enum | Set to yes or no. If yes, use a slightly different algorithm where x-z-distortions are ignored. |
| Out | mesh:mesh | Output mesh |

Table 8-61.

| ocean_tunnel | | Category special effects. Uses the ocean simulation from module ocean, but maps calculated coordinates onto a tube |
|---------------------|-----------------|---|
| In | time_spec:float | Time scaling factor for the movement |
| Out | mesh:mesh | Output mesh |

Table 8-62.

| ribbon | | A twisted 3D ribbon, changes when time goes by |
|---------------|---------------------|--|
| In | spatial | |
| | start_point: float3 | Where the ribbon starts |
| | end_point: float3 | Where the ribbon ends |
| | up_vector: float3 | Up-vector for the skew |
| | shape | |
| | width:float | Ribbon width |
| | skew_amp: float | Skew amplitude |
| | time_amp: float | Shape change amplitude |
| | time_source: enum | One of sequence or real. If sequence, take sequenced time (needs to be started manually). If real, take real time since ThMAD start. |
| | segm_count: float | Number of segments to draw the ribbon |
| Out | mesh:mesh | Output mesh |

Table 8-63.

| xtra → kaleido_mesh | | A mesh like a hyperbolic bowl |
|----------------------------|--------------------|---|
| In | Hemispheric: float | The shape of the bowl. 0.0 is flat, 1.0 is like a half ball, more than 1.0 will squeeze further |
| Out | mesh_out: mesh | Output mesh |

Importers

Imports Cal3D or Blender files. The path is Mesh → Importers. See Tables 8-64 and 8-65.

Table 8-64.

| cal3d_importer | | Imports a cal3d project. An example is the state in the examples folder at examples/dolphin |
|-----------------------|---------------------------|--|
| In | filename:resource | Points to a Cal3D *.cfg file |
| | use_thread:enum | Set to yes or no. Whether to do the mesh generation in background |
| | transforms: complex | |
| | pre_rotation: quaternion | Describes a pre-rotation while loading. |
| | pre_rotation_center:float | The center of the pre-rotation |
| | rotation: quaternion | A rotation after import |
| | rotation_center: float3 | The center for the rotation |
| | post_rot_translate:float3 | Translation after import |
| | bones:complex | Allows for adapting various parameters of the Cal3D model |
| Out | mesh:mesh | Output mesh |
| | bones_bounding_box:mesh | The bones bounding box |
| | absolutes:complex | The output counterpart of the “bones” input anchor |

Table 8-65.

| obj_importer | | Imports a Blender *.obj file (Blender is an industrial grade 3D authoring program) |
|---------------------|-------------------------|--|
| In | filename:resource | The *.obj file to import. The examples/meshes resource folder contains some examples |
| | preserve_uv_coords:enum | Set to NO or YES, whether or not to also read the texture uv-coordinates from the file |
| | center_object: enum | Set to NO or YES, whether to center the object at the origin (uses center of mass algorithm) |
| Out | mesh:mesh | The output mesh |

Modifiers: Color

Mesh vertices color related modules. The path is Mesh → Modifiers → Color. See Table 8-66.

Table 8-66.

| | | |
|------------------------|--------------------|--|
| mesh_colorfield | | Sets colors of vertices according to a color field. Given a vector $A \rightarrow B$, all vertices get projected onto that vector, and the position there describes the position in the interpolated RGBA color gradient. If off limits, a modulus (repetition) will apply |
| In | mesh_in: mesh | The input mesh |
| | start_pos: float3 | A start position for the color field |
| | color_from: float4 | Color at the start position |
| | end_pos: float3 | An end position for the color field |
| | color_to: float4 | Color at the end position |
| Out | mesh_out: mesh | The output mesh |

Modifiers: Converters

Converts mesh data. The path is mesh → modifiers → converters → mesh_to_float3_arrays. See Table 8-67.

Table 8-67.

| | | |
|------------------------------|------------------------------|---|
| mesh_to_float3_arrays | | Converts a mesh to corresponding float3 arrays. Since no modules have a float3_array input, this module is currently not of much use |
| In | mesh_in: mesh | Input mesh |
| Out | vertices: float3_array | The vertices of the mesh |
| | vertex_normals: float3_array | The vertex normal |
| | face_normals: float3_array | The face normals |
| | face_centers: float3_arrays | The face centers |

Modifiers: Deformers

Modules for altering mesh coordinates. The path is Mesh → Modifiers → Deformers. See Tables 8-68 to 8-74.

Table 8-68.

| mesh_explode | | Explodes a mesh. Breaks mesh constituents boundaries and lets them blow apart |
|---------------------|-----------------------------|--|
| In | mesh_in:mesh | Input mesh |
| | start:float | An explosion trigger. Once set to a value > 0, explosion starts. The value of this parameter controls the exploding particle speed. The higher, the faster. Once set to < 0, start all over with the unexploded mesh |
| | explosion_factor:float | A factor multiplied to the explosion speed (you can set start to 1.0 and let only this parameter control the speed) |
| | velocity_deceleration:float | Controls the amount the exploding particles get slower after exploding. |
| | use_weights: enum | Set to no or yes. If yes, each particle's weight after the explosion is calculated from its spatial extent, and the weight will influence the speed the particles blow apart. |
| | weight_power:float | Controls the influence of the weight if enabled. If higher than one, small particle's weight differences do not count that much. |
| Out | mesh_out:mesh | The output mesh. |

Table 8-69.

| | |
|---------------------|--|
| mesh_inflate | This module is currently under construction and not functional |
|---------------------|--|

Table 8-70.

| mesh_noise | | Adds noise to vertex coordinates. Normals, texture coordinates and face assignments stay untouched |
|-------------------|----------------------|---|
| In | mesh_in:mesh | The input mesh |
| | noise:amount: float3 | The noise amount in each direction |
| Out | mesh_out: mesh | The output mesh |

Table 8-71.

| mesh_normal_randistort | | Randomly distorts vertex coordinates and/or vertex normals. If just distorting normals, you can add interesting light effects, since the normals control the light reflection |
|-------------------------------|--------------------------------|--|
| In | mesh_in_mesh | Input mesh |
| | distortion_factor:float3 | The random distortion factor in x, y and z dimension |
| | distort_normals:enum | Set to no or yes, whether to distort the normals (heavily affecting light reflectivity) |
| | distort_vertices:enum | Set to no or yes, whether to distort vertex coordinates. |
| | vertex_distortion_factor:float | If both normals and vertices are distorted, use this to control the relative amount of vertex distortion |
| Out | mesh_out: mesh | The output mesh |

Table 8-72.

| mesh_rain_down | | Simulates a raining down or falling down of particles |
|-----------------------|--------------------------|---|
| In | mesh_in:mesh | The input mesh. |
| | start:float | Controls both the onset of the effect (start > 0) and the magnitude of the effect (explosion and raining speed) . |
| | floor_level:float | Where raining stops on the y-axis. |
| | explosion_factor:float | Explosion of the particles (decoupled parts of the input mesh) on the x-z plane. Default is 1.0 (no explosion), and higher values increase the explosion magnitude. |
| | landing_fluffiness:float | Landing, that is the final stage of the movement along the y-axis, can be tuned according to this input parameter. Default is 0.0, which means off, and higher values lower the floor_level parameter on a per-particle basis, weighted by the size of each particle. |
| Out | mesh_out: mesh | The output mesh |

Table 8-73.

| | |
|------------------|--|
| mesh_vertex_move | This module is currently under construction and not functional |
|------------------|--|

Table 8-74.

| | |
|-------------|--|
| mesh_vortex | This module is currently under construction and not functional |
|-------------|--|

As for the mesh_rain_down module, given an input mesh, this deformer will decouple all the parts of it and start exploding and moving them down the y-axis, all the way down to some floor level. It thus mimics rain when coming out of a cloud.

If you need a movement in a different direction, you can use the rotation module at Mesh → Modifiers → Transforms → mesh_rotate_quat before input, and a rotation back using another mesh_rotate_quat after output. The module mesh_rotate_quat uses a quaternion to specify the rotation. If you prefer the axis-angle-rotation notation, you can use the converter module Maths → Arithmetic → Quaternion → axis_angle_to_quaternion. For an example, see Figure 8-5.

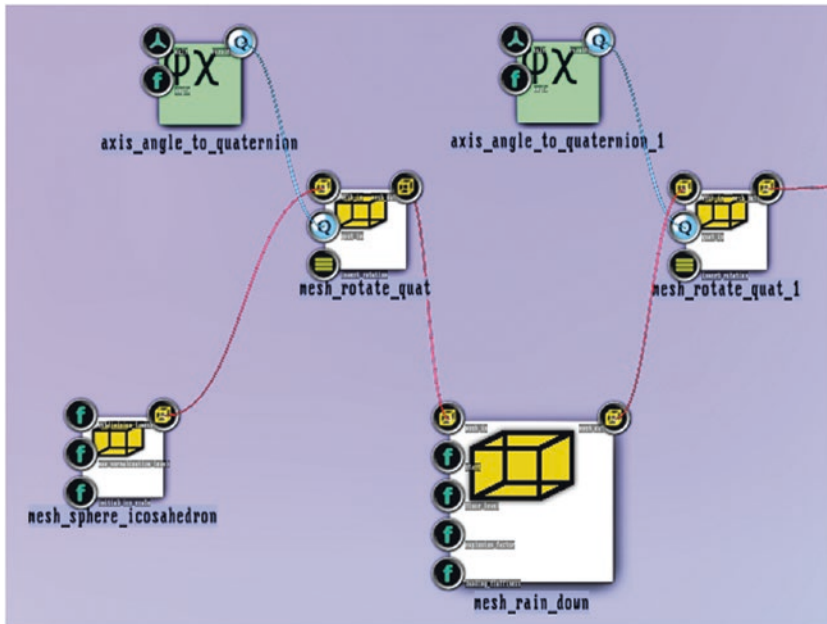


Figure 8-5. The mesh_rain_down module with rotation transformation

Modifiers: Helpers

Helper modules for meshes. The path is Mesh → Modifiers → Helpers. See Tables 8-75 and 8-76.

Table 8-75.

| | |
|-----------------------|--|
| mesh_compute_tangents | This module is currently under construction and not functional |
|-----------------------|--|

Table 8-76.

| | | |
|---|---------------|---|
| mesh_compute_tangents_vertex_color_array | | Computes tangents of vertices and stores them in a color array. Can be used in shaders |
| In | mesh_in:mesh | Input mesh |
| Out | mesh_out:mesh | Output mesh |

Modifiers: Pickers

Used to extract values from a mesh. The path is Mesh → Modifiers → Pickers. See Tables 8-77 and 8-78.

Table 8-77.

| | | |
|---------------------------|----------------------|---|
| mesh_attach_picker | | Given two vertex IDs, returns the coordinates of ID1 and a rotation quaternion describing the rotation necessary to look along the normal of ID1 and with ID2-ID1 horizontal |
| In | mesh_in:mesh | The input mesh |
| | id_a:float | The integer part points to vertex A (0, 1, 2, ...) |
| | id_b:float | The integer part points to vertex B (0, 1, 2, ...) |
| Out | position:float3 | The coordinates of vertex A |
| | rotation: quaternion | The rotation vector defined by the normal at A, and the vector from A to B |

Table 8-78.

| mesh_vertex_picker | | Obtains complete data for a certain mesh vertex |
|---------------------------|---|---|
| In | mesh_in:mesh id:float | Input mesh The integer part denotes the mesh vertex ID (0, 1, 2, ...) |
| Out | vertex:float3 normal:float3 color:float4 texcoords:float3 passthru:mesh | The vertex coordinates The normal vector at that vertex The color at that vertex The texture coordinates at that vertex Output mesh. Note that you have to connect this to a result bearing rendering sub-pipeline for the module to work. Just put it between the mesh producer and the intended mesh consumer (like renderer) |
| | sizes:complex | Mesh array sizes |
| | vertices_size: float | Number of vertices |
| | normals_size: float | Number of vertex normals |
| | colors_size: float | Number of vertex colors |
| | faces_size: float | Number of faces |

Both the `mesh_attach_picker` and the `mesh_vertex_picker` modules can be used to let a visualization scan through the vertices of a mesh and do things based on the coordinates it gets. Note that using `mesh_attach_picker` you do not have to specify the second vertex ID if you are not going to use the rotation quaternion.

Modifiers: Transforms

Used to modify vertex data. The path is Mesh → Modifiers → Transforms. See Tables 8-79 to 8-85.

Table 8-79.

| mesh_mirror | | Mirrors a mesh |
|--------------------|---------------------------|--|
| In | mesh_in:mesh axis:enum | The input mesh Which axis to mirror, one of x, y, or z. |
| Out | mesh_out: mesh | The output mesh |

Table 8-80.

| mesh_norm_scale | | Scales and moves a mesh such that all its vertices lie inside [0;1]. Size relations are maintained (the biggest span will be [0;1]) |
|------------------------|----------------|--|
| In | mesh_in:mesh | The input mesh |
| Out | mesh_out: mesh | The output mesh |

Table 8-81.

| mesh_rotate_quat | | Rotates a mesh by a rotation quaternion |
|-------------------------|-----------------------|---|
| In | mesh_in:mesh | The input mesh |
| | quat_in: quaternion | The rotation quaternion (if you need to control this with an axis-angle rotation, use module Maths → Arithmetic → Functions → axis_angle_to_quaternion) |
| | invert_rotation: enum | Set to no or yes, whether to invert the rotation |
| Out | mesh_out:out | The output mesh |

Table 8-82.

| mesh_rotate_quat_around_vertex | | Rotates a mesh around one of its vertices. Additionally translate afterwards |
|---------------------------------------|----------------------|--|
| In | mesh_in:mesh | The input mesh |
| | quat_in: quaternion | The quaternion describing the rotation |
| | vertex_rot_id: float | The integer part points to the vertex around which the rotation will be performed. Values are 0, 1, 2, ... |
| | offset_pos: float3 | An additional translation after the rotation |
| Out | out_mesh: mesh | The output mesh |

Table 8-83.

| mesh_scale | | Performs a scaling |
|-------------------|----------------|-------------------------------------|
| In | mesh_in:mesh | The input mesh |
| | scale:float3 | A scaling factor for each dimension |
| Out | mesh_out: mesh | The output mesh |

Table 8-84.

| mesh_translate | | Translates the mesh |
|-----------------------|---------------------|----------------------------|
| In | mesh_in:mesh | The input mesh |
| | translation: float3 | The translation vector |
| Out | out_mesh: mesh | The output mesh |

Table 8-85.

| mesh_translate_ edge_wraparound | | Does a translation of vertices, but wraps around at some limiting coordinate values |
|--|---------------------|---|
| In | mesh_in:mesh | The input mesh |
| | translation: float3 | The translation |
| | edge_min: float3 | Defines the x, y, and z values at which a coordinate-wise wrap-around occurs if edge coordinates will be less than |
| | edge_max: float3 | Defines the x, y, and z values at which a coordinate-wise wrap-around occurs if edge coordinates will be greater than |
| Out | mesh_out: mesh | The output mesh |

Particles

Particle system related simple meshes in two dimensions. The path is Mesh → Particles. See Tables 8-86 to 8-88.

Table 8-86.

| mesh_disc | | A disc or a ring. If used with the basic mesh renderer, be aware that the texture is spread over the complete perimeter. You could try to use the use_tex_center parameter if you do not want that |
|------------------|----------------------|---|
| In | num_segments: float | The number of segments to generate along the perimeter |
| | diameter:float | The hole's diameter. Make this zero if you want to have a disc instead of a ring |
| | border_width: float | The ring's width. If the diameter is zero, the disc's radius. |
| | use_tex_center: enum | Set to no or yes. If yes, do not spread the texture over the whole perimeter, instead use only the center part, which makes this texture used effectively one-dimensional. |
| Out | mesh:mesh | The output mesh |

Table 8-87.

| mesh_rays | | Random fan triangles starting at (0,0,0) and up to some size limit |
|------------------|--------------------------|--|
| In | num_rays:float | The number of rays |
| | center_color: float4 | The center color to assign |
| | options / limit_ray_size | If < 0, make the maximum ray size 1.0. Otherwise use the limit given here. |
| Out | mesh:mesh | The output mesh |

Table 8-88.

| | |
|-----------|--|
| mesh_star | This module is currently under construction and not functional |
|-----------|--|

Segmesh

Segmesh modules under mesh → segmesh are currently under construction and not functional.

Solid

Generate solid 3D bodies using meshes. The path is Mesh → Solid. See Tables 8-89 to 8-102.

Table 8-89.

| mesh_arrow | | An arrow |
|-------------------|-----------------|---|
| In | start:float3 | Start position |
| | end:float3 | End position |
| | radius:float | The radius of the body |
| | segments:float | The number of segments |
| | head_size:float | The ratio the head takes of the complete length |
| Out | mesh:mesh | The output mesh |

Table 8-90.

| mesh_box | | A box made of six quadratic faces. Linear dimensions are always [-0.5;0.5] |
|-----------------|-----------|---|
| Out | mesh:mesh | The output mesh |

Table 8-91.

| mesh_cone | | A cone |
|------------------|----------------|-------------------------------|
| In | base:float3 | The center of the cone's base |
| | peak:float3 | The cone's peak |
| | radius:float | The radius at the base |
| | segments:float | The number of segments |
| Out | mesh:mesh | The output mesh |

Table 8-92.

| mesh_cylinder | | A cylinder |
|----------------------|----------------|------------------------|
| In | start:float3 | The center of the base |
| | end:float3 | The center of the top |
| | radius:float | The radius |
| | segments:float | The number of segments |
| Out | mesh:mesh | The output mesh |

Table 8-93.

| mesh_grid | | A quadratic grid made of small quads. The center will be at (0;0;0) and the size of the grid will be 1x1 |
|------------------|-------------------------|--|
| In | power_of_two_size:float | The power of two of this is the number of quads along each side. So if this is 4, we have $2^4 = 16$ each side, and $16 \times 16 = 256$ in total. |
| | plane:enum | One of xy, xz and yz. The orientation of the grid. |
| Out | mesh:mesh | The output mesh |

Table 8-94.

| mesh_planes | | A number of parallel planes |
|--------------------|----------------------|---|
| In | num_planes:float | Number of planes |
| | space_between: float | Distance between adjacent planes |
| | diameter:float | Dimension of a plane will be 2 x diameter times 2 x diameter |
| | normals:float3 | Specifies the orientation in space of each plane. "normals" is perpendicular to a plane |
| | colors:complex | |
| | color_a:float4 | Each planes color at vertex 1 |
| | color_b:float4 | Each planes color at vertex 2 |
| | color_c:float4 | Each planes color at vertex 3 |
| | color_d:float4 | Each planes color at vertex 4 |
| Out | mesh:mesh | The output mesh |

Table 8-95.

| mesh_solid_supershape | | A complex shape by combining power and sine functions |
|------------------------------|----------------------|--|
| In | x:complex | X and y dimension for the algorithm currently all handled here |
| | x_num_segments:float | Number of segments to generate. The total number of vertices generated is the square of this |
| | x_start:float | An angle for the algorithm |
| | x_stop:float | An angle for the algorithm |
| | y_start:float | An angle for the algorithm |
| | y_stop:float | An angle for the algorithm |
| | x_a:float | A scaling factor |
| | x_b:float | A scaling factor |
| | x_n1:float | A power argument |
| | x_n2:float | A power argument |
| | x_n3:float | A power argument |
| | x_m:float | A factor for an angle inside the algorithm |
| Out | mesh:mesh | The output mesh |

Table 8-96.

| | | |
|--------------------|--|-------------------------------|
| mesh_sphere | A sphere. Uses the very basic straightforward subdividing algorithm which is used for the earth globe | |
| In | num_sectors:float | The number of longitude parts |
| | num_stacks:float | The number of latitude parts |
| Out | mesh:mesh | The output mesh |

Table 8-97.

| | | |
|--------------------------------|--|---|
| mesh_sphere_icosahedron | A sphere with a more regular subdivision. Start from an icosahedron (20 faces of type triangle), and subdivide each of its parts (making four triangles out of one) iteratively, projecting each of the new points onto the sphere (just stretch it to the desired cube's radius) | |
| In | subdivision_level:float | Number of iterations for the subdivision. Be careful not to make this too big, otherwise ThMAD will break. The default is 6 and this will do in many cases. Values greater than 7 will be ignored and get clamped to 7. |
| | max_normalization_level: float | Normals recalculation can be configured to stop at a certain level, which makes the surface bumpier if light shines on it. |
| | initial_ico_scale:float | Scale the original icosahedron before starting the subdivision. Those points will not be projected onto the sphere. Currently a little buggy since it shows artifacts. |
| Out | mesh:mesh | The output mesh |

Table 8-98.

| | | |
|--------------------------------------|---|--|
| <code>mesh_sphere_ octahedron</code> | | A sphere with a more regular subdivision. Start from an octahedron (8 faces of type triangle), and subdivide each of its parts (making four triangles out of one) iteratively, projecting each of the new points onto the sphere (just stretch it to the desired cube's radius) |
| In | <code>subdivision_level: float</code> | Number of iterations for the subdivision. Be careful not to make this too big, otherwise ThMAD will break. Values greater than 9 will be ignored and get clamped to 9. |
| | <code>max_normalization_level: float</code> | Normals recalculation can be configured to stop at a certain level, which makes the surface a little more bumpy if light shines on it. |
| Out | <code>mesh:mesh</code> | Output mesh |

Table 8-99.

| | | |
|--------------------------------|--|--|
| <code>mesh_super_banana</code> | | A flexible, possibly banana-shaped mesh. Note that you have to set the shapes first before you can see anything |
| In | <code>num_sectors:float</code> | Number of sectors perimeter-wise |
| | <code>num_stacks:float</code> | Number of stacks along its length dimension |
| | <code>shape:complex</code> | |
| | <code>x_shape: sequence</code> | The sequence along one radial dimension |
| | <code>x_shape_multiplier:float</code> | A multiplier for the x-shape |
| | <code>y_shape: sequence</code> | The sequence along the other radial dimension |
| | <code>y_shape_multiplier:float</code> | A multiplier for the y-shape |
| | <code>z_shape: sequence</code> | The sequence along the length dimension |
| | <code>z_shape_multiplier:float</code> | A multiplier for the z-shape |
| | <code>size:complex</code> | |
| | <code>size_shape_x: sequence</code> | The size shape of one radius along the length dimension |
| | <code>size_shape_x_multiplier:float</code> | A multiplier for the size shape of one radius along the length dimension |
| | <code>size_shape_y: sequence</code> | The size shape of the other radius along the length dimension |
| | <code>size_shape_y_multiplier:float</code> | A multiplier for the size shape of the other radius along the length dimension |
| Out | <code>mesh:mesh</code> | The output mesh |

Table 8-100.

| mesh_torus_knot | | A torus knot. This is a knot which lies on the surface of a torus, with possibly many windings |
|------------------------|-------------------------------|---|
| In | num_sectors:float | Number of sectors perimeter-wise |
| | num_stacks:float | Number of segments along its length |
| | p:float | P parameterization |
| | q:float | Q parameterization |
| | phi_offset:float | Angular offset |
| | size | |
| | size_shape_x: sequence | The size of the first radial component along the length |
| | size_shape_x_multiplier:float | A factor applied to the size_shape_x sequence |
| | size_shape_y: sequence | The size of the second radial component along the length |
| | size_shape_y_multiplier:float | A factor applied to the size_shape_y sequence |
| Out | mesh:mesh | The output mesh |

Table 8-101.

| metaballs | | Shows metaballs. This is a physics simulation of an equipotential surface. Functional, but currently buggy, shows some artifacts |
|------------------|-----------------|---|
| In | grid_size:float | The resolution |
| Out | mesh:mesh | The output mesh |

Table 8-102.

| plane_uv_distort | | An x-y plane with dimensions [-1;-1] → [1;1] at z=0 with a texture coordinates distortion by sequence controllers |
|-------------------------|--------------------------|---|
| In | x_res:float | The integer part of this figure is the resolution in x-direction |
| | y_res:float | The integer part of this figure is the resolution in y-direction |
| | distortion: complex | |
| | x_shape: sequence | Describes the distortion along the x axis. If you set all values to 0.5 inside the sequence, no distortion will happen. Numbers below 0.5 mean a negative distortion, above 0.5 a positive distortion |
| | x_shape_multiplier:float | A multiplier applied to the x distortion values |
| | z_shape: sequence | Describes the distortion along the y axis (don't be confused by the name; it is wrong but left like that for backward compatibility) |
| | z_shape_multiplier:float | A multiplier applied to the y distortion values |
| Out | mesh:mesh | The output mesh |

Texture

These modules are for texture coordinates manipulation of meshes. The path is Mesh → Texture. See Tables 8-103 and 8-104.

Table 8-103.

| mesh_tex_bitmap_distort | | Uses a bitmap to describe a texture coordinate distortion |
|--------------------------------|-----------------|--|
| In | mesh_in:mesh | The input mesh |
| | bitmap:bitmap | The bitmap. Only RED and GREEN are used. Red describes the U-distortion of the texture, Green the V-distortion. A value of 0.5 means no distortion |
| | intensity:float | Controls the overall intensity of the effect |
| | spatial:complex | Additional distortion parameters |
| | u_off:float | Texture u coordinate offset |
| | v_off:float | Texture v coordinate offset |
| | u_scale:float | Texture u coordinate scaling factor |
| | v_scale:float | Texture v coordinate scaling factor |
| Out | mesh_out: mesh | The output mesh |

Table 8-104.

| mesh_tex_sequ_distort | | Uses sequences to describe a texture coordinate distortion |
|------------------------------|---------------------|---|
| In | mesh_in:mesh | The input mesh |
| | spatial:complex | |
| | scale_u:float | An additional texture u coordinate scaling factor |
| | scale_v:float | An additional texture v coordinate scaling factor |
| | translate_u: float | An additional texture u coordinate translation |
| | translate_v: float | An additional texture v coordinate translation |
| | distortion: complex | |
| | u_shape: sequence | Describes the texture u coordinate distortion |
| | v_shape: sequence | Describes the texture v coordinate distortion |
| Out | mesh_out: mesh | The output mesh |

Vertices

Define meshes by algorithms. The path is Mesh → Vertices. See Tables [8-105](#) to [8-109](#).

Table 8-105.

| bspline_vertices | | Takes the input mesh's vertices and uses them to feed a B-spline pipeline. Note that the output mesh does not contain any normals or faces, so the mesh_basic_render module cannot handle it |
|-------------------------|---------------|---|
| In | source:mesh | The input mesh |
| | density:float | The density: the number of points generated for each vertex pair |
| Out | mesh:mesh | The output mesh |

Table 8-106.

| lightning_vertices | | Plasma ball effect |
|---------------------------|-------------------|--|
| In | rand_seed:float | Some random seed. Change this value for different shapes. Usually you want to send this to a mesh_dot_render module for rendering, since no normals or faces are generated by this module. |
| | lifetime:float | Increase this value if you want the simulation to be less nervous |
| | length:float | The length of the lightning strikes |
| | mesh_a:mesh | An input mesh. Vertices from here are used for the lightning start points. Normals here are used for the primary direction of the strikes |
| | num_points: float | The number of lightning vertices |
| | scaling:float3 | Use this to control the dimensions of the effect |
| | sub_divide:float | Generate that many points for each part of a lightning strike. If you use the dot renderer, increasing this value makes sense |
| Out | mesh:mesh | The output mesh |

Table 8-107.

| | | |
|--|---------------------------|---|
| modifiers → mesh_vertex_distance_sort | | From the input mesh, fetches the vertices and sorts them according to the distance to a given point (farthest away first). The output contains no color or faces; no texture coordinates! So the mesh_basic_render module cannot handle it |
| In | mesh_in:mesh | The input mesh. Will not be changed. |
| | distance_to: float3 | The point from which to measure the distance |
| Out | mesh_out: mesh | The output mesh. No colors or normals or faces; only vertices. |
| | original_ids: float_array | The IDs of the input mesh after the sort. |

Table 8-108.

| random_vertices | | Generates random vertices |
|------------------------|-------------------|--|
| In | rand_seed:float | Random seed. Change if you need different distributions |
| | num_points: float | Number of points |
| | scaling:float3 | A scaling |
| | distrib:enum | Distribution: box: Uniformly inside a box sphere: Uniformly on the surface of a sphere bowl: Uniformly inside a sphere gauss: Gauss in 3D gauss2d: Gauss in 2D, z = 0 |
| Out | mesh:mesh | The output mesh |

Table 8-109.

| ribbon_vertices | | A twisted ribbon. Changes as time goes by. Similar to mesh → generators → ribbon, but won't create faces, so the mesh_basic_render module cannot handle it |
|------------------------|-----------------------|---|
| In | start_point:float3 | The starting point |
| | end_point:float3 | The ending point |
| | up_vector:float3 | The up-vector, defines how it moves |
| | num_segments: float | The number of segments |
| | particle_scale: float | Influences the coloring. Coloring is from black to white, or gray if you change this to a value less than 1.0 |
| | width:float | Controls the width of the ribbon |
| | skew_amp:float | The skew amplitude |
| | time_amp:float | The time scale for the twisting |
| Out | mesh:mesh | The output mesh |

Xtra

Some special mesh generators and modifiers. The path is Mesh → Xtra.
See Tables 8-110 to 8-112.

Table 8-110.

| cloud_plane | | A colored cloud plane |
|--------------------|---------------|------------------------------|
| Out | mesh_out:mesh | The output mesh |

Table 8-111.

| planeworld | | Used in conjunction with the mesh → vertices → random_vertices as input, creates a complex knot with full mesh properties: vertices, colors, texture coordinates, faces |
|-------------------|--------------------------------|--|
| In | bspline_vertices_ mesh:mesh | The input mesh. You do not have to use the output from the bspline_vertices module here! The b-splines are calculated internally. |
| Out | mesh_result:mesh | The output mesh |

Table 8-112.

| thorn | | Generates a colored thorn |
|--------------|------------------|----------------------------------|
| Out | mesh_result:mesh | The output mesh |

Particlesystems

Particlesystems deal with many objects of small size each. They somehow compensate for the missing of a loop construct inside states.

Fractals

Fractals are geometric objects or object collections with some special, fractal related, mathematical properties. The one module here uses an iterated function system algorithm to produce fractals. The path is Particlesystems → Fractals. See Table 8-113.

Table 8-113.

| ifs_modifier | Continuously change particles' coordinates according to an iterated function system (IFS). This is $x \rightarrow f(x) \rightarrow f(f(x)) \rightarrow f(f(f(x)))$ and so on. The function f here is a random mixture of two affine mappings |
|--------------------------------------|---|
| In | |
| in_particlesystem: particlesystem | The input particle system |
| change_probab: float | Set this to a value less than 1.0 if you want to change only a certain part of the particles each frame (1/60 sec.) |
| change_random: enum | This is a trigger. Choose go and a random IFS parameter set will be generated and used. Note that only a fraction of the randomly generated sets will yield usable results |
| save_params: enum | A trigger saving the current parameter set to a file inside /home/[USER]/thmad/ [VERSION]/data/resources/ifs So it can be used in future states. |
| load_params: resource | Load a previously saved parameter set. The value of this anchor is not persisted, so it is a one-time setting action. The <i>parameters</i> gained here will be persisted, of course. |
| ifs:complex | Current parameter set: two affine transformations. The algorithm switches randomly between them, with equal probabilities |
| sys1:complex | First system. |
| sys2:complex | $\begin{pmatrix} a11 & a12 & a13 & at1 \\ a21 & a22 & a23 & at2 \\ a31 & a32 & a33 & at3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Second system |
| Out | |
| particlesystem: particlesystem | $\begin{pmatrix} b11 & b12 & b13 & bt1 \\ b21 & b22 & b23 & bt2 \\ b31 & b32 & b33 & bt3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ The output particle system |

Generators

Generators produce particles. The path is Particlesystems → Generators.
See Tables 8-114 to 8-116.

Table 8-114.

| | |
|------------------------------------|--|
| basic_spray_emitter | Creates particles at a given point in space. The number of particles will not change later – particles that are considered dead still remain in memory and possibly later start to live again at this point |
| In | |
| num_particles: float | The total number of particles |
| particles_per_second: float | If not set to -1, this number specifies the maximum number of particles per second that will be resurrected when dead. |
| spatial: complex | |
| emitter_position: float3 | The position in space where particles start to live. |
| speed: complex | This specifies the speed the particles start their movement when emitted |
| speed_x: float | |
| speed_y: float | |
| speed_z: float | |
| speed_type: enum | The way the speed parameter is applied. If random_balanced, the speed-x for example will be chosen randomly from [-speed_x/2; speed_x/2]. If directional, the speed will be taken as specified |
| size: complex | The size calculation for each particle: |
| particle_size_base: float | size = size_base + |
| particle_size_random_weight: float | rnd[-0.5; 0.5] · random_weight |
| | Where rnd[a; b] specifies a random number between a and b. |
| time_source: enum | Where to take the time from. One of real or sequencer |
| particle_rotation_dir: quaternion | If particle rotation is enabled, use this to specify the rotation. |
| enable_rotation: enum | Whether or not to enable rotation: one of true or false |

(continued)

Table 8-114. (continued)

| | |
|---|--|
| basic_spray_emitter | Creates particles at a given point in space. The number of particles will not change later – particles that are considered dead still remain in memory and possibly later start to live again at this point |
| <pre> appearance: complex color:float4 time:complex particle_lifetime_base: float particle_lifetime_random_weight: float </pre> | <p>The color</p> <p>Each particle’s lifetime: $lifetime = lifetime_base + rnd[-0.5;0.5] \cdot random_weight$ Where $rnd[a;b]$ specifies a random number between a and b.</p> |
| Out particlesystem: particlesystem | The output particle system |

Table 8-115.

| | |
|--|---|
| bitmap_to_particlesystem | A particlesystem using the pixels of a bitmap for movement origin and color |
| In | |
| <pre> bitmap_in:bitmap particles_per_second:float spatial:complex bitmap_size:float bitmap_normal: float3 bitmap_upvector:float3 bitmap_position: float3 speed:complex speed_x:float speed_y:float speed_z:float </pre> | <p>Input bitmap</p> <p>If not set to -1, this number specifies the maximum number of particles per second that will be resurrected when dead.</p> <p>Size of the bitmap, from a coordinate origin point of view</p> <p>The normal, specifying the bitmap’s orientation in space</p> <p>Rotation of the bitmap around the normal vector</p> <p>Position of the bitmap’s center</p> <p>This specifies the speed the particles start their movement when emitted</p> |

(continued)

Table 8-115. (continued)

| bitmap_to_particlesystem | A particlesystem using the pixels of a bitmap for movement origin and color |
|--|--|
| speed_type: enum | The way the speed parameter is applied. If random_balanced, the speed-x for example will be chosen randomly from [- speed_x/2;speed_x/2]. If directional, the speed will be taken as specified |
| size:complex | The size calculation for each particle: size = size_base + rnd[-0.5;0.5].random_weight |
| particle_size_base:float | |
| particle_size_random_weight:float | Where rnd[a;b] specifies a random number between a and b. |
| time_source: enum | Where to take the time from. One of real or sequencer. For simple states, use real. |
| particle_rotation_dir: quaternion | If particle rotation is enabled, use this to specify the rotation. |
| enable_rotation | Whether or not to enable rotation: one of true or false |
| appearance: complex | |
| color:float4 | The color |
| time:complex | Each particle's lifetime: lifetime = lifetime_base + rnd[-0.5;0.5].random_weight |
| particle_lifetime_base: float | Where rnd[a;b] specifies a random number between a and b. |
| particle_lifetime_random_weight: float | |
| Out particlesystem: | The output particle system |
| particlesystem | |

Table 8-116.

| <code>particles_mesh_spray</code> | Similar to <code>basic_spray_emitter</code> , but uses a mesh for the particles' movement origin |
|---|---|
| In | |
| <code>num_particles:float</code> | The total number of particles |
| <code>particles_per_second:float</code> | If not set to -1, this number specifies the maximum number of particles per second that will be resurrected when dead. |
| <code>mesh_properties: complex</code> | Mesh properties apply only while a particle is born or resurrected |
| <code>pick_type:enum</code> | One of <code>sequential</code> or <code>random</code> . How to choose vertices from the mesh as particle movement origins. |
| <code>center:float3</code> | A translation vector common to all particles |
| <code>spread:float3</code> | A multiplier applied to the input vertex position coordinates while adding to the particles' final positions |
| <code>random_deviation:float3</code> | Specifies the amount of randomness when calculating the particle's position |
| <code>spatial:complex</code> | |
| <code>speed_type: enum</code> | How to calculate the speed: one of: <ul style="list-style-type: none"> - <code>random_balanced</code>: for each coordinate, speed will be from <code>inside_rnd[-0.5;0.5] · speed_* · speed_mult</code> - <code>directional</code>: take <code>speed_mult · speed_*</code> - <code>mesh_beam</code>: take <code>speed_mult</code> <code>origin_to_vertex + add_vector</code> Where <code>rnd[a;b]</code> specifies a random number between a and b, and <code>origin_to_vertex</code> is the normalized vector from the origin to the vertex point. |
| <code>speed_multiplier: float</code> | Speed multiplier (<code>speed_mult</code>) |
| <code>speed_random_value:float</code> | A multiplier for a random value <code>[-0.5;0.5]</code> added to <code>speed_mult</code> |

(continued)

Table 8-116. (continued)

| particles_mesh_spray | Similar to basic_spray_emitter, but uses a mesh for the particles' movement origin |
|---------------------------------------|---|
| speed:complex | This specifies the speed the particles start their movement when emitted. The way it is interpreted is specified by speed_type, speed_multiplier and speed_random_value |
| speed_x:float | |
| speed_y:float | |
| speed_z:float | |
| add_vector: float3 | The add_vector only applies to speed_type = mesh_beam |
| size:complex | The size calculation for each particle: |
| particle_size_base:float | size = size_base + rnd[-0.5;0.5]·random_weight |
| particle_size_random_weight:float | Where rnd[a;b] specifies a random number between a and b. |
| appearance: complex | |
| color:float4 | Particles color |
| time:complex | Each particle's lifetime: |
| particle_lifetime_base: float | lifetime = lifetime_base + rnd[-0.5;0.5]·random_weight |
| particle_lifetime_random_weight:float | Where rnd[a;b] specifies a random number between a and b. |
| time_source: enum | Where to take the time from. One of real or sequencer. For simple states, use real |
| Out particlesystem: particlesystem | The output particle system |

Modifier

The modifiers tell how particles behave after they were emitted, apart from the speed that was assigned to them at the emitter point. The path is Particlesystems → Modifiers. See Tables 8-117 to 8-122.

Table 8-117.

| basic_gravity | | Adds gravitational force and friction |
|----------------------|--|---|
| In | in_particlesystem: particlesystem basic_parameters: complex center:float3 amount:float3 friction:float3 time_source: enum mass_calculations: complex mass_type: enum uniform_mass: float | The input particle system Center of gravity A multiplier for each coordinate of the gravity. Setting one or two to zero lets you simulate s.th. like a radial or a planar center of gravity The friction, will limit the end velocity like particles in air Where to take the time from, one of sequencer or real One of: <ul style="list-style-type: none"> - individual: take each particle's size as its mass - uniform: instead use the uniform_mass parameter If mass_type = uniform, this will be each particle's mass |
| Out | particlesystem: particlesystem | The output particle system |

Table 8-118.

| basic_wind_deformer | | Applies wind |
|----------------------------|--|---|
| In | in_particlesystem: particlesystem wind:float3 | The input particle system Wind speed |
| Out | particlesystem: particlesystem | The output particle system |

Table 8-119.

| floor | Simulates a floor, where particles stop their movement or bounce off |
|--|---|
| In <code>in_particlesystem: particlesystem</code> <code>axis:complex</code> <code>x:complex</code> | The input particle system |
| <code>x_floor:enum</code> | One of no or yes. Whether to enable a floor at <code>x = floor-x</code> |
| <code>x_bounce: enum</code> | One of no or yes. Whether to enable bouncing once the floor is hit. |
| <code>x_loss:float</code> | Only if bouncing, the loss in velocity experienced at the bounce |
| <code>y:complex</code> | |
| <code>y_floor:enum</code> | One of no or yes. Whether to enable a floor at <code>y = floor-y</code> |
| <code>y_bounce: enum</code> | One of no or yes. Whether to enable bouncing once the floor is hit. |
| <code>y_loss:float</code> | Only if bouncing, the loss in velocity experienced at the bounce |
| <code>z:complex</code> | |
| <code>z_floor:enum</code> | One of no or yes. Whether to enable a floor at <code>z = floor-z</code> |
| <code>z_bounce: enum</code> | One of no or yes. Whether to enable bouncing once the floor is hit. |
| <code>z_loss:float</code> | Only if bouncing, the loss in velocity experienced at the bounce |
| <code>refraction:enum</code> | One of no or yes. If yes, and also bouncing is enabled, enable refraction, which means that particles will bounce off in more or less random directions |
| <code>refraction_ amount:float3</code> | If refraction is enabled, the amount refraction will happen in each dimension |
| <code>floor:float3</code> | The floor-x, floor-y and floor-z values |
| Out <code>particlesystem: particlesystem</code> | The output particle system |

Table 8-120.

| | |
|--------------------------------------|--|
| <code>particle_fluid_deformer</code> | This module is currently under construction and not functional |
|--------------------------------------|--|

Table 8-121.

| size_mult | | Changes the particles' size, adds or multiplies a number |
|------------------|--|---|
| In | <code>in_particlesystem: particlesystem</code> | The input particle system |
| | <code>strength:float</code> | Amount of change |
| | <code>size_type:enum</code> | One of multiply or add |
| Out | <code>particlesystem: particlesystem</code> | The output particle system |

Table 8-122.

| size_noise | | Changes the particles' size randomly, adds or multiplies a number |
|-------------------|--|--|
| In | <code>in_particlesystem: particlesystem</code> | The input particle system |
| | <code>strength:float</code> | Amount of change |
| | <code>size_type:enum</code> | One of multiply or add |
| Out | <code>particlesystem: particlesystem</code> | The output particle system |

Renderers

Once the data stream reaches a renderer in a pipeline, we arrive in the OpenGL world of objects. That means after the rendering step we will have points, lines, polygons, quads, and other graphics primitives that will eventually produce output on the screen.

The modification steps to render data, such as orientation, rendering modes, and the like are described in this section. That is, modules that have a renderer input and a renderer output anchor are explained here as well.

Renderers are divided into sections according to what they render or what they do to rendered objects.

Basic

Basic renderers. The path is Renderers → Basic. See Tables 8-123 to 8-127.

Table 8-123.

| colored_rectangle | | Draws a colored rectangle |
|--------------------------|-----------------------|--|
| In | spatial:complex | |
| | position:float3 | The center of the rectangle |
| | angle:float | Rotation angle |
| | rotation_axis: float3 | The rotation axis |
| | size:float | A size multiplier. If this reads 1.0, the size will be 2x2 |
| | border:complex | |
| | border_enabled: enum | One of no or yes. |
| | border_width: float | The border width, if enabled |
| | border_color: float4 | The border color, if enabled |
| | color:float4 | The area color |
| Out | render_out:render | The renderer output |

Table 8-124.

| points | | A very simple renderer drawing the four corner points of a rectangle on the screen. Consider this kind of a “hello world” program |
|---------------|-----------------------|--|
| In | spatial:complex | |
| | position:float3 | The center of the rectangle |
| | angle:float | Rotation angle |
| | rotation_axis: float3 | The rotation axis |
| | size:float | A size multiplier. If this reads 1.0, the size will be 2x2 |
| Out | render_out:render | The renderer output |

Table 8-125.

| render_line | | Draws a line on the screen |
|--------------------|-------------------|---|
| In | spatial:complex | |
| | point_a:float3 | Start point |
| | point_b:float3 | End point |
| | color_a:float4 | Start color |
| | color_b:float4 | End color. Colors in between are interpolated linearly in a RGBA color space. |
| | width:float | Width. Note that depending on your hardware an upper limit might apply |
| Out | render_out:render | The renderer output |

Table 8-126.

| textured_rectangle | | Important renderer; draws a rectangle with a texture on the screen |
|---------------------------|-----------------------|--|
| In | spatial:complex | |
| | position:float3 | The position of the rectangle's center. |
| | size:float | A size multiplier. If this is 1.0, the size will be 2x2 since coordinates are inside [-1;- 1] x[1;1] |
| | angle:float | Rotation angle about the z-axis |
| | x_aspect_ratio: float | If you want to apply an aspect ratio, put something unequal 1.0 here |
| | tex_coord_a: float3 | The texture coordinate of the lower-left corner. The third coordinate here is unused |
| | tex_coord_b: float3 | The texture coordinate of the upper-right corner. The third coordinate here is unused |
| | facing_camera: enum | One of no or yes. If yes, use a special feature of OpenGL where the rotation is ignored and the rectangle is drawn right face to the camera. |

(continued)

Table 8-126. (continued)

| textured_rectangle | | Important renderer; draws a rectangle with a texture on the screen |
|---------------------------|--------------------------|--|
| | color:complex | |
| | global_alpha: float | An alpha multiplier. Just applied to the ALPHA component of the color multiplier, and as such exists only for convenience. |
| | color_multiplier: float4 | As a multiplier applied to all the colors. As stated, its ALPHA component first is multiplied by global_alpha |
| | color_center: float4 | The center color |
| | color_a:float4 | Color bottom-left corner |
| | color_b:float4 | Color top-left corner |
| | color_c:float4 | Color top-right corner |
| | color_d:float4 | Color bottom-right corner |
| | texture_in:texture | The input texture |
| Out | render_out:render | The renderer output |

Table 8-127.

| textured_triangle | | A textured triangle |
|--------------------------|---------------------|-----------------------------|
| In | spatial:complex | |
| | position_a:float3 | Position corner A |
| | position_b:float3 | Position corner B |
| | position_c:float3 | Position corner C |
| | tex_coord_a: float3 | Texture coordinate corner A |
| | tex_coord_b: float3 | Texture coordinate corner B |
| | tex_coord_c: float3 | Texture coordinate corner C |

(continued)

Table 8-127. (continued)

| textured_triangle | A textured triangle |
|--------------------------|--|
| color:complex | |
| global_alpha: float | An alpha multiplier. Just applied to the ALPHA component of the color multiplier, and as such exists only for convenience. |
| color_multiplier: float4 | As a multiplier applied to all the colors. As stated, its ALPHA component first is multiplied by global_alpha |
| color_a:float4 | Color corner A |
| color_b:float4 | Color corner B |
| color_c:float4 | Color corner C |
| texture_in:texture | The input texture |
| Out render_out:render | The renderer output |

For the textured rectangle and textured triangle we have two competing color concepts: first the color as given by the vertex colors, second the color as given by the texture. It is left to the rest of the rendering pipeline and the internal state of OpenGL which one wins, or how they get blended. The standard blending mode in ThMAD is MODULATE, which means color values just get multiplied. Unfortunately, we cannot easily change this on a module basis, since OpenGL follows the notion of a *texture unit* and only for each texture unit the way textures are combined can be set by using standard API functions. But there is another way of controlling this on a module basis, and it is even more powerful.

Using shaders instead of functions in fact gives you tremendous power to define the blending. You can specify a shading program by adding the Renderers → Shaders → glsl_loader module behind the renderer; see Figure 8-6.

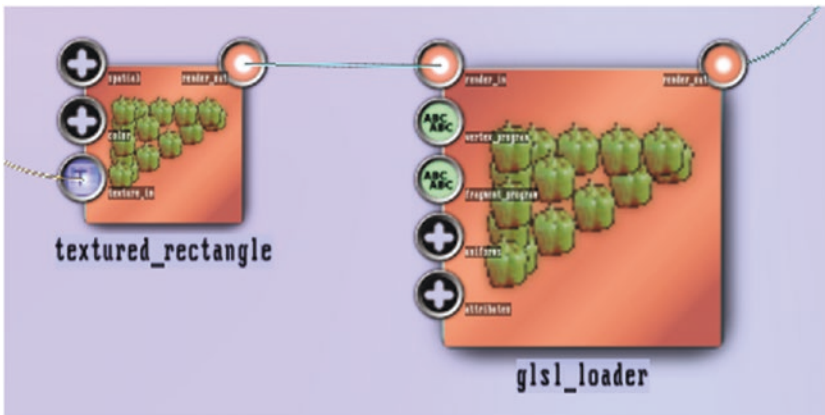


Figure 8-6. Using shaders to control the color blending

For example, the following code will allow you to *add* color values instead of multiplying them:

Vertex shader:

```

1  varying vec4 vColor;
2  void main(void) {
3      vColor = gl_Color;
4      gl_TexCoord[0] = gl_MultiTexCoord0;
5      gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
6  }
```

Fragment shader:

```

1  varying vec4 vColor;
2  uniform sampler2D sampler;
3  void main(void) {
4      vec4 tex = texture2D ( sampler,
5          gl_TexCoord[0].st );
6      gl_FragColor = vec4(vColor.r + tex.r,
7          vColor.g + tex.g,
8          vColor.b + tex.b,
9          vColor.a + tex.a);
10 }
```

The blending algorithm is shown in lines 6-9 of the fragment shader code.

Mesh

Mesh related renderers. The path is Renderers → Mesh. See Tables [8-128](#) to [8-134](#).

Table 8-128.

| mesh_basic_render | Basic renderer for meshes |
|---------------------------|--|
| In tex_a:texture | A texture for the surface described by the mesh. This anchor is mandatory, if unconnected the vertex colors will be used if defined and anchors options / vertex_colors and options / use_vertex_colors are set to yes |
| mesh_in:mesh | The main mesh input. Defines vertices, normals, tangents, faces and texture coordinates. |
| particles: particlesystem | Connect a particle system to this anchor if you want to have the mesh multiplied. Mandatory. |
| particle_cloud: mesh | Connect another mesh B here if you want to multiply the mesh. Compared to the particles anchor this is more for static systems. Mandatory. |

(continued)

Table 8-128. (continued)

| mesh_basic_render | Basic renderer for meshes |
|---------------------------------|---|
| options:complex | |
| vertex_colors: enum | One of no or yes. Whether to make a call to <code>glEnable(GL_COLOR_MATERIAL)</code> . If yes, the color of the vertices themselves will be taken into account |
| use_display_list:enum | One of no or yes. If yes, use a static drawing mode, which performs better but is less flexible. |
| use_vertex_colors:enum | One of no or yes. Whether to send vertex colors as an array to the graphics hardware. If you want to have a coloring based on the vertex colors, set this to yes |
| particles_size_center:enum | One of no or yes. If yes, and only if also the <code>particles</code> input anchor is connected, make particles bigger when farther away from their birth position. |
| particles_size_from_color: enum | One of no or yes. If yes, and only if also the <code>particles</code> input anchor is connected, misuse the vertex color R, G, B values to define a scaling of the particles x, y, and z sizes. |
| ignore_uv_in_vbo_updates: enum | One of no or yes. If yes, and only in dynamic mode (<code>use_display_list = false</code>), upload vertex texture coordinates to the graphics hardware only at the beginning. This is a performance optimization. |
| Out render_out: render | The renderer output |

Table 8-129.

| | | |
|----------------------------|--|--|
| mesh_dot_billboards | | Renders dots at the mesh positions. It is called “billboard”, because the dots all have the same size on the screen, no matter how far away from the eye/camera position. |
| In | <p>mesh_in:mesh</p> <p>base_color:float4</p> <p>dot_size:float</p> <p>use_display_list: enum</p> <p>shader_params: complex</p> <p> vertex_program:string</p> <p> fragment_program:string</p> <p> uniforms: complex</p> <p> attributes: complex</p> | <p>The input mesh</p> <p>One possibility for the dot color is to use this value. The coloring is done by shader code, so this may or may not be the case.</p> <p>The dot size. May or may not be used, depending on the shader code.</p> <p>One of no or yes. Whether to use display lists. If yes, improves performance.</p> <p>The vertex program. Will receive dot_size as a <i>uniform</i> float. You can usually leave it as it is. If you want to use the shader to set the dot size, write s.th. like <code>gl_PointSize = 7.0;</code></p> <p>The fragment program. Here you can say whether to use the vertex color (<code>gl_FragColor = vColor;</code>) or the base_color (<code>gl_FragColor = base_color;</code>).</p> <p>Automatic uniforms. These are mirrors of base_color and dot_size. Use either them or the originals to control the module.</p> <p>Automatic attributes. Unused.</p> |
| Out | render_out: render | The renderer output. |

Table 8-130.

| | | |
|------------------------|-----------------------|---|
| mesh_dot_render | | This is the little brother of the module mesh_dot_billboards. Draws points at each vertex, but doesn't use shader code |
| In | mesh_in:mesh | The input mesh |
| | base_color:float4 | Unless use_vertex_color is set to true, this is the point color. |
| | dot_size:float | The dot size |
| | use_vertex_color:enum | One of no or yes. Whether to use colors set at the vertices themselves. |
| Out | render_out: render | The renderer output. |

Table 8-131.

| | | |
|-------------------------|---------------------------|--|
| mesh_line_render | | Draws lines between vertices. This will usually not create beauty, but it shows you the sequence of vertices. If you want a real wireframe, use Renderers → opengl_modifiers → rendering_mode instead |
| In | mesh_in:mesh | The input mesh |
| | base_options: complex | |
| | line_width:float | The line width. Usually your graphics hardware will impose an upper limit on that. |
| | override_base_color: enum | One of no or yes. If yes, the base_color will be used. Otherwise the vertices' colors. |
| | base_color: float4 | The color used for the line. Applies only if override_base_color is set to true |
| | base_color_add:float4 | Just added to base_color |
| | center_options: complex | |
| | each_to_center:enum | One of no or yes. If yes, draw a line from each vertex to the origin (0,0,0) instead |
| | center_color: float4 | The color used at the center |
| | center_color_add:float4 | Just added to center_color |
| Out | render_out: render | The renderer output |

Table 8-132.

| | | |
|---------------------------------|---------------------|--|
| mesh_transparency_render | | Renders a mesh in distance order to eye/camera. Since while blending transparent textures the order matters, first sorting the order gives more realistic effects |
| In | tex_a:texture | The texture to map on the surface of the mesh |
| | mesh_in:mesh | The input mesh |
| | vertex_colors: enum | One of no or yes. Whether to use the colors adjoint to the vertices. If yes, a MODULATE type blending with the texture colors will happen (color values get multiplied). |
| Out | render_out: render | The renderer output. |

Table 8-133.

| | | |
|------------------------------|--------------------|--|
| mesh_vertex_id_render | | Renders the vertex IDs (diagnosis tool) |
| In | mesh_in:mesh | The input mesh |
| | base_color:float4 | The color of the drawn IDs |
| | font_size:float | A scaling for the font size |
| | min_box:float3 | A lower limit for vertex coordinates. If any of x, y, z is below the appropriate value of this parameter, the vertex will be ignored |
| | max_box:float3 | An upper limit for vertex coordinates. If any of x, y, z is above the appropriate value of this parameter, the vertex will be ignored |
| | max_id:float | The maximum vertex ID to render. Corresponds to the maximum number of vertices shown. If set to s.th. less than 0.0, no limit will apply |
| Out | render_out: render | The renderer output |

Table 8-134.

| | |
|--------------------|--|
| render_mesh_ribbon | This module is under development and currently experimental (there are performance issues) |
|--------------------|--|

OpenGL Modifiers

The modules of this subsection are closely related to OpenGL settings and functions. They thus can be considered more low-level compared to the other ThMAD modules. Nevertheless some of them are vital for your states. The path is Renderers → opengl_modifiers. See Tables 8-135 to 8-158.

Table 8-135.

| | | |
|-------------------------|--------------------|---|
| auto_normalize | | Use this before applying lighting to any scene where scalings may happen. Allows for an automatic recalculation of the normal vectors which are necessary for correct lighting calculations. This module has no effect when shaders are at work |
| In | render_in: render | The renderer input |
| Out | render_out: render | The renderer output |
| backface_culling | | If faces are defined (areas, at the smallest scale defined by three vertices), uses the rotation defined by the vertices sorting (left or right) to find out whether we are looking at it from front or from behind. If enabled, parts at the back of the view simply get discarded |
| In | render_in: render | The renderer input |
| | status:enum | One of ENABLED or DISABLED. Whether to enable backface culling |
| Out | render_out: render | The renderer output |

Table 8-136.

| blend_mode | | Determines what happens when something is painted over something already there |
|-------------------|---|---|
| In | render_in: render source_blend: enum | The renderer input What happens with the pixels that newly arrive. For details, look at the OpenGL specification (or search for gl_blend in your favorite search engine). The default setting is SRC_ALPHA and it paints newly arriving pixels at the intensity defined by their ALPHA, but lets existing pixels shine through by an intensity of 1 - ALPHA. |
| | dest_blend: enum | What happens with the pixels that are already there. The default setting is ONE_MINUS_SRC_ALPHA, which means they get faded out at an intensity defined by 1 - ALPHA of the incoming pixels |
| | blend_color: enum | An external blend color, if one is needed depending on the rules chosen by the previous anchors |
| Out | render_out: render | The renderer output |

Table 8-137.

| buffer_clear | | Clears the output at this stage in the pipeline |
|---------------------|---|--|
| In | render_in: render color_buffer: enum | The renderer input One of no or yes. Whether you want to clear the color buffer |
| | clear_color: float4 | If the color buffer is cleared by this module, use this color for the clearing |
| | depth_buffer: enum | One of no or yes. Whether to clear the depth buffer |
| Out | render_out: render | The renderer output |

Table 8-138.

| cameras → freelook_camera | | A camera at a certain position, rotated in a certain way and with a certain up-vector |
|----------------------------------|--------------------------|--|
| In | render_in:render | The renderer output |
| | camera:complex | |
| | position:float3 | Position in space |
| | rotation:float3 | The rotation |
| | upvector:float3 | The upvector |
| | fov:float | The angle of the <i>camera field of view</i> |
| | near_clipping: float | Anything nearer than this distance will be discarded |
| | far_clipping: float | Anything farther than this distance will be discarded |
| | perspective_correct:enum | One of no or yes. If yes, take the display's aspect ratio into account |
| | ortho:complex | |
| | enable_ortho: enum | One of no or yes. If yes, use an orthographic projection (no perspective) instead of a perspective projection. The fov and near_clipping and far_clipping anchors will play no role then |
| | ortho_left:float | If ortho enabled, the left edge |
| | ortho_right: float | If ortho enabled, the right edge |
| | ortho_bottom: float | If ortho enabled, the bottom edge |
| | ortho_top:float | If ortho enabled, the top edge |
| | ortho_near: float | If ortho enabled, the near clipping plane |
| | ortho_far:float | If ortho enabled, the far clipping plane |
| Out | render_out:render | The renderer output |

Table 8-139.

| camera → orbit_ | | A camera that looks at a point, and you can define a position in an orbit to observe that point from any angle | |
|------------------------|---|---|--|
| In | <code>render_</code> <code>in:render</code> <code>camera:complex</code> | The renderer output | |
| | | <code>rotation:float3</code> | The rotation, that is where on the orbit we are |
| | | <code>distance:float</code> | The distance of the orbit from the point we are looking at |
| | | <code>destination:float3</code> | The point we are looking at |
| | | <code>upvector:float3</code> | The upvector |
| | | <code>fov:float</code> | The angle of the <i>camera field of view</i> |
| | | <code>near_clipping:float</code> | Anything nearer than this distance will be discarded |
| | | <code>far_clipping:float</code> | Anything farther than this distance will be discarded |
| | | <code>perspective_</code> <code>correct:enum</code> | One of no or yes. If yes, Take the display's aspect ratio into account |
| | <code>ortho:complex</code> | <code>enable_ortho:enum</code> | One of no or yes. If yes, use an orthographic projection (no perspective) instead of a perspective projection. The fov and <code>near_clipping</code> and <code>far_clipping</code> anchors will play no role then |
| | | <code>ortho_left:float</code> | If ortho enabled, the left edge |
| | | <code>ortho_right:float</code> | If ortho enabled, the right edge |
| | | <code>ortho_bottom:float</code> | If ortho enabled, the bottom edge |
| | | <code>ortho_top:float</code> | If ortho enabled, the top edge |
| | | <code>ortho_near:float</code> | If ortho enabled, the near clipping plane |
| | | <code>ortho_far:float</code> | If ortho enabled, the far clipping plane |
| Out | <code>render_</code> <code>out:render</code> | The renderer output | |

Table 8-140.

| camera → target_camera | | A camera looking from one point to another. Useful if you want to follow objects or points |
|-------------------------------|--------------------------|--|
| In | render_in:render | The renderer input |
| | camera:complex | |
| | position:float3 | Position in space |
| | destination:float3 | The point we are looking at |
| | upvector:float3 | The upvector |
| | fov:float | The angle of the <i>camera field of view</i> |
| | near_clipping:float | Anything nearer than this distance will be discarded |
| | far_clipping:float | Anything farther than this distance will be discarded |
| | perspective_correct:enum | One of no or yes. If yes, take the display's aspect ratio into account |
| | ortho:complex | |
| | enable_ortho:enum | One of no or yes. If yes, use an orthographic projection (no perspective) instead of a perspective projection. The "fov" and "near_clipping" and "far_clipping" anchors will play no role then |
| | ortho_left:float | If ortho enabled, the left edge |
| | ortho_right:float | If ortho enabled, the right edge |
| | ortho_bottom:float | If ortho enabled, the bottom edge |
| | ortho_top:float | If ortho enabled, the top edge |
| | ortho_near:float | If ortho enabled, the near clipping plane |
| | ortho_far:float | If ortho enabled, the far clipping plane |
| Out | render_out:render | The renderer output |

Table 8-141.

| | | |
|---------------------|--|--|
| depth_buffer | | Unless we need transparent surfaces, pixels that are behind other pixels do not need to be drawn. OpenGL can save this depth of pixels and allows for a check of this relation |
| In | render_in:render depth_test:enum depth_mask:enum | The renderer input One of ENABLED or DISABLED. Enable it to have the depth check performed One of ENABLED or DISABLED. Enable it to have a buffer stored the depth information |
| Out | render_out:render | The renderer output |

Table 8-142.

| | | |
|---------------------------|-------------------|---|
| depth_buffer_clear | | Clear the depth buffer at this place in the pipeline |
| In | render_in:render | The renderer input |
| Out | render_out:render | The renderer output |

Table 8-143.

| | | |
|-----------------------|-------------------------------------|--|
| depth_function | | If depth test is enabled, see module depth_buffer, which specifies the test function |
| In | render_in:render depth_func:enum | The renderer input One of: <ul style="list-style-type: none"> – NEVER: Test always fails (last pixel per position always wins) – LESS: The default – nearer or same depth pixels (z-axis in view direction) win – EQUAL: Same depths will not be drawn – LESS_OR_EQUAL: nearer pixels win – GREATER: pixels at the same depth or farther away win – NOT_EQUAL: Newer pixels will only show if at the same depth – GREATER_OR_EQUAL: pixels farther away win – ALWAYS: Test always passes (first pixel per position always wins) |
| Out | render_out:render | The renderer output |

Table 8-144.

| | | |
|-----------------|-------------------|--|
| gl_color | | Sets the vertex color for modules that do not do it themselves. This will <i>not</i> override module colors |
| In | render_in:render | The renderer input |
| | color:float4 | The color |
| Out | render_out:render | The renderer output |

Table 8-145.

| | | |
|---------------|-------------------|--|
| gl_fog | | OpenGL's built-in fog alters colors on a pixel basis. This module allows for setting some fog parameter for the <i>linear</i> mode: Calculate some factor |
| | | $f = (\text{end}-c) / (\text{end}-\text{start})$ |
| | | where <i>c</i> is the distance from the camera to the origin. The blending function is then |
| | | $C = f \cdot \text{color} + (1-f) \cdot \text{fog_color}$ |
| | | Fog does not affect the ALPHA channel! |
| In | render_in:render | The renderer input |
| | status:enum | One of ENABLED or DISABLED |
| | fog_color:float4 | The fog color |
| | fog_start:float | Start parameter |
| | fog_end: | End parameter |
| Out | render_out:render | The renderer output |

Table 8-146.

| | | |
|-------------------|-------------------|---|
| gl_frustum | | Applies perspective to the current PROJECTION matrix |
| In | render_in:render | The renderer input |
| | left:float | The left edge of the clipping plane |
| | right:float | The right edge of the clipping plane |
| | bottom:float | The bottom of the clipping plane |
| | top:float | The top of the clipping plane |
| | near:float | The near distance |
| | far:float | The far distance |
| Out | render_out:render | The renderer output |

Table 8-147.

| | | |
|----------------------------------|----------------------|--|
| gl_get_camera_orientation | | Gets the camera orientation, this is the MODELVIEW matrix multiplied by (0,0,1). Sounds strange, but do not place this module after a camera module in a sub-pipeline |
| In | render_in:render | The renderer input |
| Out | render_out:render | The renderer output |
| | direction_out:float3 | The orientation |

Table 8-148.

| | | |
|----------------------|-------------------|---|
| gl_line_width | | Sets the line width if modules are using lines and don't set the width themselves. Will not override settings if done by the modules |
| In | render_in:render | The renderer input |
| | width:float | The width |
| Out | render_out:render | The renderer output |

Table 8-149.

| | | |
|----------------------|--------------------|--|
| gl_matrix_get | | Gets the MODELVIEW, PROJECTION, or TEXTURE matrix |
| In | render_in:render | The renderer input |
| | matrix_target:enum | One of MODELVIEW, PROJECTION, or TEXTURE |
| Out | render_out:render | The renderer output |
| | matrix_out:matrix | The 4x4 matrix selected |

Table 8-150.

| gl_matrix_multiply | | Multiplies one of MODELVIEW, PROJECTION, or TEXTURE matrix with the matrix given |
|---------------------------|--|---|
| In | render_in:render matrix_in:matrix matrix_target:enum | The renderer input The 4x4 matrix used for the multiplication One of MODELVIEW, PROJECTION, TEXTURE |
| Out | render_out:render | The renderer output |
| gl_rotate | | |
| In | render_in:render axis:float3 angle:float | The renderer input The rotation axis The rotation angle |
| Out | render_out:render | The renderer output |

Table 8-151.

| gl_rotate_quat | | Rotates as described by a quaternion |
|-----------------------|---|---|
| In | render_in:render rotation:quaternion matrix_target:enum invert_rotation:enum | The renderer input The rotation quaternion One of MODELVIEW, PROJECTION, or TEXTURE. In which matrix stack the rotation is supposed to occur One of no or yes. If yes, invert the rotation |
| Out | render_out:render | The renderer output |

Table 8-152.

| gl_scale | | Does a scaling |
|-----------------|---|---|
| In | render_in:render scale:float3 matrix:enum | The renderer input The scaling factor in each dimension One of MODELVIEW, PROJECTION, or TEXTURE. In which matrix stack the scaling is supposed to occur. |
| Out | render_out:render | The renderer output |

Table 8-153.

| gl_scale_one | | Does a scaling, the factor for each dimension the same |
|---------------------|-------------------|---|
| In | render_in:render | The renderer input |
| | scale:float | The scaling factor far all dimensions |
| | matrix:enum | One of MODELVIEW, PROJECTION, or TEXTURE. In which matrix stack the scaling is supposed to occur. |
| Out | render_out:render | The renderer output |

Table 8-154.

| gl_translate | | Translates all objects |
|---------------------|--------------------|-------------------------------|
| In | render_in:render | The renderer input |
| | translation:float3 | The translation vector |
| Out | render_out:render | The renderer output |

Table 8-155.

| light_directional | | A directional light. “Directional” means the lighting is calculated as if all light beams come in in a parallel fashion |
|--------------------------|------------------------|--|
| In | render_in:render | The renderer input |
| | properties:complex | |
| | light_id:enum | The light ID. If you have more than one light, all lights must have different IDs |
| | enabled:enum | One of no or yes. Switch the light on or off. |
| | position:float3 | Where the light comes from. Defines the light beams direction |
| | ambient_color: float4 | An ambient color. Will shine on all surfaces and independent of the light’s position |
| | diffuse_color: float4 | A diffuse color. Will refract into all directions when surfaces are hit |
| | specular_color: float4 | A specular color. Will be reflected at a (more or less) precise angle once surfaces are hit |
| Out | render_out:render | The renderer output |

Table 8-156.

| light_model | The light model to use |
|--|---|
| In render_in:render properties:complex | The renderer input |
| ambient_color: float4 | An ambient color to use independent of any light. Ambient colors will shine on any surface, no matter how positioned |
| color_control: enum | One of: <ul style="list-style-type: none"> – SINGLE_COLOR: Standard color calculation for all types of light – SEPARATE_SPECULAR_COLOR: Use a special variable for the specular color. |
| local_viewer:enum | One of: <ul style="list-style-type: none"> – Z-AXIS: Use a point on the z-axis for calculating the final color value – EYE_COORDS: Use the camera's position for calculating the final color value. |
| num_sides:enum | One of ONE or TWO. If TWO, let light shine on the backside of surfaces as well (ThMAD however cannot handle that) |
| Out render_out:render | The renderer output |

Table 8-157.

| material_param | Material color related parameters |
|----------------------------|--|
| In render_in:render | The renderer input |
| faces_affected:enum | One of front_facing, back_facing, or front_and_back |
| properties:complex | |
| ambient_reflectance:float4 | The color used for ambient reflectance. Naturally you would use something dark here |
| diffuse_reflectance:float4 | The color used for diffuse reflectance. Usually you put the natural surface's color here |
| specular_reflectance:foat4 | The color used for specular reflectance. Usually you put something bright or very bright here |
| emission_intensity: float4 | Some color that is emitted no matter what light shines on the surface |
| specular_exponent: float | Defines the fuzziness of specular reflectance. The greater, the more spotted the specular reflectance will appear. |
| Out render_out:render | The renderer output |

Table 8-158.

| rendering_mode | The rendering mode specifies what happens with the lines and areas between vertices |
|--------------------------|--|
| In render_in:render | The renderer input |
| back_facing:enum | One of: <ul style="list-style-type: none"> – points: Render only points at the vertices – lines: Render lines between adjacent vertices, but do not paint the faces – solid: Paint the area between adjacent vertices |
| front_facing:enum | Same values as back_facing |
| smooth_edges:enum | One of no or yes. Whether to draw smooth edges. |
| Out render_out:render | The renderer output |

Oscilloscopes

Oscilloscopes serve diagnostic purposes for seeing what controller modules do in the course of time. The path is Renderers → Oscilloscopes. See Tables 8-159 and 8-160.

Table 8-159.

| simple_colorline | | Draws a line with segments colored according to the values inside an input array |
|-------------------------|-----------------------|---|
| In | spatial:complex | |
| | position:float3 | The position of the surrounding (invisible) box in space |
| | angle:float | The angle around the rotation axis |
| | rotation_axis: float3 | The rotation axis |
| | size:float | The size of the surrounding box |
| | color_a:float4 | Color for array-values zero |
| | color_b:float4 | Color for array values 1.0 - colors in between will be linearly interpolated |
| | line_width:float | The line width. Note that your graphics hardware might impose an upper limit on that |
| Out | render_out:render | The renderer output |

Table 8-160.

| simple_oscilloscope | | An oscilloscope drawing elongations designated by floats from an incoming float array |
|----------------------------|--|--|
| In | <code>data_in:float_array</code> <code>spatial:complex</code> <code>position:float3</code> <code>angle:float</code> <code>rotation_axis: float3</code> <code>size:float</code> <code>color:float4</code> <code>line_width:float</code> <code>axes:complex</code> <code>paint_y_zero: enum</code> <code>axes_color: float4</code> | Incoming data The position of the surrounding (invisible) clipping box in space The angle around the rotation axis The rotation axis The size of the surrounding box The painting color The line width. Note that your graphics hardware might impose an upper limit on that One of no or yes. Whether to paint a val=0 line The color used for the axes |
| Out | <code>render_out: render</code> | The renderer output |

While the oscilloscope is more a diagnostic tool, you might need to ensure its usability by prepending a `Maths → Array → float_array_memory_buffer` module for collecting floats, the `simple_colorline` can be used to produce visualizations. While a line by itself is a little boring, using a blurring effect might greatly improve the number of usage scenarios.

Particlessystems

Particlessystem related renderers. The path is `Renderers → particlessystems`. See Tables 8-161 to 8-166.

Table 8-161.

| render_particle_center | | A special effects particlesystem renderer disobeying physics laws by clamping some coordinates to a center point |
|-------------------------------|--------------------------------|---|
| In | particlesystem: particlesystem | The particle system to render |
| | texture:texture | The texture used for drawing each particle |
| | position:float3 | A base position used for rendering the particle system |
| | alpha:float | An artificial additional ALPHA value; all colors from the particles just have their RGB values (not the ALPHA!) multiplied by that number |
| | size:float | A size to apply for each particle |
| Out | render_out:render | The renderer output |

Table 8-162.

| render_particle_ribbon | | Renders particles along a moving gravitational ribbon in 3D |
|-------------------------------|--------------------------------|--|
| In | particlesystem: particlesystem | The particle system to render |
| | params:complex | |
| | ribbon_width: float | The width of the ribbon |
| | length:float | The length of the ribbon |
| | friction:float | A frictional parameter (the particles may change their speed while moving along the ribbon trajectory) |
| | step_length: float | A factor for the movement speed of the particles along the ribbon trajectory |
| | color0:float4 | A color for the particles, will have a decreasing ALPHA during the lifetime of each particle |
| | color1:float | Another static color parameter. |
| Out | render_out:render | The renderer output |

Table 8-163.

| render_particle_shader | A more advanced particle system renderer which lets you define lifetime behavior and lets you use shader code for maximum rendering flexibility |
|--|--|
| <p>In</p> <p>particlesystem: particlesystem</p> <p>texture:texture</p> <p>options:complex</p> <p> size_lifespan_type:enum</p> <p> size_lifespan_sequence:</p> <p> alpha_lifespan_sequence:</p> <p> color_lifespan_type:enum</p> <p> r_lifespan_sequence:</p> <p> g_lifespan_sequence:</p> <p> b_lifespan_sequence:</p> <p> ignore_particles_att_center:enum</p> <p> shader_params: complex</p> <p> vertex_program: string</p> <p> fragment_program:string</p> <p>Out</p> <p>render_out:render</p> | <p>The particle system to render</p> <p>A texture used to render the particles</p> <p>How the size during the lifetime is calculated:</p> <ul style="list-style-type: none"> - normal: increase linearly during lifetime - sequence: as specified by the sequence <p>The size sequence if size_lifespan_type is set to sequence.</p> <p>How the ALPHA of the particles develop during their lifetime</p> <p>How the colors during the lifetime are calculated:</p> <ul style="list-style-type: none"> - normal: just take the specified particle colors - sequence: as specified by the sequences <p>The RED sequence if color_lifespan_type is set to sequence</p> <p>The GREEN sequence if color_lifespan_type is set to sequence</p> <p>The BLUE sequence if color_lifespan_type is set to sequence</p> <p>One of no or yes. Whether to draw particles which are at or very close to the center</p> <p>The vertex program. Use this to tweak positions</p> <p>The fragment program. Use this to tweak the coloring</p> <p>The renderer output</p> |

Table 8-164.

| render_particlesystem_ext | | The little brother of the module render_particle_shader. Just always uses sequences for all the lifetime parameters size, alpha, color channels |
|----------------------------------|---|--|
| In | particlesystem: particlesystem texture:texture options:complex size_lifespan_ sequence: sequence alpha_lifespan_ sequence: sequence r_lifespan_ sequence: sequence g_lifespan_ sequence: sequence b_lifespan_ sequence: sequence Ignore_particles_att_center:enum vertex_program: string fragment_program: string | The particle system to render A texture used to render the particles The size sequence if size_lifespan_type is set to sequence. How the ALPHA of the particles develop during their lifetime The RED sequence if color_lifespan_type is set to sequence The GREEN sequence if color_lifespan_type is set to sequence The BLUE sequence if color_lifespan_type is set to sequence One of no or yes. Whether to draw particles which are at or very close to the center The vertex program. Use this to tweak positions The fragment program. Use this to tweak the coloring |
| Out | render_out:render | The renderer output |

Table 8-165.

| | | |
|---------------|---|--|
| simple | | The basic particle system renderer. Similar to the <code>render_particle_shader</code> module, but allows for choosing between texture rendering (no shader code used) and point sprite rendering (with shader code used) |
| In | <code>particlesystem:</code> <code>particlesystem</code> | The particle system to render |
| | <code>texture:texture</code> | A texture used to render the particles |
| | <code>options:complex</code> | |
| | <code>render_type:enum</code> | One of <code>quads</code> or <code>point_sprites</code> |
| | <code>size_lifespan_type:enum</code> | How the size during the lifetime gets calculated: <code>normal</code> : increase linearly during lifetime <code>sequence</code> : as specified by the sequence |
| | <code>size_lifespan_sequence:sequence</code> | The size sequence if <code>size_lifespan_type</code> is set to <code>sequence</code> . |
| | <code>alpha_lifespan_sequence:sequence</code> | How the ALPHA of the particles develop during their lifetime |
| | <code>color_lifespan_type:enum</code> | How the colors during the lifetime gets calculated: – <code>normal</code> : just take the specified particle colors – <code>sequence</code> : as specified by the sequences |
| | <code>r_lifespan_sequence:sequence</code> | The RED sequence if <code>color_lifespan_type</code> is set to <code>sequence</code> |
| | <code>g_lifespan_sequence:sequence</code> | The GREEN sequence if <code>color_lifespan_type</code> is set to <code>sequence</code> |
| | <code>b_lifespan_sequence:sequence</code> | The BLUE sequence if <code>color_lifespan_type</code> is set to <code>sequence</code> |
| | <code>Ignore_particles_att_center:enum</code> | One of <code>no</code> or <code>yes</code> . Whether to draw particles which are at or very close to the center |
| | <code>shader_params:complex</code> | |
| | <code>vertex_program:string</code> | The vertex program. Use this to tweak positions |
| | <code>fragment_program:string</code> | The fragment program. Use this to tweak the coloring |
| Out | <code>render_out:render</code> | The renderer output |

Table 8-166.

| sparks | | Draws sparks between particles which approach each other too closely |
|---------------|-----------------------------------|--|
| In | particlesystem: particlesystem | The particle system to render. Should not be too big, because proximity calculations are expensive |
| | float_array_in: float_ array | An array that modulates the proximity distance when sparks emerge. Use for example some array you get from sound input |
| | proximity_level: float | The proximity level when sparks emerge |
| | color:float4 | The spark color |
| Out | render_out:render | The renderer output |

Shaders

The shaders in this subsection are special modules; they all use the same code basis but have arbitrary shader code at work. The latter is initialized with some code, but can be adjusted freely once the module is instantiated on the canvas.

One noticeable difference with all the other modules is that defaults for the parameters don't apply. So in most cases you have to carefully set all adjustable parameters before the shaders can work as expected.

The basis code leads to following parameters; see [Table 8-167](#).

Table 8-167. Shader Parameters

| <Name given by shader code> | Shader basis code | |
|-----------------------------|--------------------------|---|
| In | render_in:render | The renderer input |
| | vertex_program:string | The vertex shader code |
| | fragment_program: string | The fragment shader code |
| | uniforms:complex | Automatically generated anchors. ThMAD parses the shader code and exposes all uniforms it finds there |
| | attributes:complex | Automatically generated anchors for attributes. Attributes get assigned on a per- vertex basis, which makes them somewhat hard to handle in ThMAD |
| Out | render_out:render | The renderer output |

The following shaders exist; see Table 8-168.

Table 8-168. *Built-in Shaders*

| Name | Description |
|--|---|
| glsl_loader | This is the basic shader. It describes a basic shader code for grounds-up developing your own shaders. For the default shader code open the anchor of a freshly instantiated module |
| blend_modes / shader_blend_color_dodge | A dodge blend shader. For the default shader code open the anchor of a freshly instantiated module. |
| blend_modes / shader_blend_overlay | An overlay blend shader. For the default shader code open the anchor of a freshly instantiated module. |
| blend_modes / shader_blend_satadd | A saturated addition blend shader. For the default shader code open the anchor of a freshly instantiated module. |
| blend_modes / shader_blend_screen | A screen blend shader. For the default shader code open the anchor of a freshly instantiated module. |
| lighting_models / normal_map | A normal mapping lighting. For the default shader code open the anchor of a freshly instantiated module. |
| lighting_models / normal_map2 | Another normal mapping lighting. For the default shader code open the anchor of a freshly instantiated module. |
| lighting_models / shader_2l_diffmap_specmap | A diffmap/specmap lighting. For the default shader code open the anchor of a freshly instantiated module. |
| lighting_models / shader_smooth_lighting_tex | A lighting of a texture. For the default shader code open the anchor of a freshly instantiated module. |
| materials / chromatic_dispersion | A chromatic refraction shader. For the default shader code open the anchor of a freshly instantiated module. |
| texture_filters / blur_shader | A blurring shader. Takes a texture and applies a Gaussian blur. You need to set the texOffset to a value unequal (0,0) to see an effect (try small values like 0.02). For the default shader code open the anchor of a freshly instantiated module. |

Shaders are a powerful concept. Although this book cannot give a thorough introduction, you can start diving into that matter using the examples.

Text

Text related renderers. The path is Renderers → Text. See Table 8-169.

Table 8-169.

| text_s | Writes a string on a plane |
|-----------------------------|---|
| In text_in:string | The text |
| font_in:resource | The TTF font (appendix *.ttf) to use |
| render_type:enum | If BITMAP, render text as bitmap, if POLYGON, create polygons to render the text. Usually using bitmaps looks better. |
| align:enum | How to align multi-line text horizontally. One of LEFT, CENTER, or RIGHT. |
| limits / limit_align: float | If greater than 0.0, the integer part will pick a line from multi-line text (0 is the first line). |
| appearance:complex | |
| glyph_size:float | The glyph size. Will look inside the font data to find the best matching glyph set. |
| size:float | The size of the surrounding box. The text will scale automatically when you change this. |
| leading:float | The distance between lines of text |
| position:float3 | Position of the surrounding box |
| angle:float | Rotation angle of the surrounding box |
| rotation_axis: float3 | Rotation axis of the surrounding box |
| text_alpha:float | ALPHA of the text |
| outline_alpha:float | If in POLYGON mode, an ALPHA channel multiplier for an outlining |
| outline_color: float4 | If in POLYGON mode, the color for an outlining |
| outline_thickness: float | If in POLYGON mode, the line thickness used for an outlining |
| color:complex | Color of the text |
| red:float | Red |
| green:float | Green |
| blue:float | Blue |
| Out render_out:render | The renderer output |

Xtra

Some extra renderers. The path is Renderers → Xtra. See Tables 8-170 to 8-172.

Table 8-170.

| | | |
|----------------------|--|--|
| gravity_lines | | Lets 40 masses revolve around a center point. Applies a non-Newton law of gravity: $F = -q \cdot (\text{pos} - \text{center})$. Note that the algorithm will collapse towards the gravitational center, so the latter (the pos anchor) should be changing during the visualization. Renders the masses' trajectories as lines |
| In | pos:float3 params:complex friction:float step_length: float color0:float4 color1:float4 | Gravitational center A friction to apply to the movements The advection per frame. The masses will be faster once you increase this parameter The starting color The finishing color |
| Out | render_out:render | The renderer output |

Table 8-171.

| | | |
|-----------------------|---|---|
| gravity_ribbon | | Lets a ribbon revolve around a center point. Applies a non-Newton law of gravity: $F = -q \cdot (\text{pos} - \text{center})$. Note that the algorithm will collapse towards the gravitational center, so the latter (the pos anchor) should be changing during the visualization The Mesh → Generators → Ribbon gives usually better results than this module |
| In | pos:float3 params:complex length:float ribbon_width: float friction:float step_length: float color0:float4 color1:float4 | Gravitational center The length of the ribbon The width of the ribbon A friction to apply to the movements The advection per frame. The ribbon will be faster once you increase this parameter The starting color The finishing color |
| Out | render_out: render | The renderer output |

Table 8-172.

| | | |
|---------------|--------------------|---|
| skybox | | A special module tailored for a skybox. Will produce six bitmaps from a single input bitmap for mapping on a cube. You probably need to connect it to a Texture → opengl → bitmcubemap module and afterwards apply shader code and connect the texture to a samplerCube type shader variable. See the state in Examples → skybox_chromatic |
| In | bitmap:bitmap | Input bitmap |
| Out | render_out:render | A renderer output. You don't have to use this |
| | bitmaps:complex | The six bitmaps for the mapping on a cube |
| | positive_x: bitmap | |
| | negative_x: bitmap | |
| | positive_y: bitmap | |
| | negative_y: bitmap | |
| | positive_z: bitmap | |
| | negative_z: bitmap | |

Selectors

Selectors allow for the programmatic selection of values from a range of values given an index. The path is Selectors, then see Tables 8-173 to 8-178.

Table 8-173.

| | | |
|------------------------|------------------|--|
| float3_selector | | Selects from the float3 vectors given by the anchors float3_* |
| In | index:float | Which of the float3 values to select |
| | inputs:enum | How many float3 values will be provided. One of 0, 1, 2, ..., 16 |
| | float3_x:complex | Inside this complex anchor the input float3 values will have to be specified |
| | float3_0:float3 | First value |
| | float3_1:float3 | Second value |
| | ... | More values |

(continued)

Table 8-173. (continued)

| float3_selector | Selects from the float3 vectors given by the anchors float3_* |
|---------------------------|---|
| options:complex | |
| wrap:enum | <p>What to do if the index is out of bounds:</p> <ul style="list-style-type: none"> - Wrap: Wrap around the index. If for example “inputs” = 3 and no interpolation, the values chosen are: -1 → val3, 0 → val1, 1 → val2, 2 → val3, 3 → val0, ... - None_zero: no warp, set off-bound indices to 0.0 - None_freeze: no warp, set to closest valid value |
| interpolation: enum | <p>One of:</p> <ul style="list-style-type: none"> - None: no interpolation. The mapping from input “index” to the selected index is: Round(inp) - Linear: The mapping from input “index” to the selected index is: Floor(inp). Fractional parts get interpolated linearly - Sequence: Same as linear, but uses the sequence given for interpolating |
| sequence: sequence | Only if interpolation is set to sequence, use this to interpolate the fractional part of the input index. |
| reverse:enum | <p>Only if interpolation is set to sequence:</p> <ul style="list-style-type: none"> - Off: No reversal - On: Interpret sequence from right to left instead. - Auto_Normal: Automatically reverse the sequence when values are decreasing - Auto_Inverted: Like Auto_Normal, but reversed once more. |
| reset_seq_to_default:enum | A trigger. If ok is chosen, reset the sequence to its default cosine shape. |
| Out result:float3 | The chosen and possibly interpolated value |

Table 8-174.

| float4_selector | Selects from the float4 vectors given by the anchors from inside float4_x |
|---------------------------|--|
| In | |
| index:float | Which of the float4 values to select. |
| inputs:enum | How many float4 values will be provided. One of 0, 1, 2, ..., 16 |
| float3_x:complex | Inside this complex anchor the input float4 values will have to be specified |
| float4_0:float4 | First value |
| float4_1:float4 | Second value |
| ... | More values |
| options:complex | |
| wrap:enum | What to do if the index is out of bounds: <ul style="list-style-type: none"> - Wrap: Wrap around the index. If for example "inputs" = 3 and no interpolation, the values chosen are: -1 → val3, 0 → val1, 1 → val2, 2 → val3, 3 → val0, ... - None_zero: no warp, set off-bound indices to 0.0 - None_freeze: no wrap, set to closest valid value |
| interpolation:enum | One of: <ul style="list-style-type: none"> - None: no interpolation. The mapping from input index to the selected index is: Round(inp) - Linear: The mapping from input index to the selected index is Floor(inp). Fractional parts get interpolated linearly - Sequence: Same as linear, but uses the sequence given for interpolating |
| sequence:sequence | Only if interpolation is set to sequence, use this to interpolate the fractional part of the input index. |
| reverse:enum | Only if interpolation is set to sequence: <ul style="list-style-type: none"> - Off: No reversal - On: Interpret sequence from right to left instead. - Auto_Normal: Automatically reverse the sequence when values are decreasing - Auto_Inverted: Like Auto_Normal, but reversed once more. |
| reset_seq_to_default:enum | A trigger. If ok is chosen, reset the sequence to its default cosine shape. |
| Out | |
| result:float4 | The chosen and possibly calculated value |

Table 8-175.

| float_selector | Selects from the float values given by the anchors float_* |
|-----------------------|--|
| In | |
| index:float | Which of the float values to select. |
| inputs:enum | How many float values will be provided. One of 0, 1, 2, ..., 16 |
| float3_x:complex | Inside this complex anchor the input float values will have to be specified |
| float_0:float | First value |
| float_1:float | Second value |
| ... | More values |
| options:complex | |
| wrap:enum | What to do if the index is out of bounds: <ul style="list-style-type: none"> - Wrap: Wrap around the index. If for example "inputs" = 3 and no interpolation, the values chosen are: -1 → val3, 0 → val1, 1 → val2, 2 → val3, 3 → val0, ... - None_zero: no warp, set off-bound indices to 0.0 - None_freeze: no wrap, set to closest valid value |
| interpolation: enum | One of: <ul style="list-style-type: none"> - None: no interpolation. The mapping from input "index" to the selected index is: Round(inp) - Linear: The mapping from input "index" to the selected index is: Floor(inp). Fractional parts get interpolated linearly - Sequence: Same as linear, but uses the sequence given for interpolating |
| sequence:sequence | Only if interpolation is set to sequence, use this to interpolate the fractional part of the input index. |

(continued)

Table 8-175. (continued)

| float_selector | | Selects from the float values given by the anchors float_* |
|-----------------------|---------------------------|--|
| | reverse:enum | Only if interpolation is set to sequence: <ul style="list-style-type: none"> - Off: No reversal - On: Interpret sequence from right to left instead. - Auto_Normal: Automatically reverse the sequence when values are decreasing - Auto_Inverted: Like Auto_Normal, but reversed once more. |
| | reset_seq_to_default:enum | A trigger. If ok is chosen, reset the sequence to its default cosine shape. |
| Out | result:float | The chosen and possibly interpolated value |

Table 8-176.

| quaternion_selector | | Selects from the quaternion values given by the anchors from inside quaternion_x |
|----------------------------|--------------------------|---|
| In | index:float | Which of the quaternion values to select |
| | inputs:enum | How many quaternion values will be provided. One of 0, 1, 2, ..., 16 |
| | quaternion_x: complex | Inside this complex anchor the input quaternion values will have to be specified |
| | quaternion_0: quaternion | First value |
| | quaternion_1: quaternion | Second value |
| | ... | More values |

(continued)

Table 8-176. (continued)

| quaternion_selector | Selects from the quaternion values given by the anchors from inside quaternion_x |
|------------------------------|--|
| options:complex wrap:enum | What to do if the index is out of bounds: <ul style="list-style-type: none"> - Wrap: Wrap around the index. If for example “inputs” = 3 and no interpolation, the values chosen are: -1 → val3, 0 → val1, 1 → val2, 2 → val3, 3 → val0, ... - None_zero: no warp, set off-bound indices to 0.0 - None_freeze: no wrap, set to closest valid value |
| interpolation: enum | One of: <ul style="list-style-type: none"> - None: no interpolation. The mapping from input index to the selected index is Round(inp) - Linear: The mapping from input index to the selected index is Floor(inp). Fractional parts get interpolated linearly - Sequence: Same as linear, but uses the sequence given for interpolating |
| sequence: sequence | Only if interpolation is set to sequence, use this to interpolate the fractional part of the input index. |
| reverse:enum | Only if interpolation is set to sequence : <ul style="list-style-type: none"> - Off: No reversal - On: Interpret sequence from right to left instead. - Auto_Normal: Automatically reverse the sequence when values are decreasing - Auto_Inverted: Like Auto_Normal, but reversed once more. |
| reset_seq_to_default:enum | A trigger. If ok is chosen, reset the sequence to its default cosine shape. |
| Out result:quaternion | The chosen and possibly interpolated value |

Table 8-177.

| string_selector | | Selects from the string values given by the anchors from inside string_x |
|------------------------|------------------|--|
| In | index:float | Which of the string values to select. |
| | inputs:enum | How many string values will be provided. One of 0, 1, 2, ..., 16 |
| | string_x:complex | Inside this complex anchor the input string values will have to be specified |
| | string_0:string | First value |
| | string_1:string | Second value |
| | ... | More values |
| | wrap:enum | What to do if the index is out of bounds: <ul style="list-style-type: none"> - Wrap: Wrap around the index. If for example “inputs” = 3 and no interpolation, the values chosen are: -1 → val3, 0 → val1, 1 → val2, 2 → val3, 3 → val0, ... - None_zero: no warp, set off-bound indices to “0.0” - None_freeze: no wrap, set to closest valid value |
| Out | result:string | The chosen value |

Table 8-178.

| texture_selector | | Selects from the textures given by the anchors from inside texture_x |
|-------------------------|-------------------|---|
| In | index:float | Which of the textures to select. |
| | inputs:enum | How many textures will be provided. One of 0, 1, 2, ..., 16 |
| | texture_x:complex | Inside this complex anchor the input textures will have to be specified |
| | texture_0:texture | First texture |
| | texture_1:texture | Second texture |
| | ... | More textures |

(continued)

Table 8-178. (continued)

| texture_selector | Selects from the textures given by the anchors from inside texture_x |
|-------------------------|---|
| options: complex | |
| wrap: enum | <p>What to do if the index is out of bounds:</p> <ul style="list-style-type: none"> - Wrap: Wrap around the index. If for example “inputs” = 3 and no interpolation, the textures chosen are: -1 → val3, 0 → val1, 1 → val2, 2 → val3, 3 → val0, ... - None_zero: no warp, set off-bound indices to empty - None_freeze: no wrap, set to closest valid value |
| blend_type: enum | <p>One of:</p> <ul style="list-style-type: none"> - Snap: switch abruptly between textures when the index changes - Linear: Blend linearly between adjacent textures when the index changes. - Sequence: Blend according to the sequence anchor |
| blend_options: complex | |
| blend_size: enum | <p>While blending an own texture is created. This is its size. One of 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048</p> |
| A_crossfade: complex | <p>Defines cross-fade options for texture A (floor of index)</p> |
| A_sequence: sequence | <p>The sequence for the texture A while blending, modulates the sequence parameter.</p> |
| A_reverse: enum | <p>Off or on, whether to reverse that sequence</p> |
| A_reset_seq_to_default: | <p>Trigger. If ok, reset that A cross-fade sequence to linear mode.</p> |

(continued)

Table 8-178. (continued)

| texture_selector | Selects from the textures given by the anchors from inside texture_x |
|------------------------------|---|
| enum | |
| B_cross-fade: complex | |
| B_sequence: sequence | The sequence for the texture B while blending, modulates the sequence parameter. |
| B_reverse: enum | Off or on, whether to reverse that sequence. |
| B_reset_seq_to_default: enum | Trigger. If ok, reset that B cross-fade sequence to linear mode. |
| sequence: sequence | Only if blend_type is set to sequence, blend while the index changes from one to the next sequence. May further be modulated by the sequences inside blend_options. |
| reverse:enum | Off or on. If 0n, and only if blend_type is set to sequence, reverse the sequence. |
| reset_seq_to_default:enum | A trigger, if set to ok, reset the sequence to its default = linearly. |
| shaders:complex | For further fine-tuning the blending between textures |
| vertex_program: string | The vertex shader |
| fragment_program: string | The fragment shader |
| shad_param1:float | Shader params (uniforms) |
| shad_param2:float | |
| shad_param3:float | |
| shad_param4:float | |
| shad_param5:float | |
| shad_param6:float | |
| shad_param7:float | |
| shad_param8:float | |
| Out result:texture | The chosen and possibly interpolated texture |

Sound

Sound modules build the interfacing to incoming sound.

input_visualization_listener

Listens to incoming sound and provide numerical output for other modules. This is the main input module if you want to react on audio input and you'd usually place this module graphically on the very left if using the GUI. The path is Sound → input_visualization_listener.

Technically it will be using a *fast Fourier transformation* (FFT) algorithm to transform the time-based input delivered by the system's sound device into the frequency domain, so you will be able to tell the frequency distribution and the volume of each part of the frequency spectrum. See Table 8-179.

Table 8-179.

| input_visualization_listener | | Listens to incoming sound |
|-------------------------------------|------------------|---|
| In | multiplier:float | Multiplies what comes from the system's audio device by some number. Default is 1.0 |
| Out | vu:complex | The current overall amplitude, determined by summing up the coefficients of the FFT outcome. A complex of two numbers, left and right stereo channel. |
| | vu_l:float | Left stereo channel. A float number between 0.0 and 1.0 |
| | vu_r:float | Right stereo channel. A float number between 0.0 and 1.0. Note that vu_l and vu_r are currently the same, because the FFT only runs over the left channel and the value from the left channel just gets copied over to the right channel. |
| | octaves:complex | Amplitudes on a coarse octave based frequency spectrum. |
| | left:complex | Left stereo channel. |
| | octaves_l_0 | Lowest octave, approximately from F to f (midi tones 41 until 53) |
| | octaves_l_1 | This is where the middle c (c' midi tone 60) will be inside. Approximately from f to f' (midi tones 53 until 65) |
| | octaves_l_2 | Approximately from f' to f'' (midi tones 65 until 77) |

(continued)

Table 8-179. (continued)

| <code>input_visualization_listener</code> | Listens to incoming sound |
|---|---|
| <code>octaves_l_3</code> | Approximately from f'' to f''' (midi tones 77 until 89) |
| <code>octaves_l_4</code> | Approximately from f'''' to f''''' (midi tones 89 until 101) |
| <code>octaves_l_5</code> | Approximately from f'''''' to f''''''' (midi tones 101 until 113) |
| <code>octaves_l_6</code> | Approximately from $f^{(5)}$ to $f^{(6)}$ (midi tones 113 until 125) |
| <code>octaves_l_7</code> | Highest octave from $f^{(6)}$ to $f^{(7)}$ |
| <code>right:complex</code> | Right stereo channel. Note that the right channel and left channel show the same numbers, since the FFT runs only over the left channel and its values are just copied to the right channel array. |
| <code>octaves_r_0</code> | Lowest octave, see description for <code>octaves_l_0</code> |
| <code>octaves_r_1</code> | See description for <code>octaves_l_1</code> |
| <code>octaves_r_2</code> | See description for <code>octaves_l_2</code> |
| <code>octaves_r_3</code> | See description for <code>octaves_l_3</code> |
| <code>octaves_r_4</code> | See description for <code>octaves_l_4</code> |
| <code>octaves_r_5</code> | See description for <code>octaves_l_5</code> |
| <code>octaves_r_6</code> | See description for <code>octaves_l_6</code> |
| <code>octaves_r_7</code> | Highest octave, see description for <code>octaves_l_7</code> |
| <code>wave:float_array</code> | The audio sample itself. An array of size 512 containing the sample data for the left channel. The right channel is neglected. |
| <code>spectrum: float_array</code> | The spectrum. An array of size 512 containing the amplitudes in the frequency space. Originally each value's index ($i=0..511$) from the FFT outcome corresponds to a frequency $i / 256 * 44100 / 2$, but we want to have equal tune ranges and thus re-index accordingly. The lowest value thus goes for 86 Hz (= F, midi 41) and the highest for $f^{(7)}$ (off MIDI range). This is eight octaves or 96 half tones, thus each value in this array is for a $96 / 512 = 0.1875$ half tones or 18.75 cents. Note that the lower frequencies are quite coarse due to the nature of the FFT algorithm. |

You might ask why we don't have lower octaves as well. This is intrinsic to the algorithm; for lower octaves we'd need a larger audio calculation buffer, more time for a sample slowing the reactivity, and more CPU power.

midi → aka_apc40_controller

This is a certain kind of hardware controller for professionally using ThMAD in realtime performances. The path is Sound ath:is → aka_apc40_controller. See Table 8-180.

Table 8-180.

| | |
|------------------------------------|--|
| midi → aka_apc40_controller | Connects to an Akai APC40 midi controller |
|------------------------------------|--|

Consult the controller manual for parameters. Anchors are named accordingly.

ogg_sample_*

Ogg Vorbis sound file format players. The path is Sound → ogg_sample_. See Tables 8-181 and 8-182.

Table 8-181.

| | | |
|------------------------|---|-----------------------------|
| ogg_sample_play | Plays an audio file in Ogg Vorbis format. You should be able to hear it, and also data goes to the input_visualization_listener. Start immediate playback when configured or ThMAD starts. If you need a triggered playback, use the module ogg_sample_trigger | |
| In | filename: resource | The OGG Vorbis file to play |
| | format:enum | Mono or stereo |

Table 8-182.

| | | |
|---------------------------|--|--|
| ogg_sample_trigger | Plays an audio file in Ogg Vorbis format. There is a trigger, and you can apply a gain and a pitch modifier | |
| In | filename: resource | The OGG Vorbis file to play |
| | trigger:float | Start play when 1.0 or above. Stop playing when 0.0 or below. While playing, defines the speed (>1 is faster, <1 slower) |
| | pitch:float | Another way to change playing speed (and pitch). Set 0 to keep unchanged (still "trigger" ≠ 1 will change the speed and pitch) |
| | gain:float | A volume gain control |

raw_sample_*

Raw means a header-less sound file, i.e., all bytes are uncompressed PCM sound data. The format needed is signed int 16-bit little endian.

You can create RAW files for example with the software `ffmpeg`. As an example, if you have a WAV file, you can convert it to a suitable RAW file by entering this in the terminal:

```
ffmpeg -i a.wav -f s16le -acodec
pcm_s16le a.raw
```

The path is Sound → raw_sample_. See Tables 8-183 and 8-184.

Table 8-183.

| | | |
|------------------------|-------------------|---|
| raw_sample_play | | Plays an audio file in raw format. You should be able to hear it, and also data goes to the input_visualization_listener. Start immediate playback when configured or ThMAD starts. If you need a triggered playback, use the module raw_sample_trigger. |
| In | filename:resource | The RAW file to play |
| | format:enum | Mono or stereo |

Table 8-184.

| | | |
|---------------------------|-------------------|--|
| raw_sample_trigger | | Plays an audio file in raw format. There is a trigger, and you can apply a gain and a pitch modifier |
| In | filename:resource | The RAW file to play. |
| | trigger:float | Start play when 1.0 or above. Stop playing when 0.0 or below. While playing, defines the speed (>1 is faster, <1 slower) |
| | pitch:float | Another way to change playing speed (and pitch). Set 0 to keep unchanged (still “trigger” ≠ 1 will change the speed and pitch) |
| | gain:float | A volume gain control |

Strings

String related modules. The path is String, then see Tables [8-185](#) and [8-186](#).

Table 8-185.

| float_to_string | | Converts a float to a string |
|------------------------|-----------------------------------|---|
| In | float_in:float precision:float | Float input The number of digits to show after the decimal separator |
| Out | string_out:string | The string representation |

Table 8-186.

| res_to_str | | Use this to get the path from a resource. Will output something like resources/my.png |
|-------------------|----------------------|--|
| In | resource_in:resource | The resource |
| Out | string_out:string | The resource path |

System

System level modules. The path is System, then see Tables [8-187](#) to [8-198](#).

Table 8-187.

| blocker | | Use this to conditionally prevent a rendering sub-pipeline from being run |
|----------------|----------------------------------|--|
| In | render_in: render block:float | The renderer input If this is less than 0.5, block the sub-pipeline. If equal or greater than 0.5, let it run |
| Out | render_out: render | The renderer output |

Table 8-188.

| blocker_loading | | Renders a sub-pipeline only when ThMAD is loading |
|------------------------|--|---|
| In | render_in: render fadeout_time: float | The renderer input Do not block for that many seconds after ThMAD's loading started. Then block. |
| Out | render_out: render fadeout_out: float | The renderer output Will linearly decrease from 1.0 to 0.0. When 0.0 is reach, start the blocking. |

Table 8-189.

| clock | | Allows access to the system clock |
|--------------|---------------------|--|
| Out | clock:complex | |
| | year:float | Year minus 1900 |
| | month:float | Month, 0 = January |
| | dayofweek: float | Day of week, 0 to 6, 0 = Sunday |
| | day:float | Day in month, 1, ..., 31 |
| | hour:float | Hour of day, integer part = 0, ..., 23. Contains fraction of hour! |
| | hour12:float | Hour of day, integer part = 0, ..., 11. Contains fraction of hour! |
| | minute:float | Minute of hour. Contains fraction of minute! |
| | second:float | Second of minute. Contains fraction of second! |
| | millisecond: float | Millisecond of second. Contains fraction of millisecond! |

Table 8-190.

| filesystem → file_chooser | | Allows a dynamic selection of resources given a path |
|----------------------------------|---------------------------|--|
| In | directory_path: string | Which directory to choose inside /home/[USER]/thmad/ [VERSION]/data All files including those in sub folders inside will be taken into account! |
| | file_id:float | Which file to choose from all the files found |
| Out | filename_result: resource | The chosen resource |
| | filename_count: float | The number of files actually found |

Table 8-191.

| joystick | | Accesses joystick input from system devices /dev/js* |
|-----------------|---------------------|--|
| Out | joystick_0: complex | First joystick (only if connected, otherwise the anchor doesn't exist) |
| | j_0_name: string | The name of the joystick if the system can determine it |
| | axes:complex | |
| | j_0_axis0: float | First axis |
| | j_0_axis1: float | Second axis (if it exists) |
| | ... | Possibly more axes |
| | buttons: complex | |
| | j_0_button0: float | First button state |
| | j_0_button1: float | Second button state (if it exists) |
| | ... | Possibly more buttons |
| | joystick_1: complex | Second joystick (only if connected, otherwise the anchor doesn't exist). Sub-anchors similar to the ones for joystick_0 |
| | ... | More joysticks (dynamically detected) |

Table 8-192.

| | | |
|-----------------|----------------|--|
| shutdown | | Programatically shuts down ThMAD if the input goes above 1.0. Note that it does not take care of unsaved changes in your state if used inside Artiste |
| In | shutdown:float | Performs an unconditional shutdown of ThMAD (either Artist or Player) when this value reaches or raises above 1.0 |

Table 8-193.

| | | |
|---------------------|--------------------|--|
| state_loader | | Loads a *.vsx file into the current state |
| In | filename: resource | The *.vsx file. If using state_loader, ThMAD considers the file a resource, so it must lie inside the /home/[USER]/thmad/ [VERSION]/data/resources folder. |
| Out | render_out: render | The output from this loaded state. |

Table 8-194.

| | | |
|---------------------------------|-----------------|--|
| system_sequencer_control | | Controls the sequencing time, which is an alternative timing concept to the system clock. Inside the GUI, place it anywhere to use it. The module has no output, but controls the internal state of the engine |
| In | trig_play:float | If not already playing, changing its value from 0.0 or below to a positive value triggers a PLAY event. If already playing, do nothing. If event triggered, starts the sequencing timer (i.e., one second in the real world will show up as a second in the sequencing timer). If event triggered, and trig_set_time is 0 or smaller, the sequencing timer will start at its last value. Otherwise, it starts with the value given by trig_set_time. |
| | trig_stop:float | If not already stopped, changing its value from 0.0 or below to a positive value triggers a STOP event. The sequencing time stops at its latest value and doesn't advance any longer. It can be restarted later by using the trig_play trigger. |

(continued)

Table 8-194. (continued)

| | |
|---------------------------------|---|
| system_sequencer_control | Controls the <i>sequencing time</i>, which is an alternative timing concept to the system clock. Inside the GUI, place it anywhere to use it. The module has no output, but controls the internal state of the engine |
| trig_rewind:float | If currently running, changing its value from 0.0 or below to a positive value triggers a REWIND event. If trig_set_time is 0 or smaller, change the sequence timer to 0.0. If trig_set_time is greater than zero, change the sequence timer to trig_set_time. In both cases stops the engine (no more sequencing time advancement) |
| trig_set_time: float | Controls the value when rewinding or restarting the engine. |

Table 8-195.

| | |
|--------------------|--|
| time | Tells the operating system time and the sequencing time |
| Out normal:complex | Sequencing timer. |
| time:float | The sequencing time |
| dtime:float | The time difference between this and the previous frame, using the sequencing timer. |
| real:complex | Operating system timer |
| r_time:float | The operating system time |
| r_dtime:float | The time difference between this and the previous frame, using the operating system timer. |

Table 8-196.

| to_console | | Write values periodically to the console. Only applies if Artiste or Player was started from a console |
|-------------------|---------------------------|--|
| In | out_id:string | If set, prepend this string to the console output |
| | enabled:enum | One of no or yes |
| | show_each:float | Output frequency in seconds. If set to 1.0, output each frame which will slow down ThMAD. Maybe set to 60 or something like that |
| | params:complex | |
| | show_float: enum | One of no or yes. Whether to print the value |
| | in_float:float | Input float |
| | show_float3: enum | One of no or yes. Whether to print the value |
| | in_float3:float3 | Input float3 |
| | show_float4: enum | One of no or yes. Whether to print the value |
| | in_float4:float4 | Input float4 |
| | show_quaternion: enum | One of no or yes. Whether to print the value |
| | in_quaternion: quaternion | Input quaternion |

Table 8-197.

| viewport_size | | Use this to determine the actual graphical output size in pixels. You don't normally use that, because most coordinates are relative |
|----------------------|----------|---|
| Out | vx:float | Width in pixels |
| | vy:float | Height in pixels |

Table 8-198.

| visual_fader | | A fader module exclusively used for visuals transition inside the Player. Use <code>New → Transition For ThMAD Player</code> to see how it is used |
|---------------------|--------------------------------------|---|
| In | <code>texture_a_in: texture</code> | Do not use |
| | <code>texture_b_in: texture</code> | Do not use |
| | <code>fade_pos_in: float</code> | Do not use |
| | <code>options:complex</code> | |
| | <code>transition_length:float</code> | How many seconds the transition will take |
| Out | <code>texture_a_out: texture</code> | The faded-out texture, i.e., the faded-out visual |
| | <code>texture_b_out: texture</code> | The faded-in texture, i.e., the faded-in visual |
| | <code>fade_pos_out: float</code> | Inside [0.0;1.0], where in the transition phase we are |

Texture

Textures are images living in your graphics hardware. They are very important for realtime graphics processing.

Buffers

Texture buffers allow for sending rendered pixel data to a buffer on the graphics hardware for fast rendering. The path is `Texture → Buffers`. See Tables [8-199](#) to [8-202](#).

Table 8-199.

| | |
|--|---|
| <code>render_buffer</code> | <p>Represents a texture. Renders its input sub-pipeline and stores the result in a texture. The data will be kept there, even when the rendering input is deactivated or even removed.</p> <p>The texture doesn't use mipmaps, uses "nearest" filtering for both magnification and minification (texture pixel bigger or smaller than color buffer pixel) and clamps both coordinates to the edge for wrapping</p> |
| <p>In <code>render_in:render</code></p> <p><code>options:complex</code></p> <p style="padding-left: 20px;"><code>texture_size: enum</code></p> <p style="padding-left: 40px;"><code>size_x:float</code></p> <p style="padding-left: 40px;"><code>size_y:float</code></p> <p style="padding-left: 40px;"><code>float_texture: enum</code></p> <p style="padding-left: 40px;"><code>alpha_channel: enum</code></p> <p style="padding-left: 40px;"><code>multisample: enum</code></p> | <p>The graphics data we want to save as a texture on the graphics hardware</p> <p>The size. One of:</p> <ul style="list-style-type: none"> - 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 - VIEWPORT_SIZE: the viewport's size - VIEWPORT_SIZE_DIV_2: half the viewport's size - VIEWPORT_SIZE_DIV_4: fourth of the viewport's size - VIEWPORT_SIZE_x2: twice the viewport's size - VIEWPORT_SIZE_x4: four times viewport's size - CUSTOM_SIZE: Size as given by the "size_x" and "size_y" anchors below <p>Only if texture_size is set to CUSTOM_SIZE, use this size</p> <p>One of no or yes. If yes, use float textures instead. Especially interesting for shader code.</p> <p>One of no or yes. If yes, enable the ALPHA channel inside the texture.</p> <p>One of no or yes. If yes, enable multisampling, improving the smoothness of pixel color transitions at edges and corners.</p> |
| <p>Out <code>texture_out:texture</code></p> | <p>The output texture, i.e., the pointer to a data buffer on the graphics hardware</p> |

Table 8-200.

| render_surface_color_buffer | | A color buffer, similar to module render_buffer but without depth buffer |
|------------------------------------|-----------------------|--|
| In | render_in:render | The graphics data we want to save as a texture on the graphics hardware |
| | options:complex | |
| | texture_size: enum | The size. One of: <ul style="list-style-type: none"> - 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 - VIEWPORT_SIZE: the viewport's size - VIEWPORT_SIZE_DIV_2: half the viewport's size - VIEWPORT_SIZE_DIV_4: fourth of the viewport's size - VIEWPORT_SIZE_x2: twice the viewport's size - VIEWPORT_SIZE_x4: four times viewport's size - CUSTOM_SIZE: Size as given by the size_x and size_y anchors |
| | size_x:float | Only if texture_size is set to CUSTOM_SIZE, use this size |
| | size_y:float | |
| | float_texture: enum | One of no or yes. If yes, use float textures instead. Especially interesting for shader code. |
| | alpha_channel: enum | One of no or yes. If yes, enable the ALPHA channel inside the texture. |
| Out | color_buffer: texture | The output texture, i.e., the pointer to a data buffer on the graphics hardware |

Table 8-201.

| render_surface_color_depth_ | | A buffer that can use a separate shared depth depth buffer and also outputs the depth buffer separately |
|------------------------------------|------------------------------------|--|
| In | <code>render_in:render</code> | The graphics data we want to save as a texture on the graphics hardware |
| | <code>depth_buffer: texture</code> | If connected, the depth buffer to use. Must have the same size as specified by the size parameter in this module |
| | <code>options:complex</code> | |
| | <code>texture_size: enum</code> | The size. One of: <ul style="list-style-type: none"> - 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 - VIEWPORT_SIZE: The viewport's size - VIEWPORT_SIZE_DIV_2: Half the viewport's size - VIEWPORT_SIZE_DIV_4: Fourth of the viewport's size - VIEWPORT_SIZE_x2: Twice the viewport's size - VIEWPORT_SIZE_x4: Four times the viewport's size - CUSTOM_SIZE: Size as given by the size_x and size_y anchors |
| | <code>size_x:float</code> | Only if texture_size is set to CUSTOM_SIZE, use this size |
| | <code>size_y:float</code> | |
| | <code>float_texture: enum</code> | One of no or yes. If yes, use float textures instead. Especially interesting for shader code. |
| | <code>alpha_channel: enum</code> | One of no or yes. If yes, enable the ALPHA channel inside the texture. |
| Out | <code>color_buffer: texture</code> | The output texture, color buffer only |
| | <code>depth_buffer: texture</code> | The output depth buffer. Can be shared with other render_surface_color_depth_buffer modules. |

Table 8-202.

| | |
|--------------------------------------|--|
| <code>render_surface_single</code> | A buffer with color and depth data. Allows for backfeeding, i.e., rendering later in the sub-pipeline may be fed back to the input of this module. The texture doesn't use mipmaps, uses "linear" filtering for both magnification and minification (texture pixel bigger or smaller than color buffer pixel) and clamps both coordinates to the edge for wrapping |
| In <code>render_in:render</code> | The graphics data we want to save as a texture on the graphics hardware |
| <code>options:complex</code> | |
| <code>texture_size: enum</code> | The size. One of: <ul style="list-style-type: none"> - 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 - <code>VIEWPORT_SIZE</code>: the viewport's size - <code>VIEWPORT_SIZE_DIV_2</code>: half the viewport's size - <code>VIEWPORT_SIZE_DIV_4</code>: Fourth of the viewport's size - <code>VIEWPORT_SIZE_x2</code>: Twice the viewport's size - <code>VIEWPORT_SIZE_x4</code>: Four times viewport's size - <code>CUSTOM_SIZE</code>: Size as given by the <code>size_x</code> and <code>size_y</code> anchors |
| <code>size_x:float</code> | Only if <code>texture_size</code> is set to <code>CUSTOM_SIZE</code> , use this size |
| <code>size_y:float</code> | Only if <code>texture_size</code> is set to <code>CUSTOM_SIZE</code> , use this size |
| <code>support_feedback: enum</code> | One of no or yes. Only if enabled, feedback is possible. |
| <code>float_texture:enum</code> | One of no or yes. If yes, use float textures instead. Especially interesting for shader code. |
| <code>alpha_channel: enum</code> | One of no or yes. If yes, enable the ALPHA channel inside the texture. |
| <code>clear_color:float4</code> | A clearing color used |
| Out <code>texture_out:texture</code> | The output texture, i.e., the pointer to a data buffer on the graphics hardware |

Dummies

Dummy modules. The path is Texture → Dummies. See Table 8-203.

Table 8-203.

| texture_dummy | | A dummy; just passes the texture through. Useful for example inside macros |
|----------------------|----------------------|---|
| In | texture_in: texture | Input texture |
| Out | texture_out: texture | Output texture |

Effects

Texture effects. The path is Texture → Effects. See Tables 8-204 and 8-205.

Table 8-204.

| blur | | A blurring. Uses an internal shader to blur the input texture. If you want to achieve a glow, you might want to blend an unblurred and a blurred version of the same image |
|-------------|----------------------|---|
| In | glow_source: texture | Input texture. |
| | start_value: float | The intensity of the effect. The higher, the more blurring. |
| | attenuation: float | An attenuation factor applied just before the output happens. |
| | texture_size: enum | The size. One of: <ul style="list-style-type: none"> – 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 – VIEWPORT_SIZE: The viewport's size – VIEWPORT_SIZE_DIV_2: Half the viewport's size – VIEWPORT_SIZE_DIV_4: Fourth of the viewport's size – VIEWPORT_SIZE_x2: Twice the viewport's size |
| | passes: enum | One of ONE or TWO. Applying two passes greatly enhances the effect. |
| Out | texture_out: texture | Output texture |

Table 8-205.

| highblur | | An elaborated blurring effect. Uses shading code and only works if used inside a backfeeding sub-pipeline |
|-----------------|-----------------------|---|
| In | texture_in: texture | The input texture to be blurred |
| | translation:float | Translational part of the effect |
| | blowup_center: float3 | The center for the scaling (blowup) part of the effect |
| | blowup_rate: float | The intensity of the scaling (blowup) part of the effect. The higher, the less the effect. |
| | texture_size: enum | The size. One of: <ul style="list-style-type: none"> - 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 - VIEWPORT_SIZE: The viewport's size - VIEWPORT_SIZE_DIV_2: Half the viewport's size - VIEWPORT_SIZE_DIV_4: Fourth of the viewport's size - VIEWPORT_SIZE_x2: Twice the viewport's size |
| Out | texture_out: texture | The output texture |

Loaders

Receiving and sending texture data. The path is Texture → Loaders.

See Tables 8-206 to 8-211.

Table 8-206.

| bitmap2texture | | Converts a bitmap to a texture. Implies the bitmap is uploaded to the graphics hardware as a texture |
|-----------------------|-----------------|--|
| In | bitmap:bitmap | The input bitmap. |
| | mipmaps:enum | One of no or yes. Whether to use mipmaps, meaning the hardware will maintain downsized versions of the texture data for faster rendering in case height resolutions are unnecessary. |
| Out | texture:texture | Output texture |

Table 8-207.

| jpeg_tex_load | | Loads a JPEG and send pixel data to the graphics hardware as a texture |
|----------------------|--------------------|---|
| In | filename: resource | The JPEG file to load. Note that JPEGs don't have an ALPHA channel |
| Out | texture:texture | The pointer to the texture |
| | bitmap:bitmap | In case you need it, the module also provides the image as a bitmap |

Table 8-208.

| jpeg_tex_load_alpha | | JPEGs don't have an ALPHA channel. But you can use this module and provide a second JPEG of the same size, with the gray value of it mapped to the ALPHA of the output image |
|----------------------------|--------------------------|---|
| In | filename_rgb: resource | The RGB data |
| | filename_alpha: resource | The ALPHA data |
| Out | texture:texture | The pointer to the texture |
| | bitmap:bitmap | In case you need it, the module also provides the image as a bitmap |

Table 8-209.

| | |
|------------------|---|
| png_cubemap_load | Under development and currently not functional. |
|------------------|---|

Table 8-210.

| png_tex_load | | Loads a PNG file and sends it as a texture to the graphics hardware |
|---------------------|-------------------|--|
| In | filename:resource | The PNG file. Must lie inside inside the /home/[USER]/thmad/ [VERSION]/data/resources folder |
| | reload:enum | A trigger. If yes is selected, once reload the data from the file |
| Out | texture:texture | The pointer to the texture |
| | bitmap:bitmap | If you need also the bitmap, it is here |

Table 8-211.

| texture2bitmap | | Loads a texture from the graphics hardware and provides access to the data as a bitmap |
|-----------------------|---------------------|---|
| In | texture_in: texture | The input texture |
| Out | bitmap:bitmap | The bitmap |

Modifiers

These operations happen directly on the graphics hardware, hence they are fast operations. The path is Texture → modifiers. See Tables 8-212 to 8-216.

Table 8-212.

| rotate | | Rotates a texture. |
|---------------|----------------------------|--|
| In | texture_in: texture | The input texture |
| | rotation_angle: float | The rotation angle |
| | rotation_axis: float3 | The rotation axis |
| | center:complex | |
| | use_rotate_center:enum | yes or no, whether a rotation center is specified (otherwise the rotation happens around (0;0) in texture coordinates, which is probably not what you want). |
| | rotate_center: float3 | The rotation center. The point (0.5;0.5) for example lies in the middle. The third coordinate is ignored. |
| Out | texture_rotate_out:texture | The rotated texture |

Table 8-213.

| scale | | Scales a texture |
|--------------|----------------------------|---|
| In | texture_in: texture | The input texture |
| | scale_vector: float3 | The scaling for all coordinates. The third coordinate is ignored. |
| | center: complex | |
| | use_scale_center: enum | yes or no, whether a scaling center is specified (otherwise the scaling happens against (0;0) in texture coordinates, which is probably not what you want). |
| | scale_center: float3 | The scaling center. The point (0.5;0.5) for example lies in the middle. The third coordinate is ignored. |
| Out | texture_scale_out: texture | The scaled texture |

Table 8-214.

| scale_one | | Same as scale, but uses one scaling factor for both texture coordinates |
|------------------|----------------------------|---|
| In | texture_in: texture | The input texture |
| | scale_vector: float3 | The scaling for the coordinates. The third coordinate is ignored. |
| | center: complex | |
| | use_scale_center: enum | yes or no, whether a scaling center is specified (otherwise the scaling happens against (0;0) in texture coordinates, which is probably not what you want). |
| | scale_center: float3 | The scaling center. The point (0.5;0.5) for example lies in the middle. The third coordinate is ignored. |
| Out | texture_scale_out: texture | The scaled texture |

Table 8-215.

| tex_parameters | Texture parameters |
|--|---|
| In texture_in:texture parameters: complex | Input texture |
| wrap_s:enum | Wrap mode for first texture coordinate if off-bounds: <ul style="list-style-type: none"> - repeat: Repeat texture - clamp: Use the last pixel of the texture; may show artifacts because of interpolation - clamp_to_edge: Clamps texture coordinate to $[0+t/2;1-t/2]$ where t is a texel width/height (texture pixel) - clamp_to_border: Clamps texture coordinate to $[0-t/2;1+t/2]$ where t is a texel width/height (texture pixel) - mirrored_repeat: Repeating, i.e., taking only the fractional part of texture coordinates, but alternate switching $[0;1]$ to $[1;0]$ |
| wrap_t:enum | Wrap mode for first texture coordinate if off- bounds, values the same as for wrap_s |
| border_color: float4 | Specifying a border color |
| anisotropic_filtering:enum | One of no or yes. Whether to use anisotropic filtering. If yes, improve quality of interpolation, but costs performance |

(continued)

Table 8-215. (continued)

| tex_parameters | Texture parameters |
|--------------------------|--|
| min_filter:enum | <p>What to do if a texel (texture pixel) is smaller than the area it is mapped to. One of:</p> <ul style="list-style-type: none"> - nearest: use the value of the nearest texture element, measured from the center of pixel which is being textured - linear: apply a linear interpolation between adjacent texture pixels - nearest_mipmap_nearest: first choose a mipmap which most closely matches the pixel size, then proceed as with “nearest” - linear_mipmap_nearest: linearly interpolate between the two closest matching mipmaps, then proceed as with “nearest”. - nearest_mipmap_linear: first choose a mipmap which most closely matches the pixel size, then proceed as with “linear” - linear_mipmap_linear: linearly interpolate between the two closest matching mipmaps, then proceed as with “nearest”. |
| mag_filter: enum | <p>What to do if a texel (texture pixel) is bigger than the area it is mapped to. One of:</p> <ul style="list-style-type: none"> - nearest: use the value of the texture element which is nearest (Manhattan distance) to the center of the textured pixel. - linear: linearly interpolate between the four adjacent texture pixels. |
| Out texture_out: texture | Output texture |

Table 8-216.

| translate | | Translates a texture |
|------------------|--|---|
| In | texture_in:texture translation_vector: float3 | The input texture The translation vector. The third coordinate is ignored. |
| Out | texture_translate_out:texture | The output texture |

OpenGL

Operations close to the OpenGL standard functionalities. The path is Texture → OpenGL. See Tables 8-217 to 8-219.

Table 8-217.

| 6bitm2cubemap | | A cubemap maps six faces on the surface of a cube. You could for example connect the output of Renderers → xtra → skybox module to this module and later in the pipeline use a shader to do the rendering |
|----------------------|---|--|
| In | bitmaps:complex positive_x:bitmap negative_x:bitmap positive_y:bitmap negative_y:bitmap positive_z:bitmap negative_z:bitmap | The bitmaps |
| Out | texture_out:texture | The output texture |

Table 8-218.

| | | |
|---------------------|---|---------------------|
| texture_bind | Provides a texture to objects that do not provide their own textures. Note that it depends on the modules earlier in the pipeline if the texture coordinates are set correctly. So binding a texture to modules providing their own texture may lead to an undefined behavior if you use this module | |
| In | render_in: render | The renderer input |
| | tex_in:texture | The texture to bind |
| Out | render_out: render | The renderer output |

Table 8-219.

| | | |
|--------------------------|---|--|
| texture_coord_gen | Lets OpenGL provide texture coordinates (for texture mapping) for objects. | |
| In | render_in:render | The renderer input |
| | gen_s:enum | How to generate the first texture coordinate |
| | gen_t:enum | How to generate the second texture coordinate |
| | gen_r:enum | How to generate the third texture coordinate. Probably unused, since ThMAD doesn't support 3D textures |
| | parameter_s: float3 | A parameter to the first coordinate mapping, if applicable |
| | parameter_t: float3 | A parameter to the second coordinate mapping, if applicable |
| | parameter_r: float3 | A parameter to the third coordinate mapping, if applicable |
| Out | render_out: render | The renderer output |

The following mechanisms apply, according to the input parameters:

- Enumeration `gen_* = OFF`: no generation
- Enumeration `gen_* = OBJECT_LINEAR`:

They apply this function:

$$g = p_1 \cdot x_0 + p_2 \cdot y_0 + p_3 \cdot z_0 + p_4 \cdot w_0$$

where the (x_0, y_0, z_0, w_0) are the object coordinates and p_i the parameters supplied in the corresponding `parameter_*` anchor. The g is then the texture coordinate value. This mathematically describes a projection onto the vector provided in the parameter anchor and measuring the distance to the origin.

- Enumeration `gen_* = EYE_LINEAR`: Similar to `OBJECT_LINEAR`, but `p` first is multiplied by the inverse of the `MODELVIEW` matrix. This means the calculation will follow the camera's position
- Enumeration `gen_* = SPHERE_MAP`: Texture coordinates are generated in such a way that the object seems to be reflecting the texture pixel data.
- Enumeration `gen_* = NORMAL_MAP` or `REFLECTION_MAP`: Refers to cube maps, where the texture represents the six faces of a cube. The way the normals are being calculated depends on the interpretation of the texture data, which is different for a `NORMAL_MAP` compared to a `REFLECTION_MAP`. The details are rather complex, so the reader is asked to consult OpenGL tutorials or documentation which can be found in the net for that purpose.

Particles

A couple of artificial textures, which can be used for particle systems, but also for other purposes. The path is `Texture → Particles`. See Tables 8-220 and 8-221.

Table 8-220.

| blob | A blob, star, or leaf, depending on the settings |
|----------------------------------|---|
| In <code>settings:complex</code> | |
| <code>arms:float</code> | Number of arms if you want a star or leaves |
| <code>attenuation: float</code> | Increase this number to decrease blurriness |
| <code>star_flower: float</code> | Increase this number to make the center parts thinner, as for flower leaves |
| <code>angle:float</code> | Specify a rotation angle here |
| <code>color:float4</code> | The color to use |
| <code>alpha:enum</code> | One of no or yes. If yes, use the ALPHA to create the transparency in the shape. Otherwise pre-multiply color values with the ALPHA given in <code>color</code> , and let the output ALPHA = 1.0 everywhere |
| <code>size:enum</code> | One of: 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 |
| Out <code>texture:texture</code> | A pointer to the texture on the graphics hardware |

Table 8-221.

| concentric_circles | | Generates a texture with concentric circles |
|---------------------------|-------------------|---|
| In | settings:complex | |
| | frequency:float | Controls the spacing between the circles |
| | attenuation:float | Controls the sharpness of the circles |
| | color:float4 | The color to use |
| | alpha:enum | One of no or yes. If yes, use the ALPHA to create the transparency in the shape. Otherwise pre-multiply color values with the ALPHA given in color, and let the output ALPHA = 1.0 everywhere |
| | size:enum | One of: 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, or 2048x2048 |
| Out | texture:texture | A pointer to the texture on the graphics hardware |

Macros

Macros are generated using the Artist GUI. They are containers for sub-pipelines and behave like modules with input and output anchors.

Once you save the macros, they will appear in the module lister menu the next time you start Artiste. In the operating system file structure, macros will end up here:

```
/home/[USER]/thmad/[VERSION]/data/macros/
```

ThMAD in the current version cannot build the menu dynamically, which is why you have to restart it after you make changes to the module structure.

Summary

This chapter listed all modules available in ThMAD and described all their parameters.

Index

■ A

- Accumulators, 225–226
- Analog-to-digital conversion (ADC), 3
- Arithmetical operations, 226
- Array, 232–233
- Artiste operation, ThMAD
 - fullscreen mode, 41
 - fullwindow mode, 40
 - options, 38
 - performance mode, 41
 - stopping method, 42
 - windowed mode, 37–39

■ B

- Binary operators, 227–229
- Blobs, 108
 - basic, 109, 113
 - bitmap, 109
 - in different variations, 110
 - Perlin noise, 113–114, 116
 - sub-pipeline, parameters, 111–112
 - transformation, 112

■ C

- Coordinate systems, 51–54

■ D

- 3D concepts
 - coordinate systems, 51–54
 - lights
 - ambient, 59
 - diffuse, 59
 - material parameters, 60

- reflection modes, 59, 61
- screen's clear color, 59
- specular, 59
- space mapping (*see* Space mapping)
- ThMAD meshes, 64
- ThMAD particle systems, 64–65
- translation, rotation, and scaling, 57–58

DVD Styler, 21

■ E

- Eye and camera
 - aforementioned depth buffer, 63
 - backface culling, 64
 - device-normalization, 62
 - modelview transformation, 61
 - projection transformation, 62
 - screen coordinates, 63

■ F

- Fast Fourier transformation (FFT), 319
- Float anchors, 194
- Fourier transformation, 4

■ G, H

- Graphical module chooser, 184
- GUI functionalities, ThMAD Artiste assistant, 184–185
 - desktop
 - empty state, 172
 - keyboard shortcuts, 172–173
 - mouse shortcuts, 173–174
 - fullscreen mode, 176–177
 - main menu functions, 178–180

GUI functionalities, ThMAD Artiste (*cont.*)
 module choosers (*see* Module choosers)
 modules (*see* Modules)
 notes, 206
 object inspector, 186
 performance mode, 176
 saving and loading states, 186–187
 standard and fullwindow mode, 175
 starting and stopping, 171
 window modes, 174

■ I, J, K

Inverse Fourier transformation, 4

■ L

Light reflection modes, 59
 Linear transformation, 54

■ M, N

Macro anchors, 205
 Main menu functions, 178–180
 Meshes, 64
 Module choosers
 graphical, 183–184
 module list, 181–182
 Modules
 anchors
 connections, 192–193
 types, 190–191
 cloning, 190
 connectors, 190
 enumeration type input anchor, 194
 float type input anchor, 194–195
 float3 type input anchor, 195–196
 float4 type input anchor, 197
 macros, 203–206
 placing and deleting, 189
 quaternion type input anchor, 198
 resource selector, 200
 sequence editor anchor
 interpolations, 202
 sequence input, module parameter
 float_sequence module, 200
 interpolation type, 202
 sequence editor, 201
 string type input anchor, 198–199
 types, 188–189
 visuals, 203

■ O

Object inspector, 186
 Ocean module, 86
 anchor modules, 88–90
 direct color entry process, 88
 graphical color chooser, 87
 output, 90
 quadruple slider combo, 86
 sky color, setting, 86
 Oscillators
 float_sequencer, 240–241
 inside_range, 241–243
 pulse_oscillator, 245, 247
 Oscilloscopes, 300–301

■ P, Q

Particle systems, 64–65
 Perlin noise blobs, 113–114, 116
 Player operation, ThMAD
 faders
 installation, 47–48
 smooth transitions, 47
 transition state, 48
 options, 45–46
 starting, modes, 43–44
 stopping method, 47
 PulseAudio, 2, 5, 7
 input device objects, 13
 playback objects, 12
 to ThMAD, 11–15

■ R

Rotation, 58

■ S

Sound card, 4
 Sound input, 1
 Sound modules
 input_visualization_listener,
 319–320
 midi → aka_apc40_controller, 321
 ogg_sample_*, 321
 raw_sample_*, 322
 Sound structure
 frequency-power representation, 3–4
 sound card, 4
 time-elongation representation, 3

Space mapping
 embracing transformations, 56
 homogeneous coordinates, 54
 identity matrix, 57
 linear transformation, 54
 transformation combinations, 55
 transformation matrix, 54

String modules, system level
 modules, 323–328

System sounds, 11

■ T

Ternary operators, 230

Texture mapping, 90, 92
 automatic texture coordinates,
 93, 95
 anchor parameters, 94–95
 camera module, 95
 on cube, 97–98
 glTexGen() function, 92
 NORMAL_MAP mode, 96
 OBJECT_LINEAR mode, 96, 97
 onto sphere, 92, 96
 SPHERE_MAP and
 REFLECTION_MAP modes, 97

basic, 91

floating textures
 anchor parameters, 101–102
 distortions, 106–108
 macro, 103–105
 mesh_grid module, 106
 modifiers, 102
 output, 102
 plane mesh generator, 99
 reusable sub-pipelines, 103

shaders, 92

Texture modules
 buffers, 329–330, 332–333
 dummy modules, 334
 effects, 334–335
 loaders, 335–337
 macros, 344
 modifiers, 337–340
 OpenGL, 341–343
 particle systems, 343–344

Textures
 blurring effect
 blend_mode modules,
 119, 121, 122
 blurring distortion, 127–129

colored_rectangle module, 120
 dest_blend anchor, 124
 grid, 119
 highblur, 122
 Maths → oscillators →
 oscillator, 126
 mesh_basic_render, 120
 mesh_sequ_distort module, 127
 module texture → particles →
 blob, 124
 OpenGL behavior, 121
 oscillator, 119
 parameters, 118
 renderers → basic → textured_
 rectangle, 125
 texture_out, 122
 tremendous effects, 118

description, 117

objects, 163–169

particle systems
 center clamped particle
 systems, 159–161
 emitter types, 145
 image bit particles, 154–159
 methodology, 145
 MODIFIER conversion, 145
 RENDERER conversion, 146
 ribbon particles, 161–162
 ThMAD, 145
 waterfall, 146–148, 150–154

self-similarity
 ALPHA value, 135
 backfeeding pipeline,
 132–133
 fancy Koch curve, 137,
 139, 141
 gl_color module, 139
 Koch curve, 130, 136, 142, 144
 mesh_basic_render modules, 136
 parameters, 133–134
 pythagoras tree, 130
 rendering and backfeeding
 pipeline, 134
 render_surface_single
 modules, 135, 139, 142
 ribbon module, 137–138
 rotation module, 141
 set parameters, 131–132
 ThMAD, 131
 trees, 129

ThMAD, 117

ThMAD

- Artiste startup window, 9
- blobs (*see* Blobs)
- connecting PulseAudio sound to, 11–15
- 3D rendering pipelines, 67
 - advantages and disadvantages, 67
 - anchor values, 68–69
 - complex anchors, 70
 - ocean module, 86–88, 90
 - scaling, 77–79
 - transformations, 67–68, 70–75, 77–79
 - translation, 76
 - wireframes, 79–85
- 2D sample, 21
 - Artiste, 21–22
 - assistant modes, 23–24
 - automatic mode, 24
 - complex anchors, 26–27
 - module browser, 26
 - performance mode, 25
 - rectangle on canvas, 24–25
 - sampling rate, 28–29
 - Ubuntu desktop, 28
 - wobbling rectangle, 27–28
- 3D sample, 29
 - anchor values, 33–35
 - complex anchors, 31
 - elaborated 3D scene, 32
 - input anchors, 33
 - mesh_box, 30
 - orbit_camera, 30
 - sound input, 33
 - surface material, 29
 - surface properties, 30
- installing, 8–11

ThMAD modules

- bitmap filters, 212–218
- bitmap generators, 219–220, 222
- categories, 210
- data types, 209–210
- dummies, 223–224
- loads bitmaps, 222
- math modules
 - accumulators, 225–226
 - arithmetical operations, 226
 - array, 232–233
 - binary operators, 227–229
 - color, 233–234
 - converters, 234–236

- dummies, 236–237
- functions, 230–231
- interpolation, 237–239
- limiters, 239–240
- ternary operators, 230
- unary operators, 226–227

mesh

- dummies, 247
- generators, 248–249
- importers, 249–250
- particle system, 258–259
- segmesh modules, 259
- solid, 259–263, 265
- texture, 265–266
- vertices, 266–268
- Xtra, 269

modification of bitmaps, 223

modifiers

- color, 251
- converts, 251
- deformers, 252–254
- helpers, 255
- pickers, 255–256
- transforms, 256–258

particlesystems

- fractals, 269–270
- generators, 271–275
- modifiers, 275–278

renderers

- basic, 279–283
- mesh, 283, 285–287
- OpenGL modifiers, 288–297, 299
- oscilloscopes, 300–301
- particlesystem, 301–306
- shaders, 306–307
- text, 308
- Xtra, 309–310

screen, 211

selectors, 310, 312–313, 315–318

ThMAD Player

- GUI operations, 206
- keyboard operations, 207
- starting and stopping, GUI, 206

Time-elongation representation, 3

Toolchain, 7

- making DVD from recording, 21
- recording video, 15, 17–18, 20

ThMAD

- connecting PulseAudio sound to, 11–15
- installing, 8–11

Transformation matrix, [54](#)
Translation, [57](#)

■ **U, V**

Ubuntu Linux system, [7-8](#)
Unary Operators, [226-227](#)

■ **W, X, Y, Z**

Wireframe models, [79](#)
 massive and dynamic, [85](#)
 scale anchor, [80](#)
 sphere anchors, [81-84](#)
 translation anchor, [80](#)
 triple value slider, [81](#)