The Cinematic Sandbox: Utilizing Video Game Engines for Rapid Prototyping of Animated Films

By

Nicholas Garrett

A senior thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science
With Departmental Honors
Computer Science & Engineering
University of Washington
December 2010

Presentation of work given on ____________________

Thesis and presentation approved by ____________________

Date ____________________
# Table of Contents

Table of Contents .................................................................................................................. 2  
Abstract .................................................................................................................................. 3  
Introduction ............................................................................................................................. 4  
Other Work in Rapid Prototyping for Animated Films .............................................................. 6  
Goals for the Cinematic Sandbox ............................................................................................ 8  
Platform Considerations ......................................................................................................... 9  
Simplifying the Project and Required Features ..................................................................... 11  
Architecture ............................................................................................................................ 14  
  C# Classes .......................................................................................................................... 14  
    Setup ............................................................................................................................... 14  
    Sequence ......................................................................................................................... 15  
    Shot .................................................................................................................................. 15  
    MayaStyleControl .......................................................................................................... 15  
    CameraInfo ..................................................................................................................... 16  
    Communicator ............................................................................................................... 16  
    Util .................................................................................................................................. 16  
  HTML, CSS, and JavaScript ................................................................................................. 17  
  Python Script ....................................................................................................................... 17  
Future Work ........................................................................................................................... 18  
  Additional Features ............................................................................................................. 18  
  Other 3D Applications Besides Maya ................................................................................ 19  
  User Interface Improvements ............................................................................................ 19  
  Testing Improvements ........................................................................................................ 20  
  Cross-Browser Compatibility Improvements ................................................................... 20  
  Code Improvements .......................................................................................................... 20  
  Documentation .................................................................................................................... 21  
    Beta Testing the Cinematic Sandbox for the Animation Capstone ................................. 21  
Conclusion ............................................................................................................................ 23  
Acknowledgements ............................................................................................................... 24  
Source Code ........................................................................................................................... 24  
References .............................................................................................................................. 25
Abstract

Feature length and short animated films take years to produce, and each new film raises different challenges. It is very expensive when any part of the animation process needs to be redone whether it is lighting, animation, or rendering. Therefore prototyping the motion and cinematography of the film early on is important to ensure that the story is communicated clearly and shots don't need to be changed once they are animated, lit, and rendered. Solutions to speeding up the process of animation have been approached in industry and in academia, but these solutions generally focus on prototyping individual shots, not an entire film. An approach to prototyping an entire film is possible, and I addressed this problem with a proof of concept model called the Cinematic Sandbox. I focused on building the core functionality necessary to prototype an entire film and left other features for future work in order to make the project goals realistic.
Introduction

The animation capstone at the University of Washington is a collaborative, year-long series of courses in which students learn to develop and produce animated films. The capstone culminates with all of the enrolled students working together on a single animated short that is screened at graduation. The courses are organized as a hybrid of academia and industry, and students need no prior knowledge of animation. Students learn to use animation software through homework assignments and demonstrations in lecture, and work on films using a production pipeline similar to industry. One of the biggest challenges the students face is actually finishing a polished short film in time for graduation. In the past, the screened film has been rough, and dedicated students have worked to polish the film over the summer and into the next school year. It would be very beneficial for the students in the capstone if the animation process could be simplified so that they could produce animation more rapidly and have a polished film to show at graduation.

Computer-generated movies, including short films, can take several years to produce. The entire process from the conception of the story to creating models, animating characters, rendering shots, and post production is fraught with artistic and technical challenges. One such challenge is the transition from two-dimensional storyboards that depict the story as an artist sees it, to a three-dimensional animatic that depicts the story as a very rough animated film.

Creating the first animatic usually reveals problems with the storyboards like perspectives and camera angles that don’t work in three dimensions, scale problems where particular models are too big or small relative to others, and logistical problems, like when a character needs to interact with objects that are too far away. Catching these issues as soon as possible during a production is important because they will inevitably affect the final film. If these issues are left uncorrected and persist through motion, lighting, and renders, then much of that work will have to be redone. In the film industry, this translates into
wasted money or cancelled films [1]; in the animation capstone at the University of Washington, this translates into films not getting finished on time, or a film that looks unpolished during its screening.

Fixing the problems discovered in the transition from two-dimensional to three-dimensional representations of a film can involve minor story rewrites, and therefore usually requires several iterations. Each of these iterations is time consuming and is a multi-step process. First, the problem must be fixed in Autodesk Maya, the software students use to create three dimensional models and animate them. Then, the fixed shot must be playblasted, which is the process of converting the shot data into a video file. The new video file must be re-rendered along with the rest of the film in Adobe Premier in order to see the changes to one shot in the context of the surrounding shots.

Each of these fixes has the potential to bring to light other problems or create new ones, so it is important to be able to iterate rapidly through each new version of the animatic. The animation capstone needs software that will expedite the process of creating animatics so that students will consistently be able to finish films by the end of spring quarter. The goal of this research project is to assess existing tools that expedite the process of creating animation, and create new tools if necessary in support of the students in the animation capstone.
Other Work in Rapid Prototyping for Animated Films

Due to the expensive and time consuming nature of animation, there are several groups, both industry and academic, that are interested in expediting the process. Motion Builder, a software package made by Autodesk, is designed for producing animation very quickly [2]. It is designed to combine multiple styles of animation such as motion capture, physics simulations, and hand animation fluidly into one character by blending the different inputs onto a single skeleton that controls a character. This approach is well-suited to quickly producing animation for individual shots, but does not address the transitions between shots.

Another company, Lightstorm Entertainment, worked with custom software to rapidly iterate on animation for production work on the film Avatar. The system Lightstorm used enabled the director to see motion captured animation converted into digital characters interacting with digital sets in real time [3]. This approach uses real time rendering to remove the previous capture/edit/render/view pipeline in the traditional motion capture process. Lightstorm’s system is well suited for rapid iteration on the motion capture process, but does not address building hand animation from scratch.

The GRAIL (Graphics and Imaging Laboratory) research group at the University of Washington has experimented with novel approaches to animating characters in order to make creating animation more efficient. GRAIL approaches animation by giving animators a simple way to create motion capture data, and applying that data to a skeleton in layers [4]. Animators manipulate a physical object that is tracked by a camera over time, and the motion of the object is mapped onto a specific part of the character skeleton. This allows animators to quickly achieve a very natural feel for their animations. The approach that GRAIL has adopted is effective for achieving natural character motion from scratch very quickly, but doesn't address cinematography.
These examples of related work for rapid prototyping generally address swiftly creating motion from scratch and quickly viewing motion capture data as it will appear in the final film. These approaches are helpful for rapid iteration on the motion for a single shot, but fail to address the layout and cinematographic issues caused when creating an animatic. Though the animation capstone can incorporate some of the techniques addressed by the related work in order to make more efficient use of students' time, the capstone will need some additional, custom software to address quickly iterating on an animatic. I built the Cinematic Sandbox to address the rapid prototyping needs of the animation capstone.
Goals for the Cinematic Sandbox

There were several overarching goals that the Cinematic Sandbox had to satisfy. The first was that it needed to make prototyping animated films fast. Students should not have to open up multiple software packages in order to view changes to the cinematography in the context of the surrounding shots. Ideally, students should see the changes they make reflected in real time in order to make prototyping films as fast as possible, and students need to be able to do this all in a single software package.

The second goal, was that all of the layout, cinematography, and motion work completed with the Cinematic Sandbox had to be compatible with Autodesk Maya. The animation capstone uses Maya for most of the production pipeline, so being able to convert the prototype to a file format that Maya understands was necessary for the project to integrate smoothly into the existing pipeline. Maya is also an industry standard software package for animation, so integration with Maya makes the Cinematic Sandbox potentially useful for broader applications than the animation capstone.

Platform independence was the third goal for the project. It would benefit students if professionals in the animation field that have maintained contact with the animation capstone are able to contribute to current films remotely. To this end, I needed to make sure that I didn't exclude potential users based on the system that they are running. In addition, I had to create the Cinematic Sandbox to be easily accessible over the internet for professionals to be able to contribute remotely.

The fourth goal of the project was that the Cinematic Sandbox needed be easy to use. Students and professionals that are familiar with Maya should feel comfortable with the software I developed without having to make too many adjustments. It needs to be intuitive and streamlined in order to facilitate rapid iteration.
Platform Considerations

In order to allow users to contribute to the animation capstone remotely, I built the Cinematic Sandbox as a web application. This fit perfectly with the Unity game engine which was used for real time rendering, because the game engine can be built as a flash application and embedded into a web page. Additionally, Unity supports directly importing scenes from Maya. In other words, Maya scenes can be viewed in the web browser and, since video game engines render in real time, users can see changes to the Maya scene instantly. There is a trade-off between rendering speed and visual quality, so the graphics shown in the browser are not as polished as real renders from Maya. However, since the goal of the Cinematic Sandbox is to help users prototype their films quickly, visual polish is not a requirement. Unfortunately, the Unity game engine is only supported on Windows and Mac, so Linux users are unable to use the Cinematic Sandbox. Since Linux accounts for the lowest market share of operating systems running on desktops and laptops, this is an acceptable sacrifice [5].

There are several alternative game engines that I could have chosen. In particular, the Elflight, GameSalad, and ShiVa game engines all offer the ability to build a game as a web application. However, the Elflight and ShiVa engines only support importing assets through the Collada interface [6][7], which is not as convenient as the direct importing offered by the Unity game engine. GameSalad is built with non-programmers in mind and only offers a visual, drag-and-drop approach to building the logic that will power a game [8]. Therefore, it doesn’t offer the same amount of flexibility and customization that a game engine that can be truly programmed offers.

Using game engines to build animation is not a new idea, it dates back to a 1992 video game called Stunt Island released by Disney Interactive. It featured the ability to record gameplay from a user-positioned camera and then edit the recorded clips into films. The first widely known use of a video game to tell a story was the 1996 short film Diary of a
Camper produced using a game called Quake. Using video games to produce films is called machinima, and has gained enough popularity to become mainstream [9]. The Cinematic Sandbox is similar to machinima in that I use a video game engine to produce the graphics, but different in that it is not a video game.

Since I built the Cinematic Sandbox as a web application, I had two options for building the user interface: I could build the interface within the game engine, or outside of the game engine in the browser. To facilitate quick iteration in building the Cinematic Sandbox, I chose to build the user interface in the web browser using HTML, CSS, and JavaScript. These three languages support very quick iteration so they were a great choice for the interface; since there was no build process, I could simply refresh the web page to see the changes I had made. Another platform choice I considered was which browser to develop for. In the spirit of rapid iteration, it was not very practical to support all browsers and have to deal with cross browser compatibility issues. I chose to support Mozilla Firefox because it is available on both Windows and Mac, the two operating systems that I targeted for development.
Simplifying the Project and Required Features

Developing an entire editing environment within Unity is a very ambitious undertaking. Therefore, I chose to simplify the scope of the Cinematic Sandbox in order to be able to finish it. I assumed that the user had already animated each sequence of their film in Maya. This made my job in developing the Cinematic Sandbox much, much simpler, while still providing a useful product that will speed up the transition process from two dimensions to three dimensions. I focused on allowing the user to load the sequences animated in Maya, split them into multiple shots, and position the camera in each shot. This serves as a proof of concept that it is possible to pull off the overall idea of the software by developing the core functionality. I built each of the following features as part of the core functionality of the Cinematic Sandbox:

- Load animated sequences from Maya
- Save changes out to XML
- Load XML list of changes back into animated Maya sequences
- Load XML list of changes back into web application

Loading animated sequences from Maya was an important requirement to satisfy for the Cinematic Sandbox. Since I did not focus on creating animation from scratch for the proof of concept model, the animation needed to come from somewhere, and Maya was the clear choice. Importing the changes made in the web application back into Maya was another important requirement because the students need their prototyped film in Maya to polish animation. I chose XML as an intermediate step because Maya doesn't support loading files directly from Unity, and having an intermediate step makes it possible to save data from the Cinematic Sandbox and load it directly back in without going through another software package. In addition, XML is easy to understand and edit, and it could be useful for the students in the class to manually edit some of the saved files. There are other data
interchange formats that I could have chosen such as Thrift or Protocol Buffers, but XML is very widely supported and is easy to work with in most programming languages. In addition, many programmers have worked with XML, so this should make development easy to pick up for anyone who should decide to join the project.

- Make changes in real time
- Pause/Play animation while making changes
- Fast forward and rewind through animation to view changes faster or in reverse

Making changes and viewing them in real time was another critical feature to target for the proof of concept model of the Cinematic Sandbox. It is vital to the core functionality of the product, so it was important to prove that it could be done early in the development process rather than later. There were several necessary features that came along with real time editing and playback. The ability to freeze the animation on a particular frame to make adjustments will be useful, so a play/pause feature was required. Also it will be very useful to play specific sections forward and backward during the editing process, so the ability to fast forward and rewind through animation was necessary. Animators are very used to fast forwarding and rewinding through animation in Maya as they work, so this will be a natural extension to their workflow.

- Split sequences into multiple shots
- Go to next or previous sequence or shot instantly
- Once shots have been defined, change the length of shots
- Position camera within each shot
- Define multiple potential cameras for each shot

The layout work that I targeted for the proof of concept model of the Cinematic Sandbox was cinematography, which is determining where shots start and end, and also where the camera should be placed during each shot. I chose to work purely with static cameras in order to simplify the design goals of the project. Adding movement to a camera in each shot leads to an explosion of required features including things like a keyframing system,
and a graph editor to control the transitions between key frames of the camera. In addition, students beginning in the animation capstone tend to use moving cameras where they are not appropriate and do not help tell the story. Having the limitation of static cameras will likely help their cinematography to be more professional and better communicate the story that they are attempting to tell with their film. Along with the core cinematography features of the Cinematic Sandbox came several related features including the ability to lengthen or shorten shots once they are defined, and the ability to define multiple camera positions for each shot. This allows animators to experiment with shot length as they edit, as well as directly compare different camera angles in order to determine the most effective way to tell their story.
Architecture

The Unity game engine allows a user to choose to code in C#, JavaScript, or Boo. I am not familiar with Boo, and the JavaScript used in Unity seems to be a mash-up between traditional browser JavaScript (which while well defined is not implemented in a standard way) and the .NET framework’s JScript. C# has many useful features, is a relatively mature language with lots of community support, and can utilize all of the classes of the .NET framework. Given my unfamiliarity with Boo, and the poor documentation for the JavaScript/JScript mash-up, I have chosen to code in C# within the Unity game engine.

Maya allows a user to choose to code in MEL, Python, or PyMel. MEL is a programming language that is syntactically similar to bash scripting and has a very limited feature set. I have found it to be cumbersome to use and frustrating to debug, so I have chosen not to use it for the Cinematic Sandbox. The Maya implementation of Python hardly seems like Python at all. It allows developers to take advantage of Python syntax, but sacrifices the object-oriented nature of Python. PyMel is a third party extension that attempts to fix many of the problems with the Maya implementation of Python that make it awkward. However, since it is not natively supported by Maya and requires additional setup, I have chosen to use the Maya implementation of Python and live with its shortcomings.

C# Classes

There are several main C# classes that interact to allow a user to control the cinematic aspects of their animation through the Unity game engine. The bulk of the implementation details in the Cinematic Sandbox lie within the C# code.

Setup

The Setup class is entirely procedural and serves to create the other classes when the web application starts up. It looks at what Maya files are in the Unity scene, and for each file creates a Sequence or collection of related shots. These are the sequences that users will split into different shots.
Sequence

The Sequence class represents a sequence in the film, which is a collection of related shots that tell a distinct part of the story and that all happen at the same time or location. The Sequence class stores all of the information about a sequence, such as whether its contents should be rendered to the screen, how long the sequence is, and how many shots are contained in the sequence. In addition, the Sequence class statically keeps track of global information related to sequences such as which sequence is currently being played and the order of all the sequences that have been loaded. The Sequence class also stores global information vital to playback such as, the current frame number, whether playback is playing or paused, and whether playback is looping over the entire film, or just the current sequence. The Sequence class adds the functionality to be able to skip to the next sequence or shot or to an exact frame.

Shot

The Shot class represents a shot in the film, or a continuous piece of motion viewed from a single camera angle. The Shot class stores all of the information about a shot, such as how long the shot is and what cameras have been added to the shot. Each Shot can contain multiple cameras, and knows what order the Shots are in for the containing Sequence. The Shot class adds the ability for users to split up sequences into shots where the camera will change position, and to lengthen and shorten existing shots once they are defined.

MayaStyleControl

The MayaStyleControl class enables a user to position the camera within a shot, and is named so because mirrors Maya's input method. It will be easier for students in the animation capstone to only have to learn one input method for manipulating objects in three dimensions rather than having to switch between different methods for Maya and the web application. Students can rotate the camera around a focal point with the left mouse
button, translate the camera in the plane of the screen with the middle mouse button, and move the camera perpendicular to the plane of the screen with the right mouse button, just like in Maya. An additional feature that matches Maya's functionality is that users can click on an object in the shot to select it, and press the F key to focus the camera on that object.

**CameraInfo**

The CameraInfo class is mostly a container for the aggregate data needed to store the positioning of a camera, which is the x, y, z, position and rotation. The user is able to change the focal length of the camera, but is not encouraged to do so because it is not consistent between Maya and Unity. Users can also show and hide "third lines," so that they can more easily follow the Rule of Thirds, a compositional rule of thumb [10].

**Communicator**

The Communicator class is in charge of handling all of the communication in the web application between the C# running in the game engine and the JavaScript running in the web browser. The JavaScript needs to be aware of changes to the underlying C# data so that it can update the HTML and CSS on the web page accordingly, and the C# needs to be aware of JavaScript events in order to receive and act upon user input. Ideally, all of this communication flows through a single channel, and that is role that the Communicator class plays.

**Util**

The Util class exists to hold functions that don't really belong to any other class. One such useful function is converting back and forth between frame number and time, which is necessary because Unity controls timing based on minutes and seconds, despite the fact that animators are used to editing in frames. Other useful functions include matching the position, rotation, or both of one object in the shot to another, and changing the parent-child relationship of objects in a shot.
**HTML, CSS, and JavaScript**

The user interface was coded in HTML and CSS to support fast iteration. The JavaScript on the page that communicates with the game engine is functional, and therefore not organized into classes. Its sole job is to update the HTML and CSS on the page based on messages from the game engine, and pass messages to the game engine based on user input. This is not the best way to implement a user interface, but I planned to port the user interface into Unity once the user interface was stable. Writing the user interface within the Unity game engine will make cross-browser compatibility much less of an issue, because most of the code currently running in the browser will be running in the game engine. Also it will make testing much easier, because there will be much less communication between C# and JavaScript.

**Python Script**

All of the scripting within Maya is written in the Maya implementation of Python. Currently, this is a single, short script that parses the XML output from the web application and applies the changes prescribed. This includes splitting a Maya file representing a sequence into a separate Maya file for each shot, and placing the cameras within each of those shots.
**Future Work**

**Additional Features**

The Cinematic Sandbox isn't really complete without some of the features that I chose to cut in order to make the project a realistic senior thesis project. Right now, the project can be used to speed up the process of creating animatics, but it could still be improved. One of the main issues that I'd like to address in the future is adding the ability for users to lay out their film entirely in the Cinematic Sandbox. Currently, the user has to work on basic animation in Maya and then tackle the cinematography in the web application. For users to work entirely within the web application for their layout, the following features are required:

- Run time importing of models into Unity
  Importing models while the web application is running is not critical, but it prevents having to rebuild the web application when a user wants to add new models and would be a great step toward speeding up the process of laying out films. There are currently several open source solutions to importing .obj files into Unity at run time, which is a file format that Maya can export.

- Translation and rotation tools to position models
  Positioning assets at key frames is critical for users to be able to lay out their films without opening Maya. It would allow users to apply basic animation to the models in each sequence as well as address the cinematography of the film.

- Keyframe system to add basic animation to models
  There are two approaches to setting key frames that will be implemented. The first is the same way that Maya approaches key frames, so it will be intuitive for animators. They will simply set playback to a specific frame, and then position the asset that they are animating. The game engine will interpolate between the key frames to provide continuous motion. The second approach is similar to a movement system common to many real time strategy video games. As the animation plays, users will update the animation by clicking
the screen to create waypoints. A waypoint tells the asset being animated that it should be at the waypoint on the frame where the waypoint was created. Creating waypoints is a way of animating in real time similar to GRAIL’s work with rapid prototyping. There is another programmer interested in developing for the Cinematic Sandbox, so much of the work to allow users to animate within the Cinematic Sandbox is already complete.

**Other 3D Applications Besides Maya**

It will be relatively easy to make the Cinematic Sandbox useful for other three dimensional animation applications besides Maya. Unity natively imports models from many other popular programs including Cinema 4D, 3ds Max, and Cheetah3D. The only trick would be getting the data saved out as XML back into those applications. All of these applications are scriptable, like Maya, so a similar approach to importing the XML data is possible.

**User Interface Improvements**

Animators used to working in Maya will find the method of positioning cameras within a scene intuitive. However, it is not an ideal input method for users without a mouse with a scroll wheel. For instance, users with laptops will have a hard time translating the camera since most laptops don't have a scroll wheel. It would be useful to add another method of input where the user can select what type of action they'd like to perform from a list on the screen, such as zoom, rotate, or translate, and then use the left mouse button to perform that action.

Another problem in the current user interface is that users can inadvertently lose their data if they refresh the web page or navigate to another page. Ideally, users would be warned when they attempt to navigate away from the page and asked to confirm that they actually want to do so.
**Testing Improvements**

Since I worked on the project alone, I didn't write unit tests for any of the classes, and I didn't write any larger tests for the system. The longer I work on the project, and as other developers join the project, this will become increasingly unmaintainable. Before too many more features are added, I will write unit tests for all of the C# classes as well as unit tests for the Python scripts that run inside of Maya. I will also implement a process of code review for all developers working on the project to follow in order to ensure that code checked in to the repository meets quality requirements and that new tests are checked in with new code.

**Cross-Browser Compatibility Improvements**

I originally chose to implement the user interface in the browser so that I could iterate quickly and see changes without having to recompile code. Now that the user interface has been developed and is fairly stable, it will be helpful to port it over to C# and run it within Unity. This will make testing much easier because there will be much less need for C# to communicate with JavaScript. Additionally, this will make the web application stable for more than just Mozilla Firefox because there won't be JavaScript running to update all of the user interface elements on the page. Ideally, I will target Internet Explorer, Firefox, Chrome, and Safari for future iterations of the Cinematic Sandbox once the user interface is integrated into Unity.

**Code Improvements**

For Maya 2011, Autodesk decided to add native support for PyMel. This is a great improvement over the Python implementation in Maya 2009, so it would be worthwhile to port the Python scripts that run in Maya from Python to PyMel. This will enable them to be written in an object-oriented manner, which will make them more consistent with the rest of the code base. Additionally, it will make it easier to write unit tests for the Python scripts.
**Documentation**

Throughout development, I have been the main user of the Cinematic Sandbox. There have been several other people that have used the software as it was developed in order to suggest which features were necessary, which features should be cut, and how the user interface has been changed, but I was able to sit with each of them and explain how the software could be used. If I hope to use the Cinematic Sandbox for the animation capstone or make it useful in other contexts, I will need to add documentation explaining how the software is used. To fit with the project goal of not having to open multiple software packages, I would like to integrate the documentation into the application as much as possible. Students should not have to go to a separate help page in order to find documentation. Each button in the user interface will have a tip that shows up when the user leaves the mouse over the button. The tip will explain how to use the button and what the button can help users to accomplish. Also, hotkeys that aren’t currently associated with buttons, such as the “focus” feature, will be documented with tips that show up when it would be reasonable to use the feature. Users will be given the option to turn off tips, because although new users will need the help, these reminders could be annoying to an experienced user. In addition, during the code review process, developers would be required to update documentation corresponding to features they have changed or added.

**Beta Testing the Cinematic Sandbox for the Animation Capstone**

No amount of individual use or unit testing can uncover all of the problems with a software package. In order to make the Cinematic Sandbox truly robust, it will need to be used by many people for real projects. I aim to have the future work listed above completed in time for the web application to be used by the animation capstone in Autumn quarter of 2011. The software should be stress tested at the beginning of next year because I will
graduate shortly after, and then there may not be developers who know the code well enough to make quick fixes if problems arise for students in the capstone.
Conclusion

There has been much research in industry and academia with the goal of making animation faster. However, most of this research addresses making the motion capture process more efficient or creating animation within a shot faster. The animation capstone is in need of software that can provide real time prototyping of an entire short film, not just an individual shot. I chose to first address prototyping the cinematography of a short film for my thesis in a proof of concept model of the Cinematic Sandbox and leave prototyping the actual animation for future work.

The proof of concept model of the Cinematic Sandbox proves that the overall goal of the project is possible and could be completed with a few more months of work. There are obvious improvements needed to make the software a great tool for prototyping animated films, but it is still useful in its current state. Right now it is possible to work on the cinematography for an animated film and view the changes in real time rather than having to wait for renders. Additionally, all of the cinematography can be done in one software package instead of having to edit in multiple pieces of software.

To make the Cinematic Sandbox truly reach the goals for the project and be worthy of integration into the animation capstone, there is still work to be done. The most critical pieces of this work are allowing users to import Maya assets at runtime without having to recompile the web application, and allowing users to create basic animation on those assets. This work is already in progress and should be finished by the time students in the animation capstone are ready to prototype a film early next school year.
Acknowledgements

I would like to thank Barbara Mones and David Hunt for overseeing my research, the development of the Cinematic Sandbox, for encouraging me to continue, and for keeping my goals realistic.

I would like to thank Robert Forsberg for starting development of the features described in the Future Work section of this paper.

I would like to thank Alicia Kellogg for help proofreading and editing this paper.

Source Code

The source code for the custom software is currently under development, and can be found at svn+ssh://attu.cs.washington.edu/projects/instr/production1/cinematic_sandbox. The code for the proof of concept model finished at the time of writing can be found at revision 13.
References


