Blender2faust: from drawn 3d objects to physically based sound models

Smilen Dimitrov, Romain Michon, Stefania Serafin

Blender2faust: from drawn 3d objects to physically based sound models

Smilen Dimitrov
Aalborg University, Copenhagen
sd@create.aau.dk

Romain Michon
CCRMA, Stanford University
rmichon@ccrma.stanford.edu

Stefania Serafin
Aalborg University, Copenhagen
sts@create.aau.dk

ABSTRACT

Faust is a functional programming language for audio applications, designed for real-time signal processing and synthesis. A part of the Faust source code distribution is the command line tool `mesh2faust` [1]. `mesh2faust` can process a 3D modelled mesh, and generate the corresponding audio physical model, as well as code to play its sound. Here we describe an interface for controlling `mesh2faust` which is implemented as a plugin for the free and open-source 3D modelling software, Blender.

1. INTRODUCTION

Faust is a functional programming language specifically designed for real-time audio processing. To work with Faust, the user typically writes a Faust language script with the extension `.dsp`. This script is then converted using the Faust interpreter to a wide variety of platforms and languages. This intermediate code can be compiled using standard tools on the platform (for example, `gcc`) to executable code that can play and process sound in realtime.

A part of the Faust source distribution is the command-line tool `mesh2faust` [1]. `mesh2faust` is a part of a collection of tools known as Faust Physical Modeling Toolkit [2], which facilitate the design of auditory physical models with the Faust language. The other essential part of this toolkit is the Faust Physical Modeling Library, `physmodels.lib` [3], which spans utilities relevant for physical modelling, mostly derived from the Synthesis toolkit [4].

The tool `mesh2faust`, as typical when interacting with command-line applications, can be configured with a variety of command line options. `mesh2faust` accepts a file path to a 3D model, with material properties and other settings. It exports a Faust `.lib` library file, that contains the corresponding auditory modal physical model. The file can thereafter be used in a Faust `.dsp` script, which would typically render its own GUI, with button elements that can trigger the sound of the auditory physical model.

Thus, there are at least two command line passes in order to get from a 3D model, to an application that would play the sound of its corresponding auditory model - a process that may seem slightly involved for non-technical users. However, even for technical users, the cognitive load would arguably be lessened, if there's a possibility to choose and edit a 3D model visually, and then upon a single click of a button, execute the command-line passes automatically to hear sound - instead of having to deal with correct filenames and parameters in the command line. This is exactly what this preliminary GUI for `mesh2faust` is attempting to address.

2. MESH2FAUST

It is important to note that the `mesh2faust` tool, brings about an important dependency of its own, unrelated to the rest of the Faust software. This dependency is the Vega FEM (Finite Element Method) library [5], a free and open-source software that allows for the conversion of the 3D model, and running finite element analysis on it. Vega FEM in turn depends on the Intel MKL [6] (Math Kernel Library) - a proprietary, but freely (against a registration wall) accessible library; and the open-source `arpack`. Note that the `mesh2faust` repository already includes a light-weight and modified version of Vega FEM within its source code.

Out of the command line options of `mesh2faust`, the only required argument is `--infile`, which specifies the input 3D model, saved in the `.obj` file format (also knows as Wavefront `.obj`). The Blender UI plugin described herein exposes all of these to the user interface, mostly as text field elements.

3. THE BLENDER UI PLUGIN

As mentioned earlier, the `mesh2faust` tool imposes a workflow where the user starts from a 3D model, and ends with an application allowing the user to virtually "hit" (or *excite*) certain modes in the respective 3D model - and hear the sound generated in real-time. The real-time application generated by the system is a `faust2caqt` program. As 3D interfaces have been widely used in computer software in past decades, it can be expected that potential users of this system would immediately imagine a 3D interface as the appropriate one for this workflow. In essence, the workflow from the user's perspective can be reduced to:

- Draw/import the 3D model
- Apply sonic material properties (i.e., how should the model sound like - as if it was made of wood, or metal...) to parts or entirety of the 3D model (including finetuning of parameters)

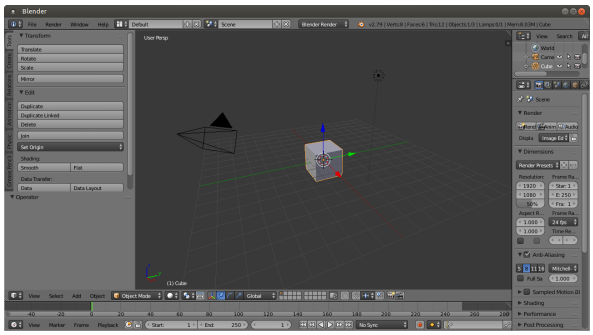

Figure 1. Normal startup UI of the Blender 3D modelling application (version 2.79a, running on Ubuntu 14.04), showing the ”Default” screen layout and the cube model.

- Choose a vertex on the 3D model, where the model will be ”hit”/excited
- Click a button to perform the virtual ”hit”, and hear the sound

Currently there are multiple free & open-source options that could be used as a starting point for such a 3D interface for the `mesh2faust` workflow. Settling on the Blender [7] 3D creation suite for this purpose, was immediate: Blender has strong developer momentum behind it, it is a cross-platform tool, it already includes a user interface for creating and manipulating 3D models, can import and export Wavefront `.obj` files, and has a Python API with a portion specifically dedicated to modifying the Blender graphical user interface itself.

The typical user interface of Blender, after it is started up, is shown on Fig. 1. Note that Blender has several so-called ”Screen Layouts”; the vanilla install starts up in the ”Default” screen layout. This layout has a 3D viewer take up the center of the window, while there is a so-called ”Tool Shelf” on the left, with multiple vertical tabs, such as ”Tools”, ”Create”, ”Animation”, etc (for more details on the Blender UI, see [8]). In addition, Blender at startup instantiates a light, a camera, and a model of a cube.

The approach taken here, was to use Blender’s Python API, to add another tab to the Tool Shelf, specifically related to `mesh2faust`. The Python API of Blender is documented, but not extensively beside several tutorials (such as [9]) - for our approach, we found it useful to consult the autogenerated documentation, starting from `bpy.types.Panel` [10]; otherwise, there are scripts already used in Blender, with names starting with `space_view3d_` (on Ubuntu, they might get installed at the `/usr/share/blender/2.79/scripts/addons` path).

Eventually, our implementation resulted with a single Python file, `space_view3d_mesh2faust.py`, representing the addon. When it runs, it simply sets up the extra tab, titled ”Faust”, its corresponding panel, and all the user elements on it; in principle, it just needs to run once at Blender startup. The script can either be run/manipulated via the ”Scripting” Screen Layout of Blender, or Blender can be started through the command line with:

```
blender --python
    space_view3d_mesh2faust.py
    standardBell.blend
```

A new ”Faust” tab is instantiated in the Tool Shelf, shown on Figure 2. The command line method, however, also ensures that an actual Blender file is loaded (Blender otherwise starts in a state where there is no defined opened file), which in the current incarnation of the workflow is important. The panel has a text field for the physical model name (with a default of `blphysmod`), and then several checkboxes: ”Material Properties”, ”Excitation Positions”, ”Limiting the number of modes”, ”Making a transposable model (freqcontrol)” - activating the checkbox activates additional elements such as textfields, and adds the corresponding command-line options to the command line. For instance, ”Material Properties” simply has a textfield, as it expects the material properties to be entered as a string; for ”Excitation Positions”, there are two buttons: ”Random, limit to” and ”Vertices ID list”, as there are two different command-line arguments depending on this choice (one of them expects a number, the other a list).

Finally, there is the ”Run mesh2faust” button; when it is clicked, the following happens:

- The addon script exports the current state of the 3D model (excluding lights and cameras) of the currently loaded `.blend` file, as a Wavefront `.obj` file of the same name
- The addon composes a command line for `mesh2faust`, with the previously exported `.obj` file, and then calls it; the output is a Faust `.lib` file, however, named as per the physical model name textfield in the addon panel
- The addon patches a template for a Faust `.dsp` script, which contains a simple GUI for exciting a physical model, so it uses the correct filenames from previous steps - then generates this `.dsp` script
- The addon composes a command line for `faust`, so it transpiles this `.dsp` into an ALSA/GTK C++ `.cpp` file
- The addon composes a command line for `gcc`, so it compiles the previously generated `.cpp` file into an executable
- The addon runs the previously generated executable as a separate process, which starts the executable’s GUI as a new window.

This, ultimately, allows the user to edit a 3D model, and then - provided the `mesh2faust` addon settings are already set - simply click the ”Run mesh2faust” button, and be presented with a Faust GUI for auditory inspection of the corresponding modal physical model soon thereafter. As the mention of ALSA/GTK implies, this preliminary incarnation of the addon has only been tested on GNU/Linux (e.g. Ubuntu) systems; however, there are no serious obstacles to further improvements, where the entire cross-platform palette of Faust code output can be utilized.

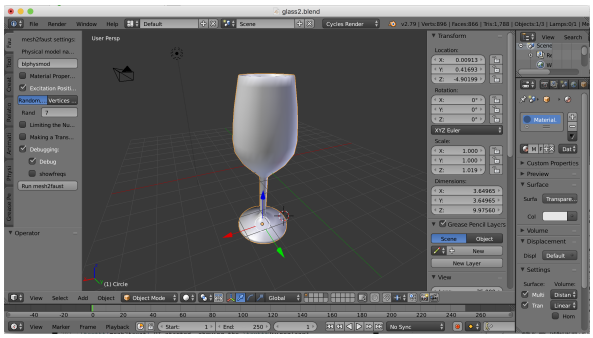

Figure 2. The UI of the Blender 3D modelling application with the `mesh2faust` UI started, showing the `wineglass` model. Notice on the left the new button called mesh2faust, which allows to run the mesh2faust application from Blender.

4. EXAMPLE

In order to show the behaviour of the tool developed, we used the sonification of a wineglass as example. We imported `wineglass.obj` into Blender, and saved it as a native Blender 3D file, `wineglass.blend` (also shown on Fig. 2); on the other hand `wineglass.dsp` is used as the `.dsp` file template in our addon script. Since our addon necessarily exports back to `.obj`, this tests whether the roundtrip of `.obj` through Blender works. As a matter of fact, our tests resulted with a working, realistically sounding executable.

However, specifically for the `wineglass` model, we cannot really talk of a realtime performance: on an Intel i7, 2.70GHz, 4 CPU laptop, it could take up to two minutes for the entire mesh2faust command pipeline to complete. In addition, there are certain models where the `mesh2faust` process breaks - in particular where the faces of the default object are rectangular while `mesh2faust` might works with triangular mesh segments. However, there are also simple models where the `mesh2faust` process completes quickly - for instance, such is the default "Ico sphere" model mesh in Blender.

After the computation is completed, the result is a GUI like the one shown in Figure 3.

5. FUTURE IMPROVEMENTS

Here are possible improvements that the addon could benefit from:

- Have a list of materials in a drop-down, insert it in text field for the "material" argument
- Choose a vertex in the 3D UI - add its `.obj` number to the list of excitable vertices
- Use more relevant UI elements - e.g. sliders - instead of generic textfields in the Blender UI plugin
- Better naming policy: currently, the Blender plugin uses a textfield for the name of the `.lib` file, and the name of the loaded `.blend` file for the `.dsp` script name and the corresponding executable

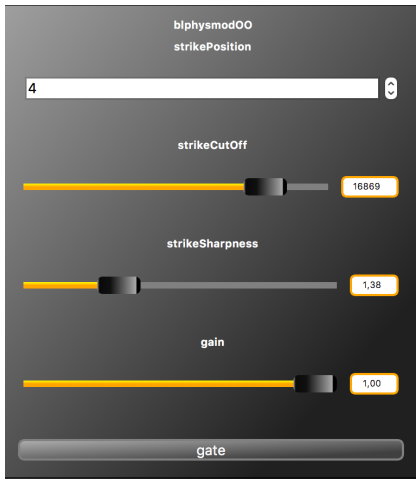

Figure 3. The generated physical model.

- Implement network communication between the Blender plugin and the Faust `.dsp` script executable, to allow for selecting a vertex on the 3D mesh, and having it be triggered right from the Blender UI plugin
- Implement a list mapping colors to material properties in the Blender UI plugin, so that objects with multiple auditory materials can be created easily in Blender by coloring their 3D mesh

6. CONCLUSIONS

In this paper we presented blender2faust, an add-on to the Blender software which allows to create physically based sound models based on 3D rendered objects in Faust. The add-on has been tested on both Linux and macOS machines, with Blender version 2.79 and the latest Faust release from github (https://github.com/grame-cncm/faust). In future editions we plan to extend the capabilities of the tool by improving the interface to select size and material of the rendered objects.

Acknowledgments

This paper was partially supported by NordForsk's Nordic University Hub Nordic Sound and Music Computing Network NordicSMC, project number 86892.

7. REFERENCES

[1] github.com, "mesh2faust," WWW, Last Accessed: 12 March, 2018. [Online]. Available: https://github.com/grame-cncm/faust/tree/master-dev/tools/physicalModeling/mesh2faust

[2] R. Michon, "Faust physical modeling toolkit," WWW, Last Accessed: 12 March, 2018. [Online]. Available: https://ccrma.stanford.edu/~rmichon/pmFaust/

[3] grame.fr, "Faust physical modeling library - faust libraries documentation," WWW, Last Accessed: 12 March, 2018. [Online]. Available: http://faust.grame.fr/library.html#physmodels.lib

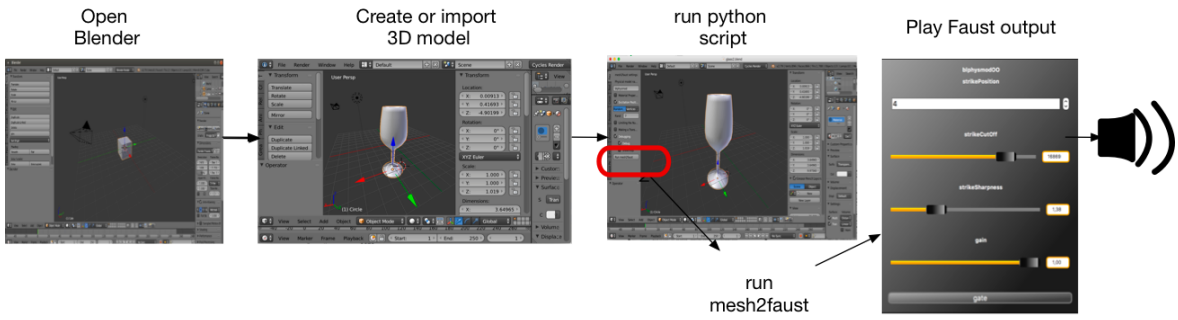

Figure 4. Visualization of the process from creating a model in Blender to synthesizing it in Faust.

[4] P. R. Cook and G. P. Scavone, "The synthesis toolkit (stk)." in *ICMC*, 1999.

[5] F. S. Sin, D. Schroeder, and J. Barbič, "Vega: Non-linear fem deformable object simulator," in *Computer Graphics Forum*, vol. 32, no. 1. Wiley Online Library, 2013, pp. 36–48.

[6] intel.com, "Intel math kernel library (intel mkl) — intel software," WWW, Last Accessed: 12 March, 2018. [Online]. Available: https://software.intel.com/en-us/mkl

[7] blender.org, "Home of the blender project - free and open 3d creation software," Last Accessed: 3/13/2018, 12:03:32 AM. [Online]. Available: https://www.blender.org/

[8] ——, "Docs user interface introduction blender manual," Last Accessed: 3/13/2018, 12:49:23 AM. [Online]. Available: https://docs.blender.org/manual/en/dev/interface/window_system/introduction.html

[9] ——, "Addon tutorial blender 2.65.5 - api documentation," Last Accessed: 3/13/2018, 12:56:31 AM. [Online]. Available: https://docs.blender.org/api/blender_python_api_2_65_5/info_tutorial_addon.html

[10] ——, "Panel(bpy_struct) blender 2.70.4 - api documentation," Last Accessed: 12:59:01 AM. [Online]. Available: https://docs.blender.org/api/blender_python_api_2_70_4/bpy.types.Panel.html