TAMPERE UNIVERSITY
OF APPLIED SCIENCES

REAL-TIME INTERACTIVE ANIMATED VISUALISATIONS FOR MUSIC PERFORMANCES

Comparing Blender Game Engine and TouchDesigner

Markku Laskujärvi

Bachelor's thesis
May 2018
Degree Programme in Media

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Media and Arts
Interactive Media

Laskujärvi, Markku:
Real-time interactive animated visualisations for music performances:
Comparing Blender Game Engine and TouchDesigner
Bachelor's thesis 37 pages
May 2018

New possibilities for media artists appear constantly with the development of technology. Increasing processing power has made real-time performances with high resolution possible for less money spent on equipment. Additionally, much of the needed software is available for free.

This thesis takes a look at two software which can be used for creating new media art experiences, namely live music visualisations as part of an event. The two pieces of software are somewhat different in their behaviour and introduce different plans for the user depending on whether they are being paid or not. One of these programmes is a game engine fitted inside a 3D production programme, and the other is a more general-purpose creative programming tool leaning towards building live interactive experiences.

The study was done by comparing these two software and building a similar programme with both of them, to be used in visualizing live music. The study acts as a guide for making simple interactive media art pieces, not going into too much detail however.

Key words: new media art, creative computing, user experience, vjing

CONTENTS

ABBREVIATIONS AND TERMS

TAMK	Tampere University of Applied Sciences
USB	Universal Serial Bus, a standard for connecting external devices to computers
MIDI	Musical Instrument Digital Interface, a standard for connecting musical instruments
VGA	Video Graphics Array, a standard for video connectivity
full HD	a video resolution standard of 1920 by 1080 pixels
GPL	GNU General Public License
GPU	the Graphics Processing Unit of a computer
icosphere	a spherical 3D structure made up of triangles
vertex	a point on any 3D object
shader	a program which tells the GPU how to draw graphics

1 INTRODUCTION

Computers are increasingly necessary tools in current-day society. Their integration into being part of more people's lives has produced software that supports extensive creative use of computer technology. Live music events can host visual creative computing by projecting the outputs or by displaying them on screens. This thesis looks into what kind of interactive graphical applications can be built for visualizing live music events using computers.

Comparing two pieces of software, TouchDesigner and Blender, this thesis aims to develop an understanding on how a live visual performance can be built, what restrictions may occur in either software and what kind of usability they offer. To meet these goals, a test application was built using both software. This test application appears similar on both platforms for the purpose of finding out differences between making applications with each software.

The first part of the thesis takes a look at VJ culture as well as some new media art theory, followed with going into rendering engines. Finally, a detailed description of setting up both Blender and TouchDesigner for music visualization purposes is presented.

2 WHAT IS VJING?

VJing is the act of live manipulation of image as it relates to sound. A VJ often accompanies a musician at an event, creating a performance together for the audience. Most of the time the content is either projected or displayed via monitors. (Spinrad 2005)

The color organ from late 1800s is often thought as one of the starting points for live light performance. However, this can be considered as lighting equipment, since it had no means to produce a recognizable picture.
In the 1950s, overhead projectors were used to generate “wet shows”. This is a performance method with its own visual capabilities from the mixing of different color paint and oil on a glass surface.
In the late 1960s, projected video was first introduced into light shows. However, the equipment required, such as television projectors, was prohibitively expensive at the time. This changed after VCR technology emerged in the late 1970s and made moving image manipulation more affordable. (Spinrad 2005)

Computerized video manipulation caught on in the 1980s with the Fairlight Computer Video Instrument, a video processing unit with the capability of applying effects on video in real-time. (Spinrad 2005)
Computer-based image manipulation has since become a basic task for personal computer systems. The post-rendering age is evident in game development with the likes of Epic Games’ Unreal Engine Blueprint programming system and Neil Blomkamp’s Adam series of short films made in real-time with a realistic look to them.
However, these advances in technology can seem like an obvious development when considering the amount of time human interest has lied in audiovisual relationships. Particularly important medium preceding real-time image manipulation for a VJ to consider is cinema. Films can be interpreted in musical terms by the VJ and then be applied to the work they put out while performing. (Spinrad 2005) Combined, visual arts and music theories span hundreds of years of material to be considered by the contemporary VJ.

3 NEW MEDIA ART

New media art can be difficult to define, its name containing three words subject to ongoing and possibly rather large changes in meaning over time. Often using cutting-edge technology, new media art could be considered to reside within contemporary art.

Christiane Paul states that new media art is most often defined as computational and based on algorithms. She goes on to describe new media art as process-oriented, time-based, dynamic and, real-time; participatory, collaborative and, performative; modular, variable, generative, and customizable. Paul also reminds that these features do not all have to be evident in an artwork but rather they can appear in varying combinations. (Paul 2007)

Lev Manovich describes four principles of new media:

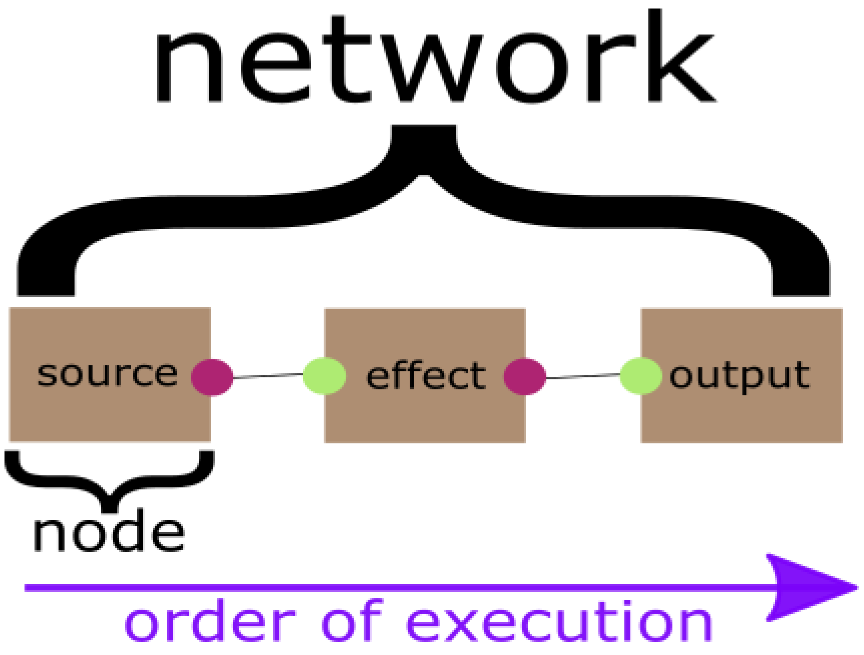
- numerical representation
- modularity
- automation
- variability

The first principle is numerical representation. All new media objects can be described numerically. For example, digital images consist of a certain amount of pixels whose color can be described with a hexadecimal string of numbers. Another example would be the vertex points of a 3D model, which can be described numerically by their coordinates on the X, Y and Z axes.
Secondly, new media objects are modular, consisting of elements made up of units such as pixels, polygons, voxels, characters or scripts. These elements can be rearranged to make new combinations. An example of this could be a video file that would work as a standalone piece of media, but which could be attached to a 3D model by using the video as a texture.
The third principle is automation. One example of this would be the Shiv Integer bot, or artificial intelligence, capable of coming into conclusions based on input which it receives and runs through algorithms embedded within it. In the case of Shiv Integer, the input and output are 3D models which are posted on the thingiverse.com -website.
Manovich's fourth principle is variability. For example, new media objects can be scaled in size, such as 3d objects or fonts. (Manovich 2000)

Thus, using computer programs for VJing could prove as an act of new media art. New media art provides theories to reflect on how VJing can be done using computers. Time is very evident in VJing with music being the starting point for this form of art. It is participatory, collaborative and performative in nature with the audience following the performance and the musician and VJ working together. Modularity, variability, generativity and customizability reside within the computer and the software being used by the VJ to put out visuals in a live event. The real-time aspect of VJing in software is executed by software frameworks often called “engines”.

4 NODES

Since both Blender and TouchDesigner make use of a concept known as “node-based programming” it could prove useful for the reader of this thesis to be described in more detail here. Simply put, a node is a unit of reference in a data structure (NIST 2018). What this means in both the case of Blender and TouchDesigner is that the user can build data structures, or programs, using these preset nodes and connecting them together with “noodles”. In the aforementioned software, nodes are graphically presented to the user with having certain labels and inputs and outputs for connecting different nodes to each other. The main gist of nodes is that the user can make up complex programs visually, without necessarily typing any code.



PICTURE 1. A network of nodes with flow of data and execution illustrated (Laskujärvi, 2018)

5 REAL-TIME RENDERING ENGINES

Real-time rendering in computer graphics is an interactive process in which the user gives input to the computer which then very rapidly produces images based on algorithms run according to this input. By contrast, non-real-time graphics rendering, also referred to as offline rendering, is more concerned with providing high detail to each frame and perhaps applying lighting effects that would be impossible to calculate in real-time. This means that offline rendering often takes time to create an output of an animated film. (Salvator 2018).
The frames per second unit of fps or Hertz is used to determine the real-time rendering system's performance, where changes in fps above 72 are considered effectively undetectable by a human observer. (Haines 2002)
Video games have been typically rendered at 30 to 60 fps, because of the monitor standards gaming systems used earlier (Gregory 2009).
"Engine" is a reference to the video game industry, which uses the term "video game engine" to describe a software framework used to build video games and enabling the reuse of code in this framework for making new games. (Gregory 2009)
So here real-time rendering engines stand for software frameworks that give the user components to produce interactive real-time rendering applications. Below are introduced two of them, both of which are rather unique and different from each other in their user experiences and capabilities.

5.1 Derivative TouchDesigner

Touchdesigner is a visual programming application where nodes, also referred to as operators, are being used to create applications by linking these pre-built software components together. Extensions can be made with Python scripting as well. The application being created in TouchDesigner can be altered while it is running. TouchDesigner enables the user to extensively composite and manipulate image and video in real-time. A 3D engine is also included. (Derivative 2018)

5.2 Blender Game Engine

“The Blender Game Engine (BGE) is Blender’s tool for real time projects, from architectural visualizations and simulations to games.” (Blender Manual 2018)
The Blender Game Engine, or BGE, resides within the 3d modeling and animation software suite Blender. This means 3d models and animations can be made within the same software that is then used to build interactivity for these objects. Interactive applications made in BGE need to be compiled before running.
Programming in BGE can be done by connecting pieces of existing software components known as Logic Bricks, a type of nodes. Extensions can be made with Python scripting as well. (Blender Manual 2018)

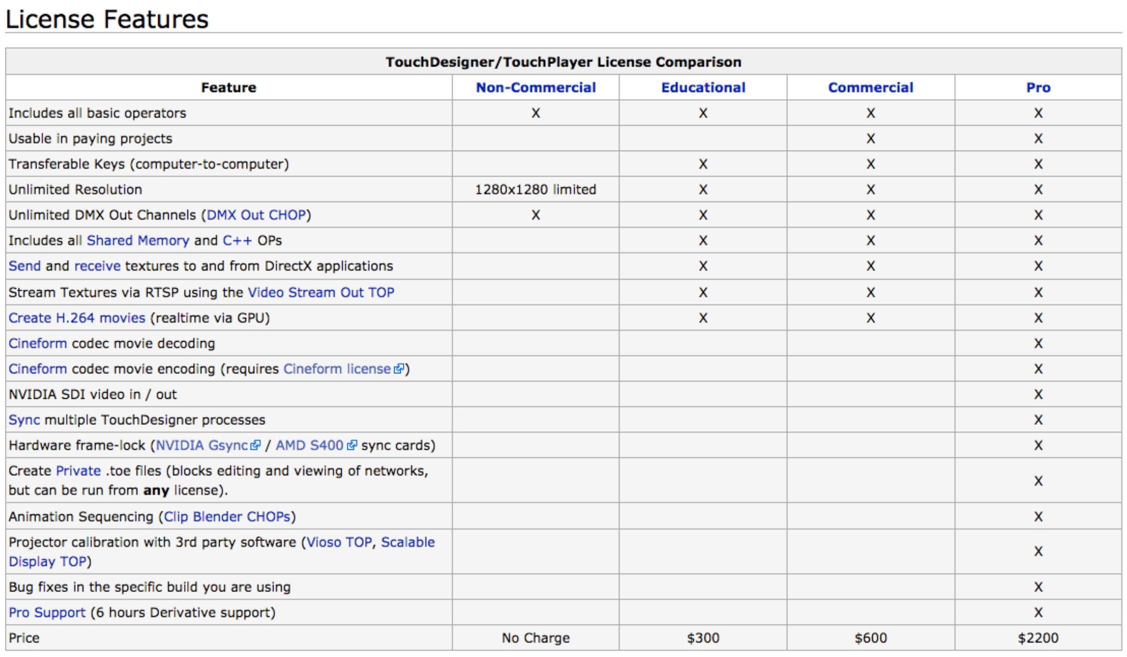
6 TOUCHDESIGNER

6.1 Installation & license

TouchDesigner can be downloaded from https://www.derivative.ca/099/Downloads/. Supported operating systems for TouchDesigner version 099 include Microsoft Windows and macOS.

TouchDesigner Non-Commercial is free to use, given that the user is not using the software to do paid work. (Derivative 2018)

There are some differences between licenses regarding the available features.



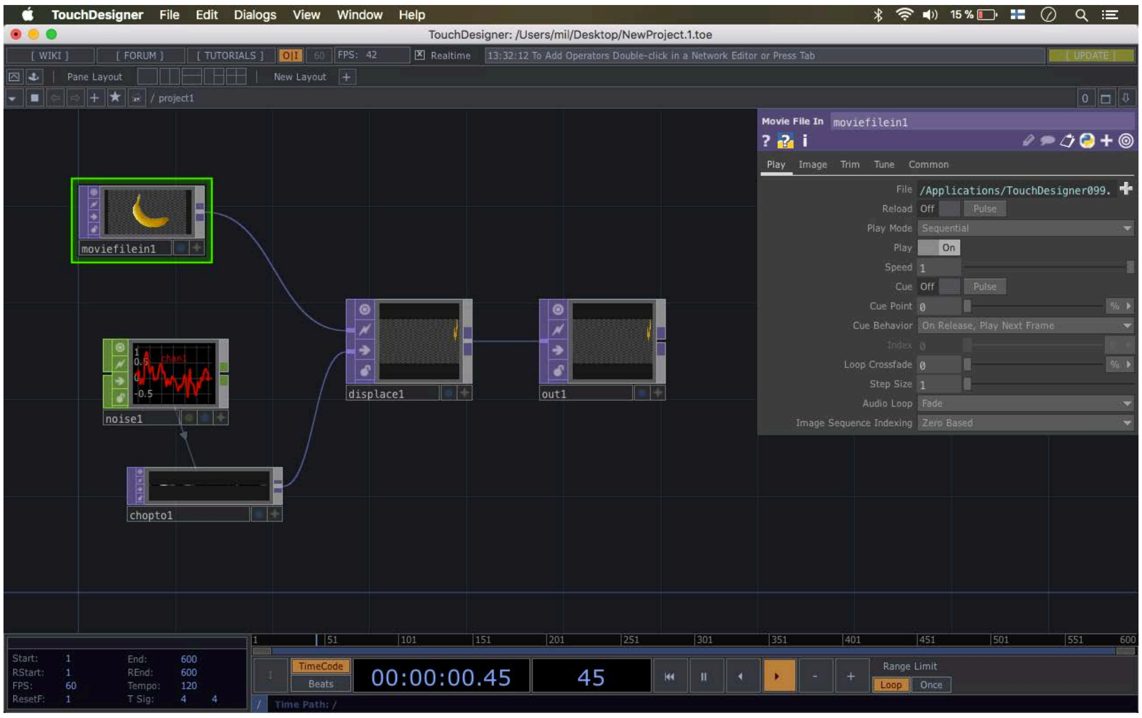
License Features

TouchDesigner/TouchPlayer License Comparison				
Feature	**Non-Commercial**	**Educational**	**Commercial**	**Pro**
Includes all basic operators	X	X	X	X
Usable in paying projects			X	X
Transferable Keys (computer-to-computer)		X	X	X
Unlimited Resolution	1280x1280 limited	X	X	X
Unlimited DMX Out Channels (DMX Out CHOP)	X	X	X	X
Includes all Shared Memory and C++ OPs		X	X	X
Send and receive textures to and from DirectX applications		X	X	X
Stream Textures via RTSP using the Video Stream Out TOP		X	X	X
Create H.264 movies (realtime via GPU)		X	X	X
Cineform codec movie decoding				X
Cineform codec movie encoding (requires Cineform license)				X
NVIDIA SDI video in / out				X
Sync multiple TouchDesigner processes				X
Hardware frame-lock (NVIDIA Gsync / AMD S400 sync cards)				X
Create Private .toe files (blocks editing and viewing of networks, but can be run from **any** license).				X
Animation Sequencing (Clip Blender CHOPs)				X
Projector calibration with 3rd party software (Vioso TOP, Scalable Display TOP)				X
Bug fixes in the specific build you are using				X
Pro Support (6 hours Derivative support)				X
Price	No Charge	$300	$600	$2200

PICTURE 2. Different license types of TouchDesigner, a screen capture from the TouchDesigner wiki (Laskujärvi, 2018)

6.2 User interface

The default user interface of TouchDesigner upon startup is the Network Editor window. In addition to this, there are windows and pop-up dialogs for handling geometry, animations, text editing, performance monitoring, MIDI device mapping and beat detection among others. The default interface shows frames per second at the top of the screen for quick performance monitoring. The amount of frames that are being looped through is shown in the bottom of the screen, along with the tempo.



13:32:12 To Add Operators Double-click in a Network Editor or Press Tab

Movie File In: moviefilein1

File: /Applications/TouchDesigner099.
Reload: Off
Play Mode: Sequential
Play: On
Speed: 1
Cue: Off
Cue Point: 0
Cue Behavior: On Release, Play Next Frame
Index: 0
Loop Crossfade: 0
Step Size: 1
Audio Loop: Fade
Image Sequence Indexing: Zero Based

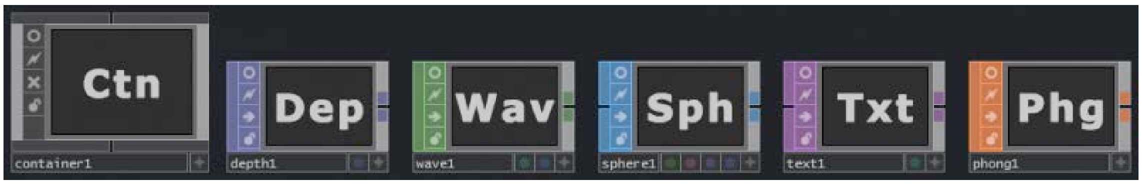
```
Start:   1      End:    600
RStart:  1      REnd:   600
FPS:     60     Tempo:  120
ResetF:  1      T Sig:  4    4
```

00:00:00.45 45
Range Limit: Loop
Time Path: /

PICTURE 3. Screenshot of TouchDesigner running on macOS with some nodes placed in the Network Editor (Laskujärvi, 2018)

6.3 Operators

Operators are the basic building blocks of applications made with TouchDesigner. Both “node” and “operator” are used somewhat interchangeably in the TouchDesigner documentation, defined as follows: “Node is generic, operator is the specific entity that does the work of generating the data output of the node.” Operators are pieces of software, “filter” operators taking inputs and producing outputs and “generator” operators taking no inputs, creating data as their output. Parameters of operators can be altered to affect their functionality. TouchDesigner uses six families of operators, organized and color-coded according to their tasks (Picture 4). Operators of the same family can be connected together straight away. Connecting operators from different families requires a specific node to convert data. (TouchDesigner wiki 2018) New operators are added by double-clicking an empty area on the Network Editor and selecting the desired operator from the menu that pops up.



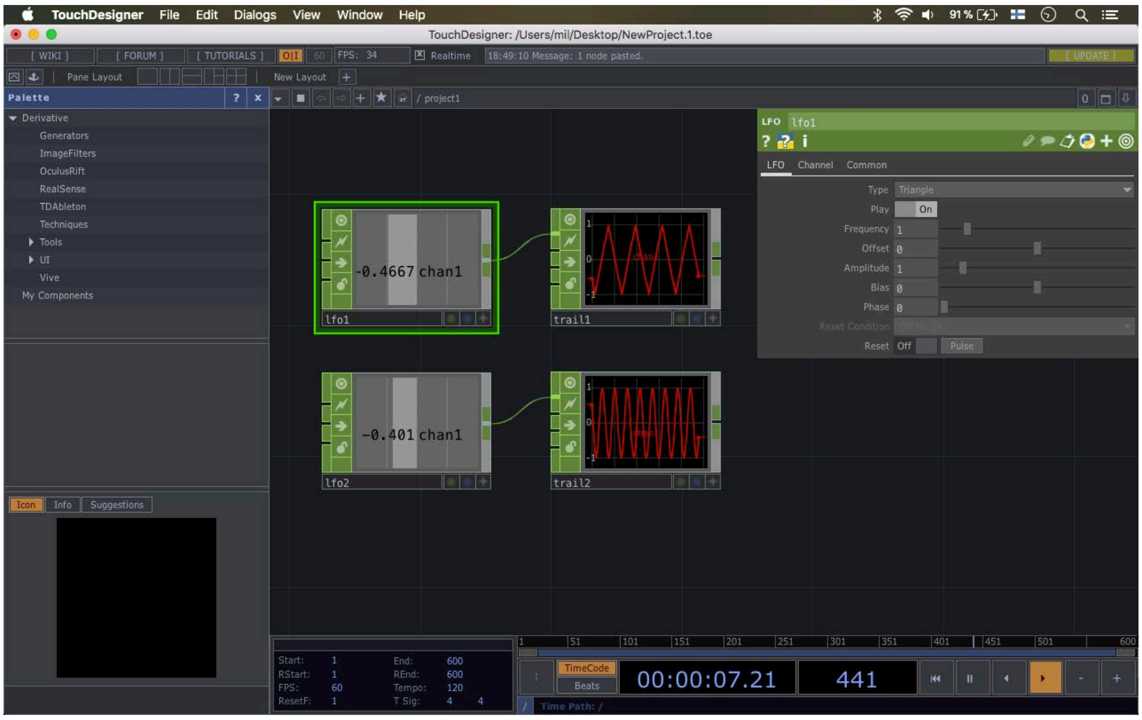
PICTURE 4. The six operator families represented from left to right: Component, texture operator, channel operator, surface operator, data operator and material operator (Touch-Designer wiki, 2018)

6.3.1 Components

Component nodes can contain networks of nodes inside them. Complicated networks can be simplified using Components such as Containers or Bases. Component inputs and outputs are determined by the operators contained within them. Thus, a component could have inputs and outputs for many different families of operators. A Component node displays its output on the preview window on the node by default.

6.3.2 Channel operators

Color-coded in green, channel operators handle motion, audio, animation and control signals. Channel operators or CHOPs output data as raw samples, meaning arrays of numbers. An example of a CHOP would be an LFO, or low-frequency oscillator, which puts out a numerical value that oscillates between defined numbers overtime.



18:49:10 Message: 1 node pasted.

Type Triangle
Play On
Frequency 1
Offset 0
Amplitude 1
Bias 0
Phase 0
Reset Off

Start: 1 End: 600
RStart: 1 REnd: 600
FPS: 60 Tempo: 120
ResetF: 1 T Sig: 4 4

PICTURE 5. Screenshot of two LFO CHOPs outputting their numerical data as different waveform types at different frequencies. (Laskujärvi, 2018)

6.3.3 Texture operators

Texture operators, or TOPs, handle image manipulations, such as displacement, scaling, compositing and cropping. 3D objects are rendered to images using a “Render” TOP. With TouchDesigner Non-Commercial license, the maximum output resolution is limited to 1280 by 1280 pixels. TOP nodes can be identified by their purple color.

6.3.4 Surface operators

Surface operators are used to generate, import and manipulate 3D objects inside Touch-Designer. SOPs can also be used for creating particle effects. A few primitive shapes are available in TouchDesigner by default, such as a torus, a cube and a polygonal grid surface. These objects can then be subdivided and deformed in real-time using Surface operators. TouchDesigner lacks an efficient solution for hands-on mesh editing such of that in Blender. Likewise, TouchDesigner by default has no physics engine to simulate rigid body physics events or collisions of 3D objects. Precise modeling or animation work should therefore be done using another software beforehand. SOPs are blue in color.

6.3.5 Material operators

Material operators work together with the SOPs to provide shading for the 3D objects being rendered. Physically-based materials, wireframe materials and Phong materials are available among others. MATs have yellow color.

6.3.6 Data operators

Data operators, shortened as DATs, can hold text data like tables or scripts. DATs are pink in color.

6.3.7 Controller connectivity

External controllers such as MIDI keyboards, game controllers and VR headsets can be connected to a computer running TouchDesigner and their input data can be used in a TouchDesigner network with specific CHOPs. Setting up a MIDI controller, one has to first set their connected device up in the MIDI Device Mapper dialog. The “MIDI In” CHOP can then receive incoming MIDI data, so that values can be adjusted by whatever human interface offered by the controller. Usually MIDI controllers feature buttons, potentiometers and sliders, offering a tactile way of controlling TouchDesigner creations.

PICTURE 6. Akai Professional LPD8 -USB MIDI controller (Laskujärvi, 2018)

6.4 Perform mode

In addition to using the Network Editor, TouchDesigner applications can be operated in "Perform mode". The default editor view for TouchDesigner is referred to as "Designer mode". Perform mode uses less of the computer's resources, only displaying a control panel configured by the user. The user can create a control panel for their Perform mode by using Component operators found under the type "Panel".

7 BLENDER GAME ENGINE

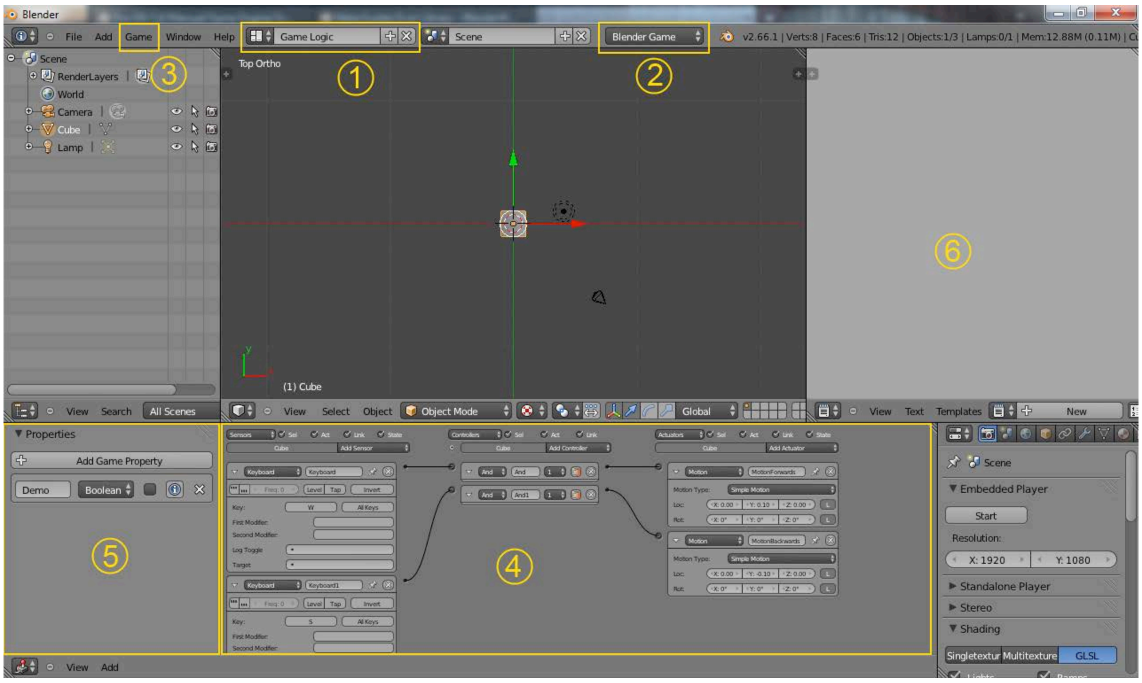
7.1 Installation & license

The Blender Game Engine comes bundled with Blender which is available at https://www.blender.org/download/. Blender is available for Linux, macOS and Microsoft Windows. On Linux and Windows, the user can choose between 32-bit and 64-bit versions. On Windows, the user can choose to have a “portable” zip-file download which is unpacked and works without an installation procedure.

Blender itself is “free software”, released under the GNU General Public License. The file output from Blender (.blend -files) is considered property of the creator, who can freely license or monetize on their creation. These are not covered by the GPL license. However, when making a standalone program out of a Blender Game Engine -based game, the user has created a GPL-licensed work. (Blender 2018)

7.2 User interface

The Blender Game Engine is an individual rendering engine within Blender. Setting up BGE as the renderer is done via the drop-down menu (Picture 7, highlight 2). A preset layout to optimize the Blender interface for game development can be chosen from the screen layout menu. This layout exhibits the Outliner for managing assets on the left, the 3D View window in the center of the screen, Text Editor on the right for scripting and the Game Logic panel in the lower third of the screen. On the bottom right is the Properties panel, not to be confused with the properties tab within the Game Logic window.



PICTURE 7. Blender user interface, with the view menu (1), renderer menu (2), game menu (3), Game Logic panel (4), Game Properties tab (5) and Text Editor panel (6) highlighted (Blender Manual, 2018)

7.3 Game Logic

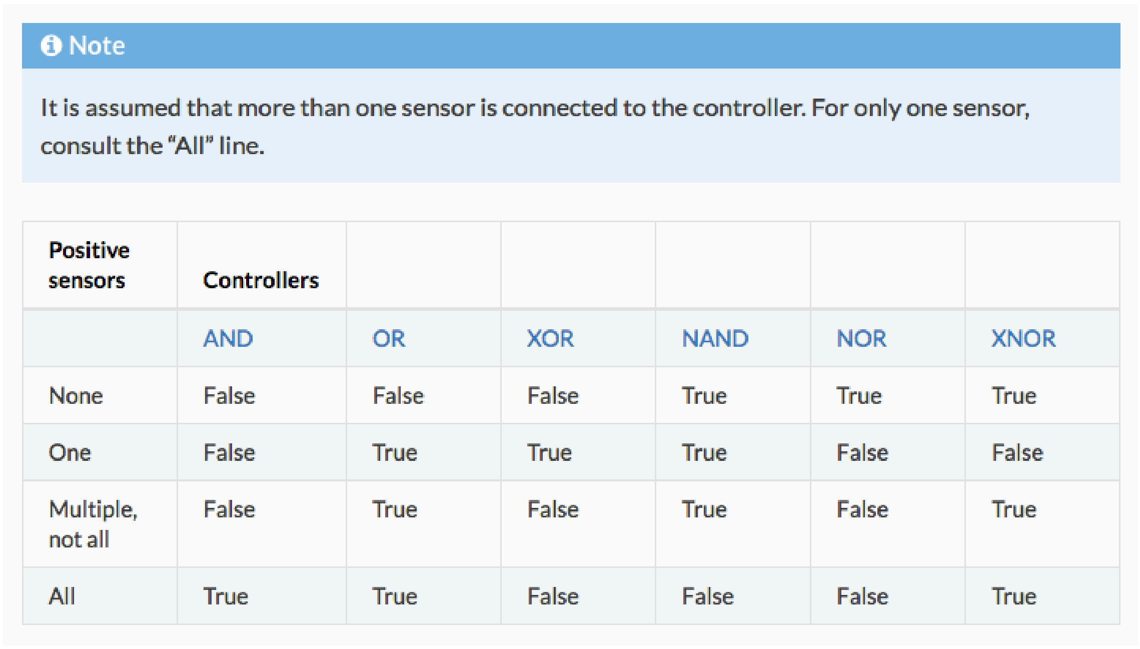
The Game Logic panel in Blender hosts the visual tools for scripting for each game object. Game objects in Blender can be any 3D objects in a scene. Three different types of “logic bricks” can be added and connected within the Game Logic panel. The Game Logic panel also hosts the Properties tab, which can contain variables to store data. These variables are referred to in Blender as “properties”.

7.3.1 Sensors

The Game Logic panel is organized in three columns, with the first one from the left being reserved for sensors. Sensors are event listeners, standing by while the game is running and sending pulses to the game logic once they are triggered. An example would be a collision sensor, which detects when an object gets hit by another object and sends a pulse to the rest of the game logic chain on the frame of the collision event.

7.3.2 Controllers

Controllers are logic nodes which take the sensor pulses as their inputs and compare these according to their settings before sending a pulse forward in the game logic. For example, an AND controller requires all of its sensor inputs to be active simultaneously, before passing a signal forward for the game logic to continue. Controllers can also host Python scripts.



Note

It is assumed that more than one sensor is connected to the controller. For only one sensor, consult the "All" line.

Positive sensors	**Controllers**					
	AND	OR	XOR	NAND	NOR	XNOR
None	False	False	False	True	True	True
One	False	True	True	True	False	False
Multiple, not all	False	True	False	True	False	True
All	True	True	False	False	False	True

PICTURE 8. A screenshot of a table depicting the different controller states in the Blender Game Engine (Laskujärvi, 2018)

7.3.3 Actuators

Actuators execute actions based on the game logic input they get from controllers. This can be movement, animations or toggling object visibility, to name a few. For example, the Message actuator can send a message within the game while its running, and a Message sensor somewhere else in the game can catch this message, setting an event in motion.

7.3.4 Properties

Properties can be used for storing and accessing data, much like variables in other programming languages. Each game object can have a set of properties assigned to it. The types of properties available include integer values, floating point values, boolean values, strings of characters or timers.

7.3.5 External controller connectivity

USB game controllers are supported by BGE with the Joystick sensor. However, no MIDI controller support is available for the user within the list of sensors.

7.4 Properties panel

The Properties panel is used for setting up additional settings in BGE. Largely remaining the same as when using other rendering engines, the Properties panel hosts settings for object materials, textures, physics, world and rendering settings among others.

7.5 Running the game engine

From the Properties panel Render tab, a Blender game can be run either embedded within Blender in the 3D View panel or in a window with the standalone player. While running a game, the Blender interface itself becomes unresponsive until the game is stopped.

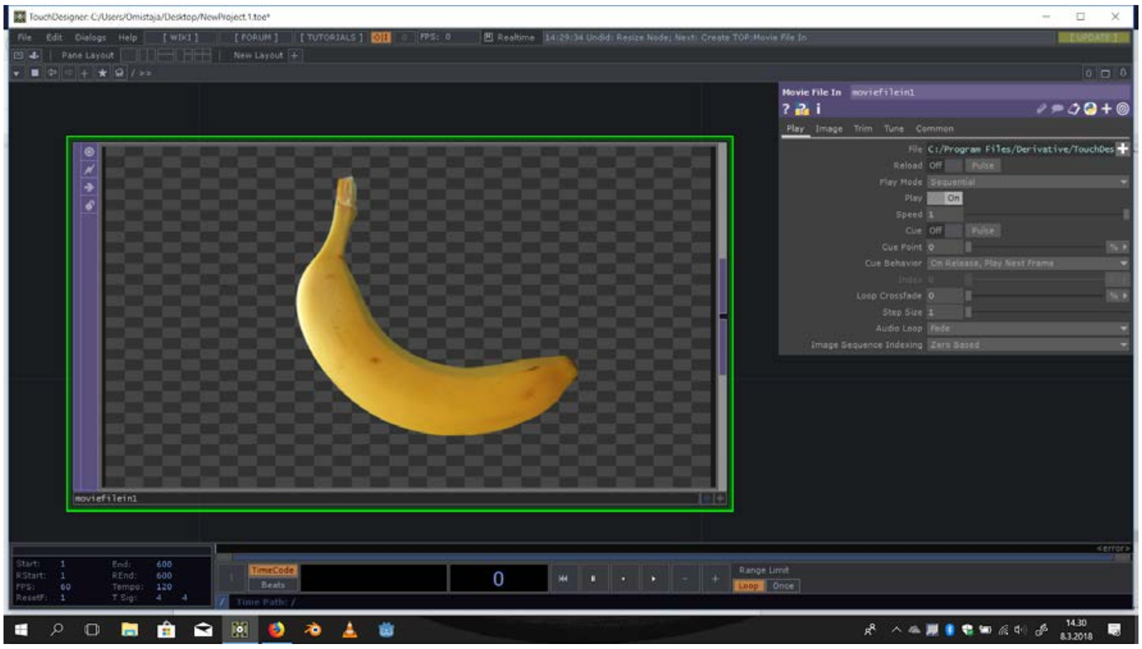
8 BUILDING MUSIC VISUALIZATION APPLICATIONS USING TOUCHDESIGNER AND BLENDER GAME ENGINE

8.1 Using images and setting up rhythmic input

The main focus in making VJ applications with BGE and TouchDesigner for this thesis was that the user should be able to express rhythm in a direct manner, such as with musical instruments. This means having buttons which trigger visual events instantaneously, lasting for various amounts of time. Topmost was also the idea of the user being able to "let go" of the controls after they had entered a rhythmic sequence of key presses which they liked.

8.1.1 TouchDesigner

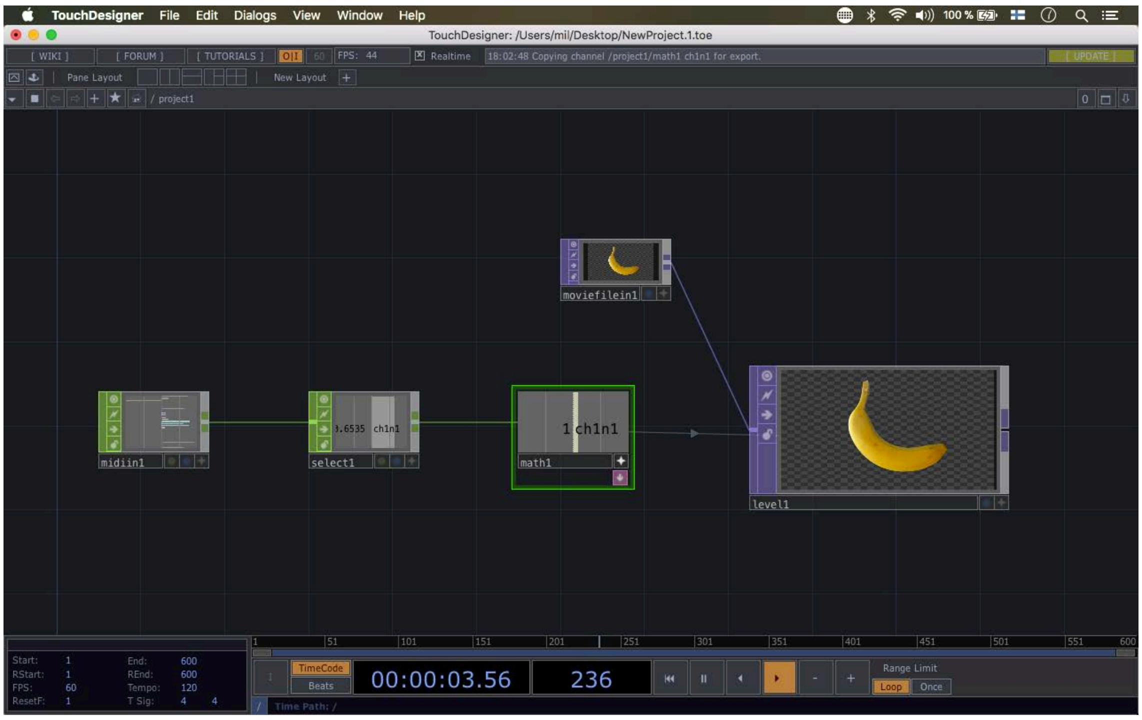
Bringing an image to TouchDesigner happens by using a "Movie File In" TOP, which has the file path for the image or video as its first property. Clicking the plus icon next to the file path opens up a graphical file search dialog (Picture 9.).


Movie File In moviefilein1

File C:/Program Files/Derivative/TouchDes
Reload Off Pulse
Play Mode Sequential
Play On
Speed 1
Cue Off Pulse
Cue Point 0
Cue Behavior On Release, Play Next Frame
Index 0
Loop Crossfade 0
Step Size 1
Audio Loop Fade
Image Sequence Indexing Zero Based

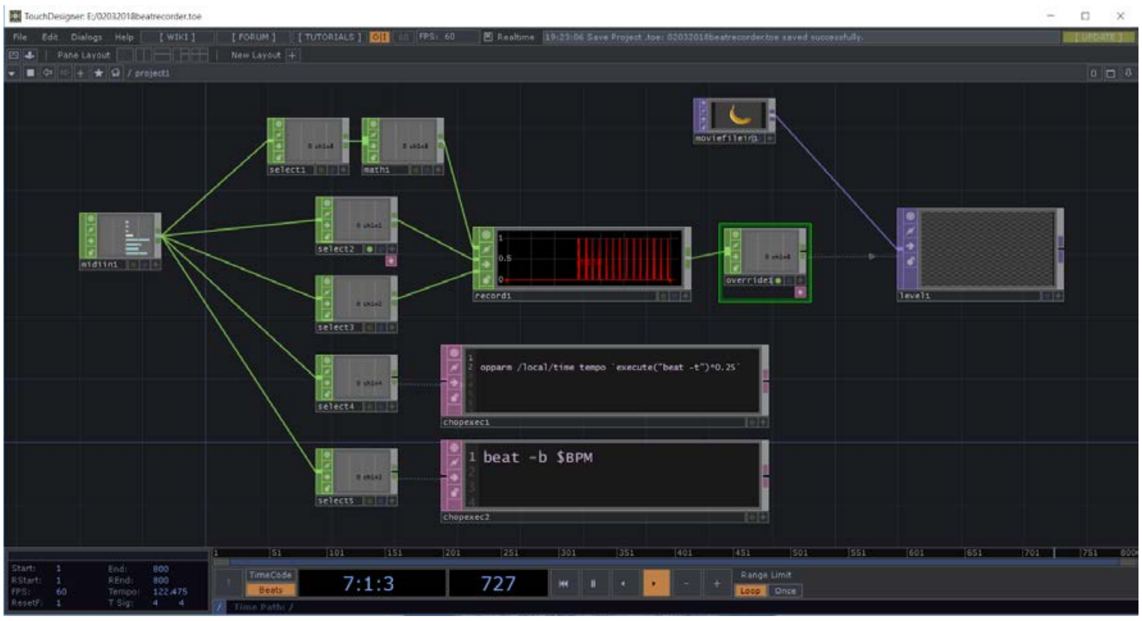
PICTURE 9. The "Movie File In" TOP in closer inspection with its parameter window to the right (Laskujärvi, 2018)

In TouchDesigner the ease of implementing a MIDI controller as a control surface made it an obvious choice for a user input device. For this practical test, the Akai LPD8 was used. In practice, making an Akai LPD8 MIDI controller trigger visual elements first requires a “Midi In” CHOP. Then the channel related to a particular button is picked from the midi input with a “Select” CHOP, outputting a floating point number value between 0 and 1 determined by the velocity at which the button is hit. This output value can be routed through a “Math” CHOP, rounding up the floating point value from between 0 to 1 to be either 0 or 1. Now this value can be exported to control an opacity value of a “Movie File In” TOP through a “Level” TOP, as illustrated in Picture 10.


TouchDesigner File Edit Dialogs View Window Help
TouchDesigner: /Users/mil/Desktop/NewProject.1.toe
18:02:48 Copying channel /project1/math1 chln1 for export.
moviefilein1
midiin1
select1
math1
level1
1 chln1
00:00:03.56
236

PICTURE 10. Controlling the opacity of an image with a MIDI control signal in TouchDesigner (Laskujärvi, 2018)

Making a sequencer based on user input is also relatively straightforward with TouchDesigner. Shown in Picture 11, MIDI inputs are used to record a sequence that the user creates while holding down one button and making a rhythmic pattern with another button. Another button resets this pattern. Additionally, beat detection is added with a tap tempo functionality to get an average tempo of the music playing. With the first button, the user can tap to the beat and get an estimate of the tempo of the music. A second button is used to reset time evaluation between taps.



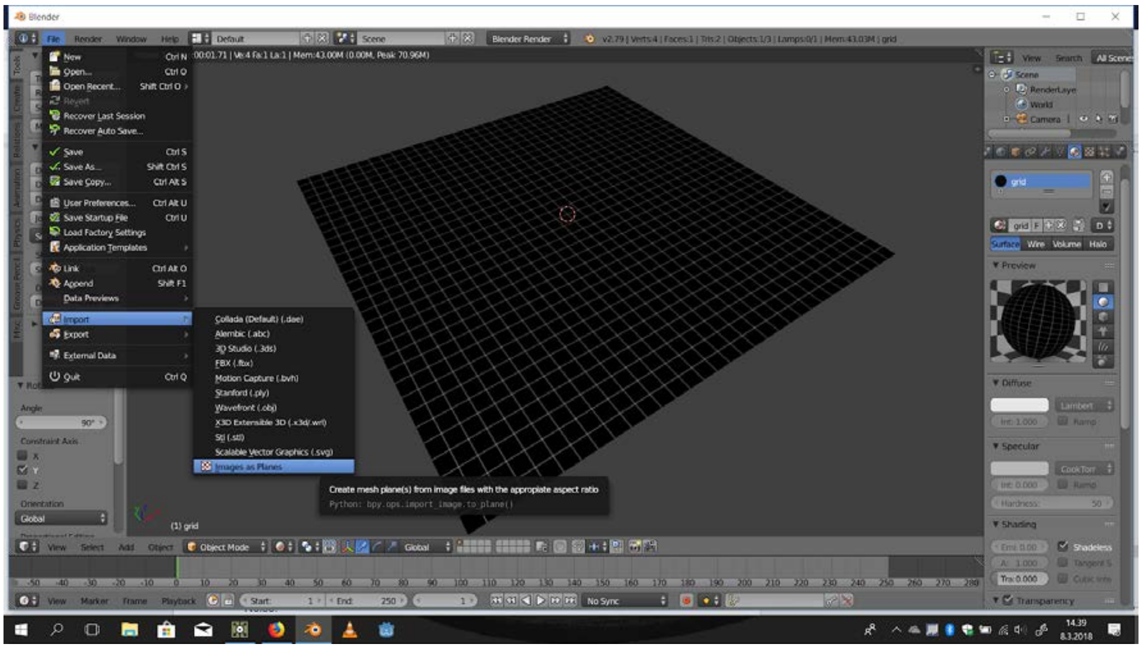
```

opparm /local/time tempo `execute("beat -t")*0.25`
```

```
beat -b $BPM
```

PICTURE 11. A sequencer for taking tap input from an external MIDI controller with the video file color level being controller by the incoming tap signals (Laskujärvi, 2018)

8.1.2 Blender Game Engine

With BGE, using image and video can be achieved by applying textures on 3D objects such as planes. Blender has an add-on for importing images as planes, otherwise the process requires UV unwrapping for the mesh, creating a material for it and then applying the image as a texture. Using video requires a bit more involvement as explained in the next chapter.



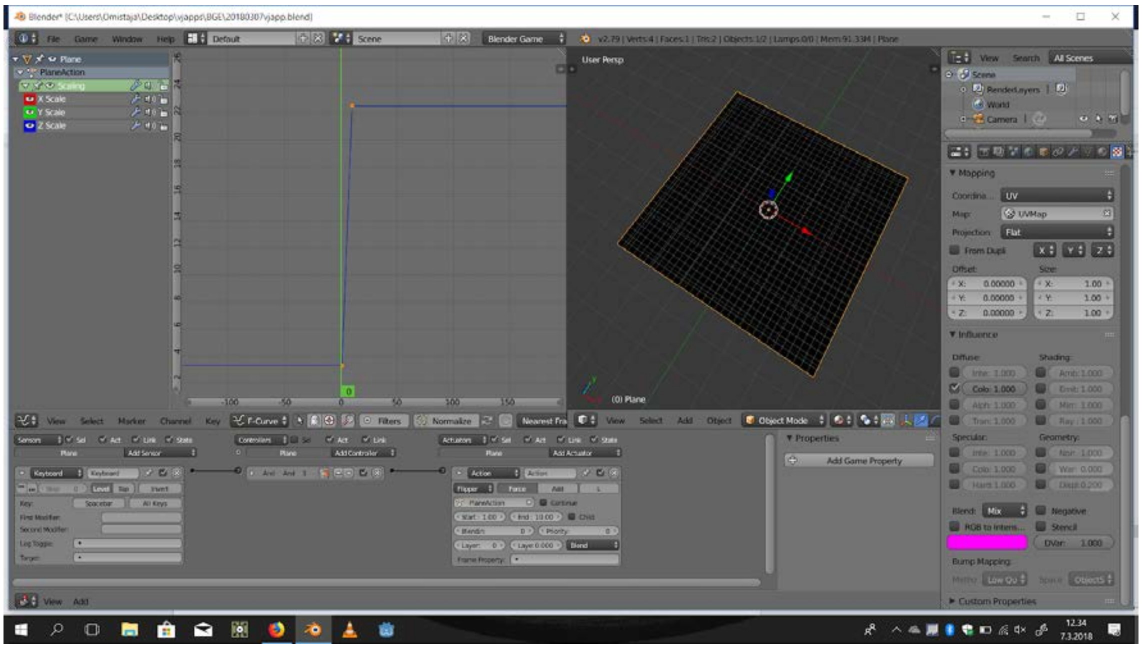
00:01.71 | Ve:4 Fa:1 La:1 | Mem:43.00M (0.00M, Peak 70.96M)

Create mesh plane(s) from image files with the appropriate aspect ratio

```
Python: bpy.ops.import_image.to_plane()
```

PICTURE 12. Importing an image as a plane using the ”Images as Planes” addon which is installed into the file import menu of Blender (Laskujärvi, 2018)

In the Blender Game Engine a rhythmic element could be implemented by key framing an animation for an object and then controlling that object with game logic. In the image below, two key frames are set for the plane object with an image texture of a grid. The plane is set to scale up along its two axes of y and x from the first frame until frame 10. This animation is then controlled with a keyboard sensor triggering an Action actuator once the spacebar is pressed. In this case, the Action actuator mode is set to Flipper, meaning it “rewinds” back to the first frame of the animation once the keyboard sensor is not active.

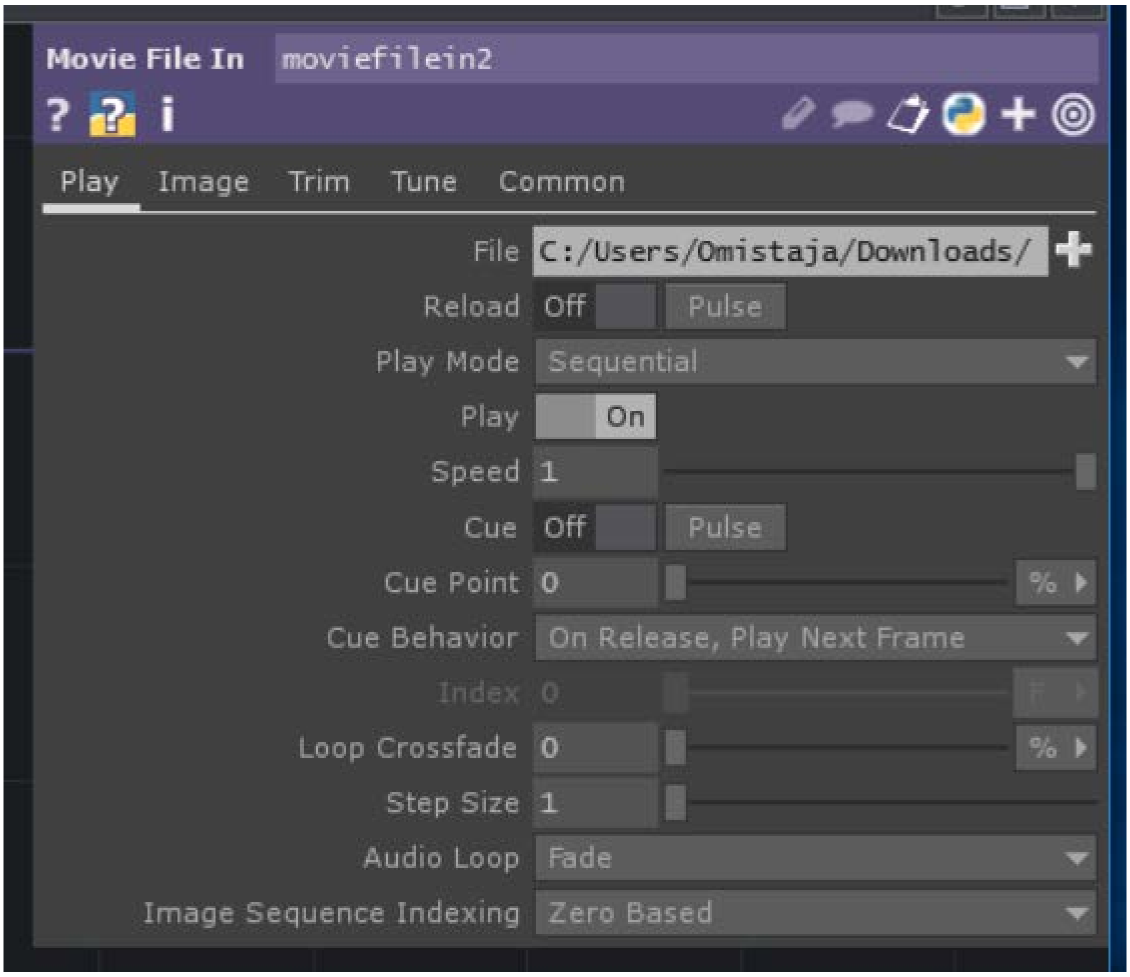


PICTURE 13. Setting up a keyboard sensor trigger for driving an animation real-time in the Blender Game Engine (Laskujärvi, 2018)

8.2 Displaying video

8.2.1 TouchDesigner

In TouchDesigner, the “Movie File In” TOP can be used to import both image and video files. Many properties for the video can be adjusted in the parameters view, such as playback speed, cue points along the video and trimming the start and end points of the video. “Transform” and “Level” TOPs were then added to the pipeline following the imported video file to adjust size and color respectively.



Movie File In moviefilein2

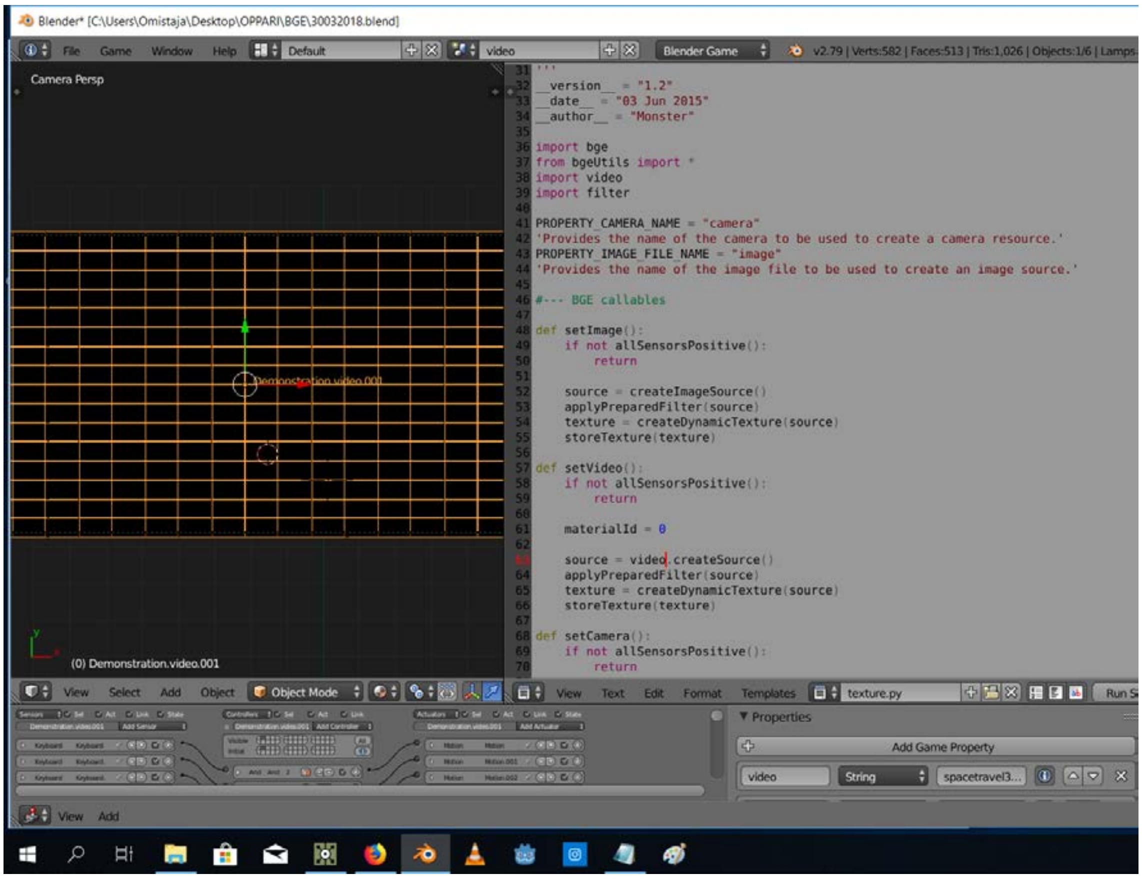
Play Image Trim Tune Common

File C:/Users/Omistaja/Downloads/
Reload Off Pulse
Play Mode Sequential
Play On
Speed 1
Cue Off Pulse
Cue Point 0 %
Cue Behavior On Release, Play Next Frame
Index 0
Loop Crossfade 0 %
Step Size 1
Audio Loop Fade
Image Sequence Indexing Zero Based

PICTURE 14. Parameters for the “Movie File In” TOP (Laskujärvi, 2018)

8.2.2 Blender Game Engine

In the Blender Game Engine, importing video files requires a few Python scripts and a logic setup that loads a video file onto a material of a 3D object. The original setup was downloaded from blenderartists.org website and is authored by Monster. A major problem with displaying video in the Blender Game Engine is the resolution. A 1080p full high-definition video would lower the framerate in the game engine to being unacceptably slow. After trial and error, a resolution of 640 by 360 pixels for the video kept Blender Game Engine within an acceptable average frame rate of 60 fps. However, this kind of low-definition leaves much to be desired in 2018. Besides this, switching from one video to another on the same object in real-time is impossible.



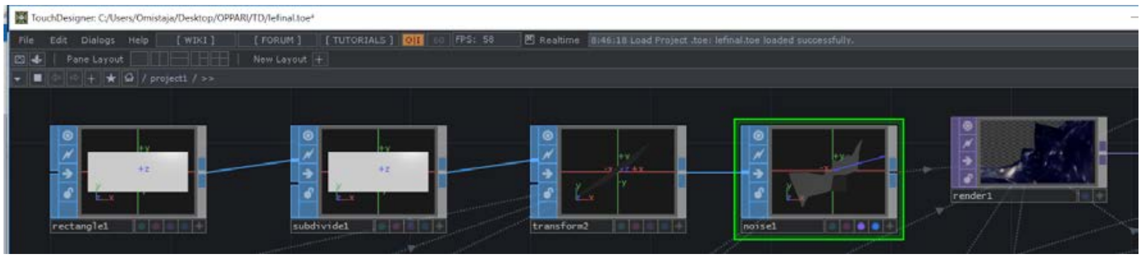
```
'''
__version__ = "1.2"
__date__ = "03 Jun 2015"
__author__ = "Monster"

import bge
from bgeUtils import *
import video
import filter

PROPERTY_CAMERA_NAME = "camera"
'Provides the name of the camera to be used to create a camera resource.'
PROPERTY_IMAGE_FILE_NAME = "image"
'Provides the name of the image file to be used to create an image source.'

#--- BGE callables

def setImage():
    if not allSensorsPositive():
        return

    source = createImageSource()
    applyPreparedFilter(source)
    texture = createDynamicTexture(source)
    storeTexture(texture)

def setVideo():
    if not allSensorsPositive():
        return

    materialId = 0

    source = video.createSource()
    applyPreparedFilter(source)
    texture = createDynamicTexture(source)
    storeTexture(texture)

def setCamera():
    if not allSensorsPositive():
        return
```

PICTURE 15. 3D plane object on the left with Monster’s applied Python script for displaying video in the Blender Game Engine partially shown on the right (Laskujärvi, 2018)

8.3 Manipulating 3D objects

8.3.1 TouchDesigner

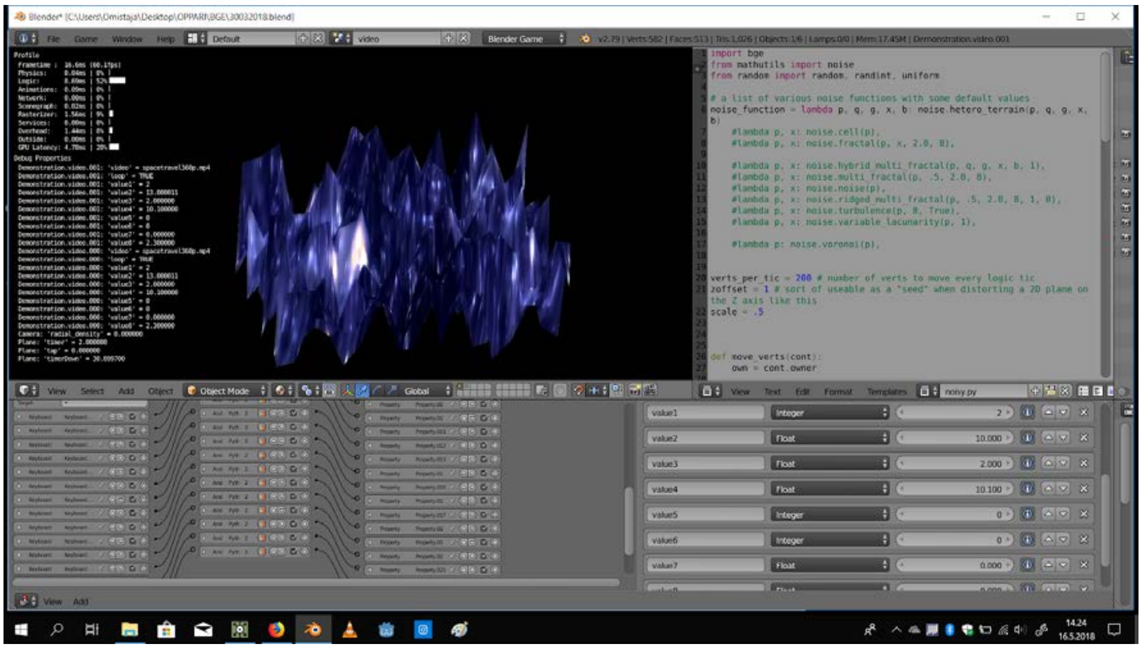
TouchDesigner offers SOP or surface operator nodes to manipulate geometry. In the application built for this thesis a rectangle object was first subdivided to make noise effects more evident. Next, a “Transform” SOP was added to enable the user to control the 3D rectangle with a MIDI controller. Lastly, “Noise” SOP was added to the end of the pipeline before rendering. Parameters on the “Noise” SOP received inputs from the MIDI controller, so the user can adjust the depth and roughness of the noise effect.



PICTURE 16. Subdividing a 3D rectangle object, scaling it and applying noise in Touch-Designer (Laskujärvi, 2018)

8.3.2 Blender Game Engine

With Blender, a plane was added as the 3D surface to render the video content on. This plane was then subdivided in Edit Mode so that it would react to noise properly. A modified noise-generating Python script originally found from the Blender Stack Exchange website was applied on the plane object and its values were connected to properties so that driving those values manually with the keyboard would be possible. Before discovering this script, using bones and animated noise from the Graph Editor was the first option to make the surface vertices appear moving randomly.


Blender* [C:\Users\Omistaja\Desktop\OPPARI\BGE\30032018.blend]
File
Game
Window
Help
Default
video
Blender Game
v2.79 | Verts:582 | Faces:513 | Tris:1,026 | Objects:1/6 | Lamps:0/0 | Mem:17.45M | Demonstration.video.001
View
Select
Add
Object
Object Mode
Global
View
Text
Edit
Format
Templates
noisy.py
14.24
16.5.2018

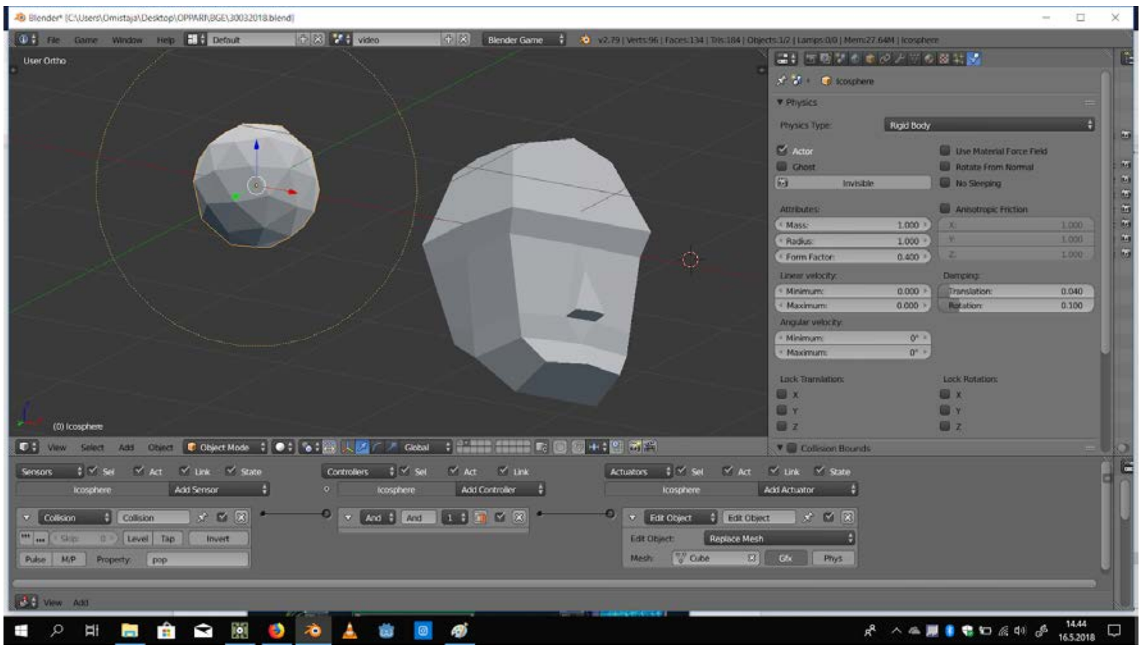
```
Profile
 Frametime :   16.6ms (60.1fps)
 Physics:      0.04ms | 0%
 Logic:        8.69ms | 52%
 Animations:   0.09ms | 0%
 Network:      0.00ms | 0%
 Scenegraph:   0.02ms | 0%
 Rasterizer:   1.56ms | 9%
 Services:     0.00ms | 0%
 Overhead:     1.44ms | 8%
 Outside:      0.00ms | 0%
 GPU Latency:  4.78ms | 28%
Debug Properties
 Demonstration.video.001: 'video' = spacetravel360p.mp4
 Demonstration.video.001: 'loop' = TRUE
 Demonstration.video.001: 'value1' = 2
 Demonstration.video.001: 'value2' = 13.000011
 Demonstration.video.001: 'value3' = 2.000000
 Demonstration.video.001: 'value4' = 10.100000
 Demonstration.video.001: 'value5' = 0
 Demonstration.video.001: 'value6' = 0
 Demonstration.video.001: 'value7' = 0.000000
 Demonstration.video.001: 'value8' = 2.300000
 Demonstration.video.000: 'video' = spacetravel360p.mp4
 Demonstration.video.000: 'loop' = TRUE
 Demonstration.video.000: 'value1' = 2
 Demonstration.video.000: 'value2' = 13.000011
 Demonstration.video.000: 'value3' = 2.000000
 Demonstration.video.000: 'value4' = 10.100000
 Demonstration.video.000: 'value5' = 0
 Demonstration.video.000: 'value6' = 0
 Demonstration.video.000: 'value7' = 0.000000
 Demonstration.video.000: 'value8' = 2.300000
 Camera: 'radial_density' = 0.000000
 Plane: 'timer' = 2.000000
 Plane: 'tap' = 0.000000
 Plane: 'timerDown' = 30.099700
```

```python
import bge
from mathutils import noise
from random import random, randint, uniform

# a list of various noise functions with some default values
noise_function = lambda p, q, g, x, b: noise.hetero_terrain(p, q, g, x, b)
    #lambda p, x: noise.cell(p),
    #lambda p, x: noise.fractal(p, x, 2.0, 8),

    #lambda p, x: noise.hybrid_multi_fractal(p, q, g, x, b, 1),
    #lambda p, x: noise.multi_fractal(p, .5, 2.0, 8),
    #lambda p, x: noise.noise(p),
    #lambda p, x: noise.ridged_multi_fractal(p, .5, 2.0, 8, 1, 0),
    #lambda p, x: noise.turbulence(p, 8, True),
    #lambda p, x: noise.variable_lacunarity(p, 1),

    #lambda p: noise.voronoi(p),


verts_per_tic = 200 # number of verts to move every logic tic
zoffset = 1 # sort of useable as a "seed" when distorting a 2D plane on the Z axis like this
scale = .5



def move_verts(cont):
    own = cont.owner
```

value1	Integer	2
value2	Float	10.000
value3	Float	2.000
value4	Float	10.100
value5	Integer	0
value6	Integer	0
value7	Float	0.000

PICTURE 17. Screenshot of Blender Game Engine running the noise script applied on the 3D plane object. Python script partly visible on the right, with the control parameters in the below panel. (Laskujärvi, 2018)

An animated Empty object with a trigger to emit objects was added into the scene to add more random elements with Rigid Body physics objects. The Empty object is not visible in the scene while the program is running but can have size, orientation and location values as well as hold and execute logic. In this case, the Rigid Body object emitted is an icosphere which changes its graphical appearance into that of another object once it collides with the video plane surface.



Physics Type: Rigid Body
Actor ✓
Ghost
Invisible
Use Material Force Field
Rotate From Normal
No Sleeping
Anisotropic Friction
Attributes:
Mass: 1.000
Radius: 1.000
Form Factor: 0.400
X: 1.000
Y: 1.000
Z: 1.000
Linear velocity:
Minimum: 0.000
Maximum: 0.000
Damping:
Translation: 0.040
Rotation: 0.100
Angular velocity:
Minimum: 0°
Maximum: 0°
Lock Translation: X Y Z
Lock Rotation: X Y Z

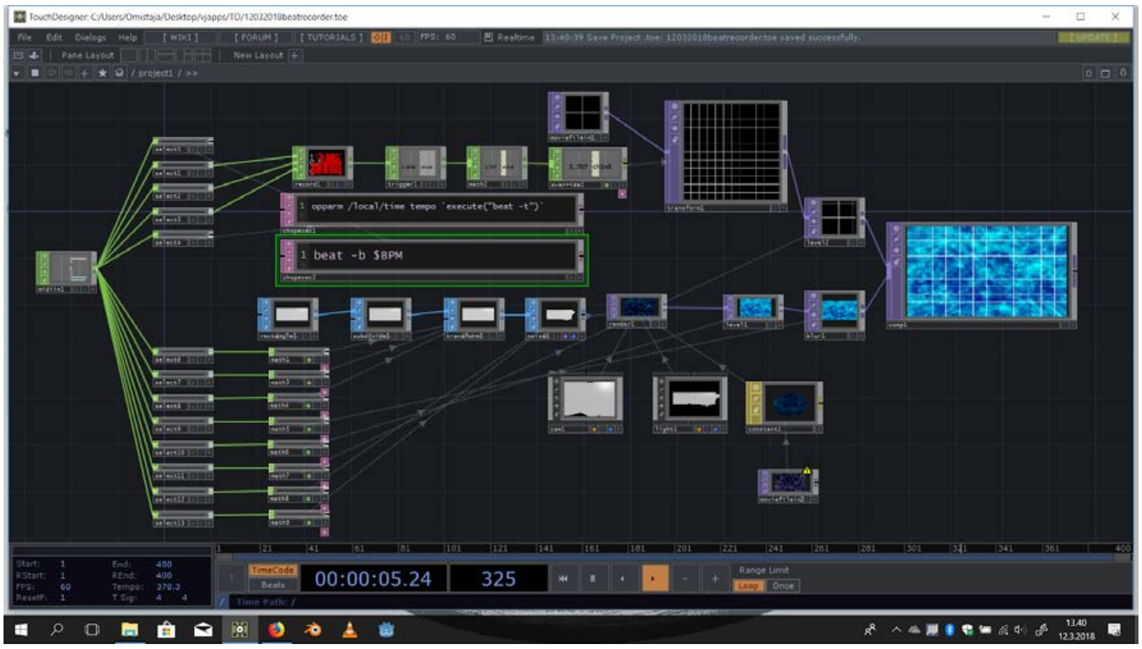
Sensors: Collision | Collision | Level | Tap | Invert | Pulse | M/P | Property: pop
Controllers: And | And | 1
Actuators: Edit Object | Edit Object | Edit Object: Replace Mesh | Mesh: Cube | Gfx | Phys

PICTURE 18. The collision icosphere object on the left and a human head object on the right, which is the appearance of the icosphere once the game is run and it collides with the video plane surface. Bottom third of the screen shows the Logic nodes for changing an object’s graphical appearance to that of another mesh. Physics properties for the icosphere visible on the far right. (Laskujärvi, 2018)

8.4 Applying 2D effects

8.4.1 TouchDesigner

Two-dimensional effects applied in this TouchDesigner project included transformations, color level adjustments, blur effects and a composition node. The grid layer scale transformations are driven by the looping pulses recorded from the MIDI input. The grid brightness is then adjusted in the following “Level” TOP by using a knob on the MIDI controller. The grid is finally connected to the “Composite” TOP where its added to the rendered 3D plane object. Like the grid layer, the 3D plane render goes through level adjustments for fading in and out with the MIDI controller. Before going to the main composition, the image goes to a “Blur” TOP, where a blur effect can be applied using the MIDI controller.



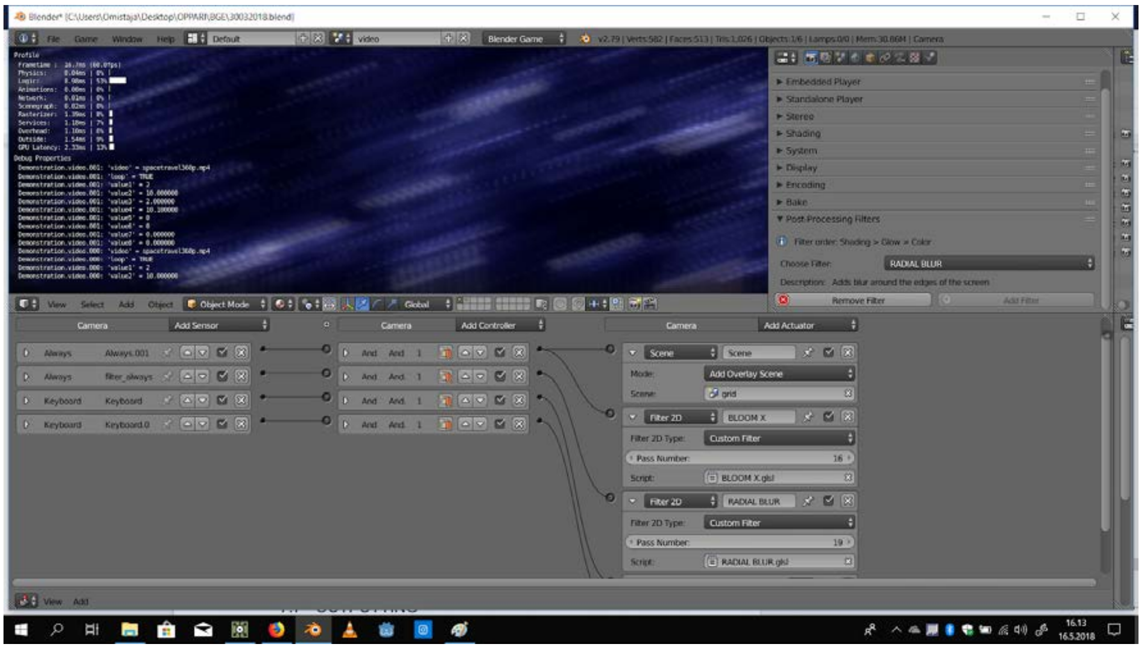
```
opparm /local/time tempo `execute("beat -t")`
```

```
beat -b $BPM
```

PICTURE 19. An image showing the final network of the VJ application in TouchDesigner (Laskujärvi, 2018)

8.4.2 Blender Game Engine

Applying 2D effects in the Blender Game Engine can be achieved by using OpenGL scripts that describe to the graphics processing unit of one’s computer how to render the output. Learning to program OpenGL scripts, or “shaders” as they are most often referred to as, can be a tedious task. Luckily an add-on called “Post-processing filters” has been made available by Tim Crellin for the Blender Game Engine which gives the user a drop-down menu of several shaders to choose from and then apply to the program. These applied scripts are then automatically added to the Logic. These shader scripts can also be easily modified by the user in the Text Editor view.



```
Profile
 Frametime :   16.7ms (60.0fps)
 Physics:      0.04ms | 0% |
 Logic:        8.98ms | 53%
 Animations:   0.00ms | 0% |
 Network:      0.01ms | 0% |
 Scenegraph:   0.02ms | 0% |
 Rasterizer:   1.39ms | 8%
 Services:     1.18ms | 7%
 Overhead:     1.10ms | 6%
 Outside:      1.54ms | 9%
 GPU Latency:  2.33ms | 13%
Debug Properties
 Demonstration.video.001: 'video' = spacetravel360p.mp4
 Demonstration.video.001: 'loop' = TRUE
 Demonstration.video.001: 'value1' = 2
 Demonstration.video.001: 'value2' = 10.000000
 Demonstration.video.001: 'value3' = 2.000000
 Demonstration.video.001: 'value4' = 10.100000
 Demonstration.video.001: 'value5' = 0
 Demonstration.video.001: 'value6' = 0
 Demonstration.video.001: 'value7' = 0.000000
 Demonstration.video.001: 'value8' = 0.000000
 Demonstration.video.000: 'video' = spacetravel360p.mp4
 Demonstration.video.000: 'loop' = TRUE
 Demonstration.video.000: 'value1' = 2
 Demonstration.video.000: 'value2' = 10.000000
```

Filter order: Shading > Glow > Color
Choose Filter: RADIAL BLUR
Description: Adds blur around the edges of the screen

Sensors: Always – Always.001; Always – filter_always; Keyboard – Keyboard; Keyboard – Keyboard.0

Controllers: And – And 1 (×4)

Scene – Scene
Mode: Add Overlay Scene
Scene: grid

Filter 2D – BLOOM X
Filter 2D Type: Custom Filter
Pass Number: 16
Script: BLOOM X.glsl

Filter 2D – RADIAL BLUR
Filter 2D Type: Custom Filter
Pass Number: 19
Script: RADIAL BLUR.glsl

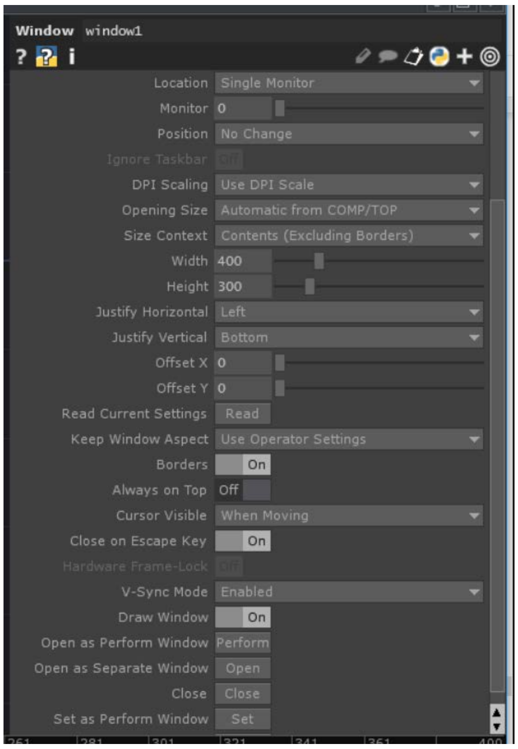
16.13
16.5.2018

PICTURE 20. A blur effect applied to the final render. On the right, the menu for choosing, applying and deleting shaders that come with the “Post-processing filters” -addon. Bottom half: Logic view with the shader nodes. (Laskujärvi, 2018)

8.5 Outputting

8.5.1 TouchDesigner

In the TouchDesigner project for this thesis an output was added to display the final render on a separate screen using the “Window” Component node. Any TOP operator can be dragged and dropped over the “Window” COMP, determining the output. The user can then make changes to the network on their primary display while the output is displayed on the other screen. Other useful options for output window adjustments include resizing to a custom resolution and ratio as well as choosing not to draw any borders for the window.



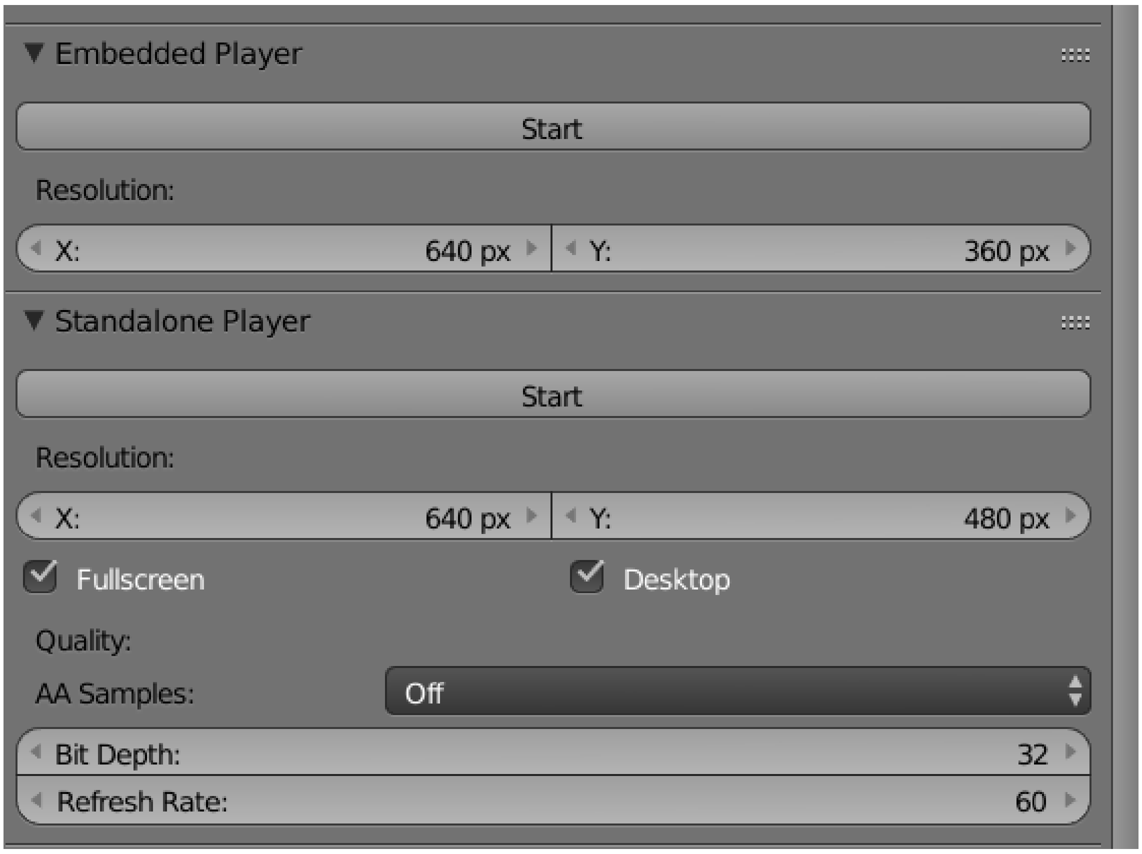
Window window1

Location	Single Monitor
Monitor	0
Position	No Change
Ignore Taskbar	Off
DPI Scaling	Use DPI Scale
Opening Size	Automatic from COMP/TOP
Size Context	Contents (Excluding Borders)
Width	400
Height	300
Justify Horizontal	Left
Justify Vertical	Bottom
Offset X	0
Offset Y	0
Read Current Settings	Read
Keep Window Aspect	Use Operator Settings
Borders	On
Always on Top	Off
Cursor Visible	When Moving
Close on Escape Key	On
Hardware Frame-Lock	Off
V-Sync Mode	Enabled
Draw Window	On
Open as Perform Window	Perform
Open as Separate Window	Open
Close	Close
Set as Perform Window	Set

PICTURE 21. “Window” COMP’s parameter panel showing the different options for controlling external display setups. (Laskujärvi, 2018)

8.5.2 Blender Game Engine

The Blender Game Engine output for rendering happens from the render settings in the Properties panel. The user can choose to either run their application with the Embedded Player or a Standalone Player. The Standalone Player option automatically pops out a window of a defined resolution, or takes up the full screen if this tick box is selected in the render options. The Standalone Player full screen mode uses the main system display by default. Using an additional display in extended desktop mode is possible, however a workaround is required if one wants to run their Blender Game Engine application on a separate full screen window. The Embedded Player occupies the 3D View panel once it is run. In Blender, panels can be popped out into separate windows by holding down Shift and dragging on either of their marked corners with the left mouse button. This separate window can then be placed on the external display on the extended desktop. The key combination of Alt and F11 toggles windows into full screen mode and back. Then the application can be run using Embedded Player by pressing P while hovering the mouse over the external 3D View window. However, setting up the correct view can be tedious because of having to manually adjust the view within camera objects.


Embedded Player

Start

Resolution:

X: 640 px | Y: 360 px

Standalone Player

Start

Resolution:

X: 640 px | Y: 480 px

☑ Fullscreen ☑ Desktop

Quality:

AA Samples: Off

Bit Depth: 32

Refresh Rate: 60

PICTURE 22. Embedded Player and Standalone Player options in the rendering settings (Laskujärvi, 2018)

9 DISCUSSION

Blender and TouchDesigner are both very capable pieces of software for producing graphics and animations and running them in real-time. They have simple interaction design tools in a user environment with otherwise steep learning curves. However, these more in-depth tools like bone systems for animation in Blender or particle systems in TouchDesigner enable the user to tweak everything just right thanks to their complexity.

Blender was originally oriented as a modelling and animation tool for film production industry. The greatest part of making interactive visualization applications derives from this as well: It is very simple and easy to change something down the line even if the user is already building logic for their application. They can go back to the drawing board with modelling and animation whenever they like and retain their control scheme already built for the interactions.

In TouchDesigner, building a network of nodes that can be easily adjusted later is possible, but changing details in 3D models or their animations may require the use of an external software, such as Blender. Blender falls short with the more complex logic patterns, where the display space runs out or gets messy very quickly. Therefore, using both Blender and TouchDesigner together could be one fruitful way of benefiting from some of the better possibilities offered by these software. For example, the Blender Game Engine render output could be captured and brought into TouchDesigner, with the more advanced physics and collision detection available while utilizing TouchDesigner’s effective MIDI controller integration.

Considering the future, TouchDesigner currently is the safer bet to get into doing real-time interactive graphics processing. This is because the Blender Game Engine has already been removed from the upcoming version 2.8 of Blender.

REFERENCES

Blender. 2018. Read 1.2.2018. https://www.blender.org/

Blender Manual. 2018. Read 12.2.2018. https://docs.blender.org/manual/

Derivative. 2018. Read 2.2.2018. http://www.derivative.ca/

Esaak, Shelley. 2017. Get the Definition of Contemporary Art. Read 4.3.2018. https://www.thoughtco.com/what-is-contemporary-art-182974

Gregory, Jason. 2014. Game Engine Architecture. Boca Raton: CRC Press.

Haines, Eric. 2002. Real-Time Rendering. Boca Raton: CRC Press.

Manovich, Lev. 2000. The Language of New Media. Read 3.3.2018. https://www.medi-amatic.net/en/page/9283

National Institute of Standards and Technology. 2017. Dictionary of Algorithms and Data Structures. Read 2.2.2018. https://xlinux.nist.gov/dads/

OpenGL Wiki. 2017. OpenGL Shading Language. Read. 1.1.2018. https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language

Paul, Christiane. 2008. New Media in the White Cube and Beyond: Curatorial Models for Digital Art. Read 1.2.2018. http://atc.berkeley.edu/201/readings/Christi-ane_Paul_Reading.pdf

Salvator, Dave. 2001. ExtremeTech 3D Pipeline Tutorial. Read. 3.3.2018. https://www.extremetech.com/computing/49076-extremetech-3d-pipeline-tutorial

Spinrad, Paul. 2005. The VJ book: Inspirations and practical advice for live visuals per-formance. Los Angeles: Feral House.

TouchDesigner 099 wiki. 2018. Read. 3.1.2018. https://docs.derivative.ca/