Blendnik: A Real-Time Performance System Using Blender and Pure Data

Nicholas J. Porcaro
Independent Musician/Software Engineer
46 Belvedere Street
San Francisco, CA 94117
(01) 415-553-8769
nick@porcaro.org

Ellen Levy
Independent Artist
151 1st Av #56
New York, New York 10003
(01) 212-982-6399
elleniya@hotmail.com

ABSTRACT

The authors, a graphic designer/painter, and a musician/software have developed a performance system for re-interpreting their live free improvisational jam sessions. The open source systems Pure Data, Open Sound Control, Blender, and Python were used to provide bidirectional mapping of aural and visual parameters and real-time interaction via MIDI.

Keywords

Painting, improvisation, Python, Blender, Pure Data.

1.INTRODUCTION

Over the last several years, we have collaborated on a series of short pieces, usually lasting a few minutes, comprising of freely improvised piano along with live painting, usually pastel/acrylic/ink on vellum.

At some point we wanted to take our collaboration to the next level, so we researched ways to digitize the music and images for purposes of live interaction. We first experimented with various common commercial applications, but found they were unsuitable for our sensibilities and budget. Then we discovered Blender [5] and Pure Data (Pd) [17].

We experimented with mapping scans of the paintings onto 3D objects, and used the built-in Blender Game Engine (BGE) [21], [22] for real-time rendering. A Python [18] script running in the BGE was written using Open Sound Control (OSC) [16] to provide two-way communication with a Pd patch.

A looping sampler implemented in the Pd patch can be configured by creating properties in a Blender scene which specify audio samples and the mappings between aural and visual parameters.

For instance, X location could map to left-right balance, Z location to pitch, Y location to reverb decay, X rotation to flange, Y rotation to tempo, Z rotation to dry/wet mix and transparency to volume. Shape morphing and camera control are also supported.

These mappings can be changed in real-time from a computer anywhere on the Internet. The computer could also be connected to a MIDI controller or sensor. Live music could be performed along with the loops, perhaps using a physically modeled guitar we are developing.

We plan to use this system to explore relationships between visual art and music in the context of live performances and gallery installations.

2.BLENDER AS A FRONT-END DESIGN ENVIRONMENT

Blender is an excellent open source 3D animation/modeling system. It can be used to create complex textured objects (or meshes) which can be animated in real-time in the integrated BGE. Our process starts by designing a mesh inspired by a painting, or series of paintings. This can be done in a variety of ways. A very basic technique would be to create a cube and then edit it by moving around vertices, extruding faces and so forth [13]. More advanced techniques such as multiresolution sculpting can also be employed [15].

Complex Hollywood-style animations have been done in Blender [7], [4] but for real-time applications there is a limit of about 10,000 faces per scene. This number can be reduced more depending factors such as the number of lights and two-sided alpha-enabled faces used. The graphics processor is also a major factor. Our MacBook Pro with a recent graphics processor far out-performs our old PowerBook G4.

Despite these limitations, there has been impressive work done in real-time using soft bodies [9] and GLSL filters [1].

Once the mesh has been designed, a texture can be applied from an image file. Procedural textures are also supported.

To apply the texture in a predictable way, UV mapping [20] is used. UV maps can be exported to an image file, which can then be used as a background layer in an image manipulation program. Imagery can then be drawn over this background layer, which will appear on the mesh when the image file is reloaded into Blender. The background layer can be discarded if desired.

Figure 1 shows a mesh for a section of an abstract tunnel scene we are developing, Figure 2 shows the UV map, Figure 3 shows imagery drawn over the map, and Figure 4 shows a screen shot from the BGE.

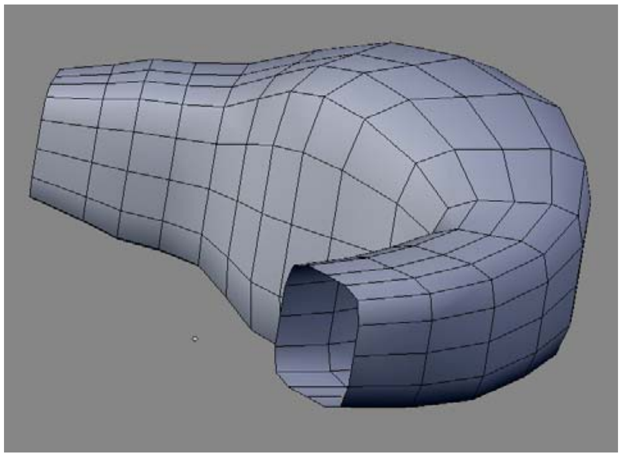

Figure 1. Abstract tunnel mesh.

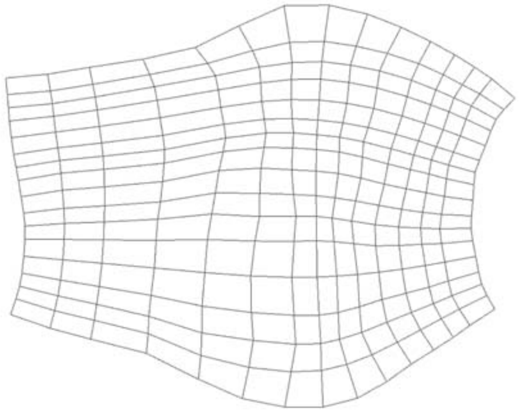

Figure 2. UV map for abstract tunnel.

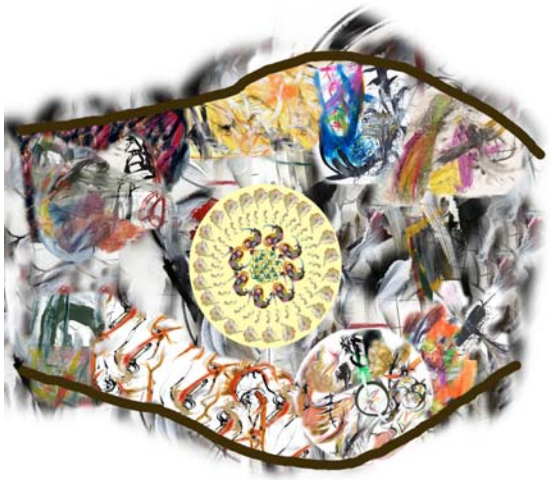

Figure 3. Imagery drawn over map.

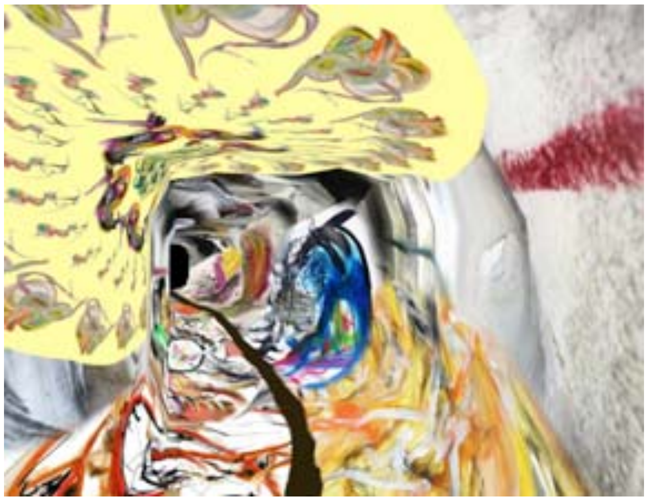

Figure 4. Screen shot of abstract tunnel in the BGE.

The next stop is animation. Each object in a Blender scene can have curves that specify how it moves, what happens to its material properties and so forth. These curves are called IPO (interpolation) curves [3], [11].

Each "curve" has a collection of "channels". We currently support use of the *LocX*, *LocY*, *LocZ*, *RotX*, *RotY*, *RotZ*, and *ColA* channels.

3.BLENDER GAME ENGINE AND PYTHON

After the animation is working, we define how the image will be used in a scene. Currently it can either be associated with a sound in Pd (which we call "blendnik enabled"), or it can just run as a regular BGE object and still use all the usual BGE Logic Bricks and properties [14].

If the object is going to be blendnik enabled, then certain BGE properties, which we call “mapping properties”, can be defined to specify the nature of the behavior.

Currently you can specify a sound file to play in a looping sampler, and *LocX, LocY, LocZ, RotX, RotY, RotZ, ColA* can be mapped to audio parameters for pitch, tempo, scratching, volume, balance, dry/wet mix, reverb level, reverb decay, flange level and 3 flange parameters.

Blender supports embedded Python scripting [8], which can be used to greatly increase the functionality of Blender and the BGE [2]. A Python script (blendnik.py) was written to manage the connection with Pd. A corresponding Pd patch, blendnik.pd was also created. Blendnik.pd and blendnik.py use OSC [12], [10] for interprocess communication.

Every blendnik-enabled object in the scene must specify a *Python Controller Logic Brick*, calling the function blendnik.process(), which is defined in blendnik.py.

When the BGE is running, blendnik.process() is called on every frame and provides Pd with the most recent ordinate values of the *LocX, LocY, LocZ, RotX, RotY, RotZ, ColA* IPO channels.

Depending on how the BGE mapping properties are defined, each of these values affects the corresponding sound parameter. Values come out of blendnik.py normalized and then a scale and offset is applied in blendnik.pd.

For example, say the animation is running at frame N and say a string BGE mapping property named **LocX** with value **balance** is defined for an object named **s1** in the scene.

The normalized value of the *LocX* IPO channel at frame N is sent via blendnik.process() to the balance parameter in the sampler instance for **s1** in blendnik.pd. In this particular example, the minimum value of *LocX* would correspond to a fully right-panned balance and the maximum value of *LocY* would correspond to a fully left-panned balance.

Reasonable default scale and offset values are defined in blendnik.pd, which can be overridden by BGE properties if desired.

4.CONNECTION TO PD

When the BGE is launched, blendnik.py launches blendnik.pd, sending pertinent data from the current Blender scene, such as instance names, sound files and other initial values specified on BGE properties.

After this happens the blendnik.py and the blendnik.pd are ready to communicate via OSC, and proceed as follows:

4.1PD TO BLENDER

When a slider is moved in Pd, or when a note or controller comes in from a MIDI device, Pd sends an OSC message to blendnik.py, which can make objects move or modify their appearance. Sliders can be ganged together to send multiple messages at the same time, and additional logic can be developed for more complex interactions. For example, you could have a slider that sends rotations which are opposites of each other and simultaneously sends a transparency value run through a lookup table. We call this idea "meta controllers".

4.2BLENDER TO PD

On every animation frame, blendnik.py sends the values of the *RotX, RotY, RotZ, LocX, LocY, LocZ* and *ColA* channels to blendnik.pd, which can modify the sound currently playing.

5.THE PD PATCH, BLENDNIK.PD

Blendnik.pd consists of the following subpatches:

s1 ... s12:
Subpatches for each blendnik-enabled Blender object. These subpatches contain a modified version of the looping sampler example B14.sampler.rockafella.pd that comes with Pd, with extra features for pitch bend, reverb and flanging. Up to 30 instances can exist under the current scheme, corresponding to 30 separate audio output channels.

control:
Sends messages to selected BGE objects via the s subpatches, via "master control" subpatches, which correspond to the various types of motion and sound parameters.

sound:
Provides for loading different sounds during a performance, turning on and off audio processing, and sound output.

init:
Initialization, deals with the default values and other initial condition handling.

comm:
Communication with Blender via OSC

axiom49:
Simulation of a MIDI keyboard controller. Presents a scheme for choreographing a performance.

midi:
MIDI input support and simulation used by axiom49.

6.EXAMPLE SCENE

Figure 5 shows a screen shot of the system running a Blender scene which uses one of our collages. Figure 6 shows part of the axiom49 subpatch and Figure 7 shows part of the blendnik.pd patch.

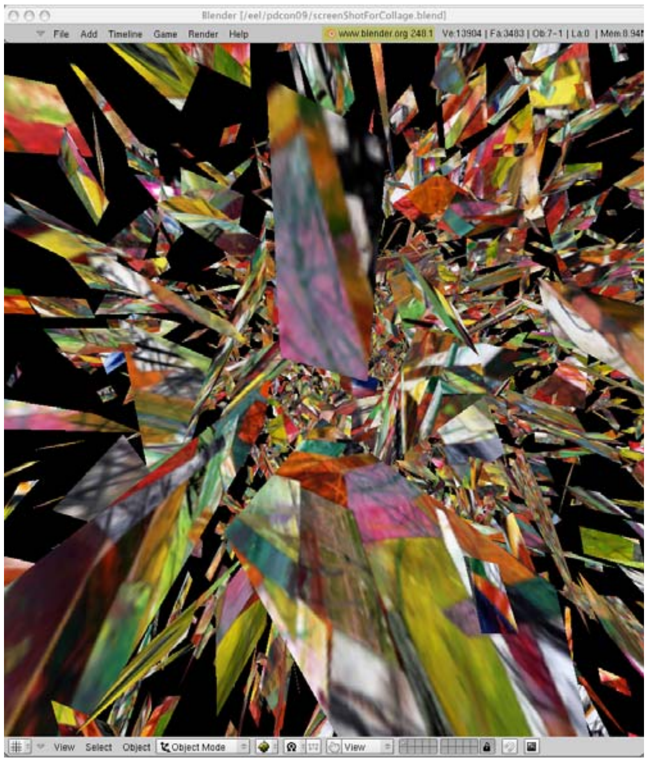
Blender [/eel/pdcon09/screenShotForCollage.blend]
File
Add
Timeline
Game
Render
Help
www.blender.org 248.1
Ve:13904 | Fa:3483 | Ob:7-1 | La:0 | Mem:8.94
View
Select
Object
Object Mode
View

Figure 5. Screen shot of system running a scene.

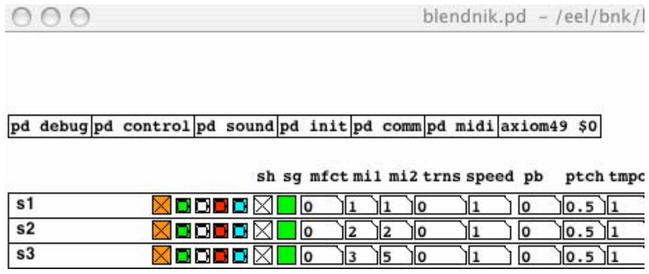

Figure 6. Part of blendnik.pd patch

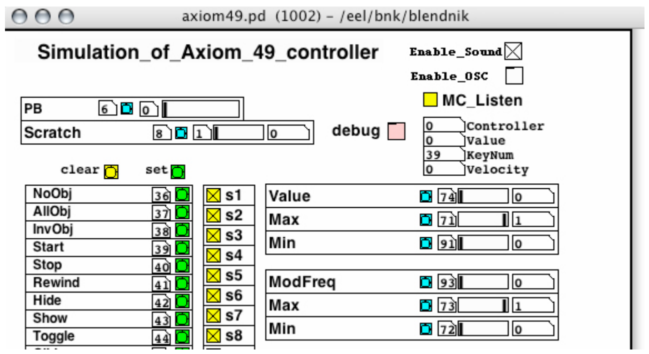

Figure 7. Part of axiom49.pd subpatch

7.FUTURE WORK

- Create a performable multi-sound channel MIDI-controlled installation and performance scenario.
- Allow for creation of new objects on the fly and dynamic allocation/deallocation of looping sampler instances,
- Make use of BGE physics.
- Better camera control to support 3D input devices.
- Optimize sampler, perhaps a new external written in C++ based on the STK [19]
- Integration of one-shot sampler and a physically modeled electric guitar.
- Fix bugs in Blender related to alpha sorting
- Make Blendnik generally available when it becomes more stable. See [6] for the latest progress

8.ACKNOWLEDGMENTS

Thanks to Ton Roosendaal and Blender foundation for developing Blender and Miller Puckette for developing Pure Data. Also thanks to the UC Berkeley Center for New Music and Audio Technology (CNMAT) for developing OSC. A special thanks goes to Julius Smith, my mentor from The Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University, who has egged on my experimentation over the years, Finally a big thanks goes to my late father Carmine Porcaro for his overall support and inspiration.

9.REFERENCES

[1] Advanced GLSL filter demo: http://blenderartists.org/forum/showthread.php?t=152343

[2] BGE Python API: http://www.blender.org/documentation/248PythonDoc/GE/class-tree.html

[3] Basic animation: http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Basic_Animation

[4] Big Bucks Bunny: http://www.bigbuckbunny.org

[5] Blender: http://www.blender.org

[6] Blendnik: http://www.porcaro.org/blendnik.html

[7] Elephant's Dream: http://www.elephantsdream.org

[8] Embedded Python: http://www.python.org/doc/2.5.2/ext/embedding.html

[9] GLSL bathroom demo: http://blenderartists.org/forum/showthread.php?t=137038

[10] Holth, D., McChesney, C. Open Sound Control for Python, http://www.ixi-software.net/content/body_backyard_osc.html

[11] IPO curves and key frames: http://wiki.blender.org/index.php/Doc:Manual/Animation/Basic/Tools/Ipo_Curves_and_Keyframes

[12] IXI Audio: http://www.ixi-software.net

[13] Learn to Model: http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Learn_to_Model

[14] Logic Bricks: http://wiki.blender.org/index.php/Doc:Manual/Game_Engine/Logic_Bricks

[15] Multiresolution Modeling: http://wiki.blender.org/index.php/Doc:Manual/Modelling/Meshes/Multiresolution_Mesh

[16] Open Sound Control: http://opensoundcontrol.org

[17] Pure Data: http://puredata.info

[18] Python http://www.python.org

[19] Synthesis Tool Kit: http://ccrma.stanford.edu/software/stk

[20] UV Mapping http://wiki.blender.org/index.php/Doc:Manual/Textures/UV

[21] Wartmann, C., and Kauppi, M., The Blender Game Kit, 2nd Edition, Blender Foundation, Amsterdam, the Netherlands, 2008.

[22] YoFrankie! - Apricot Open Game Project http://www.yofrankie.org