IMPROVING ANIMATIC CREATION IN MAYA

A Thesis

by

JOSHUA BRENT SEAL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2017

Major Subject: Visualization

ABSTRACT

The goal of this work is the development of a new tool within Autodesk Maya for the creation of animatics. A brief history and survey of pre-visualization techniques and methods is provided, and the functionality and effective uses of current pre-visualization programs and tools are explored, focusing on an analysis of the current Maya Camera Sequencer tool. Development of the modified animatic creation tool (named ‘jSequencer’) is then discussed and the new tool is demonstrated using an example animation sequence. Results of the application of jSequencer are then compared with an animatic created with the Maya Camera Sequencer. jSequencer enhances the functionalities and adds elements that improve the animatic creation process’s efficiency and effectiveness; therefore, jSequencer is a better tool for creating animatics in the Maya workspace.

DEDICATION

This thesis is dedicated to my Mom and Dad, and my brother Jordan. Thanks for always supporting my goals and dreams, however farfetched they may be.

ACKNOWLEDGEMENTS

This thesis project would not have been achieved without the support of the faculty, staff and students of the Texas A&M Visualization department. Specifically, I would like to thank Dr. Frederic Parke, my committee chair, for his guidance and support throughout the proposal and thesis process. Without Dr. Parke's help, my research and project would not have succeeded. I would also like to thank my other committee members, Dr. Stephen Caffey and Prof. Tim McLaughlin, for their counsel and continuing encouragement.

I would like to thank my family for all their support throughout my academic career at Texas A&M. Their encouragement and trust were incredibly helpful during the most stressful times. I could not have succeeded without them.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of chair Dr. Frederic Parke and Professor Tim McLaughlin of the Department of Visualization and Dr. Stephen Caffey of the Department of Architecture.

All work for the thesis project was completed independently by the student, with edits made to the paper by Dr. Frederic Parke of the Department of Visualization and Christopher Seal.

Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

A common software package used for the pre-visualization or *pre-vis* process, Autodesk *Maya*, features the *Camera Sequencer* tool for creating animatics. The research in this project focused on modifying the Camera Sequencer and designing a more efficient animatic creation tool for Maya.

To better understand the impact of an improved animatic creation process on the production team during the pre-vis stage, a survey of the evolution of pre-visualization and of current tools and software was conducted. While the issues identified in software other than Maya are not specifically addressed in this work, they informed the development of the modified animatic creation tool.

1.1 Definitions

Development of animatics is part of a broader area in film production referred to as pre-visualization. There are a number of terms associated with pre-visualization and with Maya that have specific meanings in the context of this research.

The Maya Embedded Language, or *MEL,* is a scripting language used to run and simplify tasks in Autodesk Maya. MEL is descended from UNIX shell scripting, which means MEL executes commands to accomplish functions and processes in the same way as UNIX commands. Most menu selections within Maya's Graphical User Interface (GUI) can also be accomplished at the command prompt using MEL.

Animatics are a constituent component of pre-visualization and can refer to the preliminary version of a movie, produced by recording successive images of a storyboard and adding a soundtrack. The term "animatic" also refers to the video created in 3D during the layout/pre-visualization stages of animation or visual effects (VFX). These animatics are focused on the more technical aspects of the shots, such as composition, camera moves, and character animation. The creation of animatics is the primary focus of this research.

Pre-visualization, or *pre-vis*, supports project development using digital software, developing scenes from early concept and design through filming and post production while managing technical data, digital assets, and video footage. Pre-vis is usually one of the earliest stages of film production. A more thorough discussion and understanding of the term and process is given in Section 2.

In the context of this research, *workflow* refers to the way a particular type of work is organized, or the order of the stages in its process. In the case of this research, the type of work is animatic creation. The number of people involved and their workflow can vary during animatic creation depending on the scope of the project.

A *panel* is a collection of UI screen objects (buttons, fields, graphical views) that are grouped together. A panel can be moved around as a group within the application interface or separated to exist in its own window. A scripted panel is a panel that is defined using MEL script, with all required functions available as MEL procedures in the installation files.

Workspace is defined as an arrangement of screen windows or panels, with interface options or tools designed for various groups of tasks. Within robust 3D packages such as Maya, the user can *"modify the current workspace by opening, closing, and moving windows, panels, and other UI elements, as well as docking and undocking windows and panes."* (19)

A *Playblast* is a preview "sketch" of the animation, providing an idea of the result without requiring the time needed for a complete render. "*Playblasting gives you a fast way* *to evaluate your work on the fly, taking a screen grab of the animation in the viewport at each frame during playback, and then "blasting" those images to an image viewer. By default the Playblast tool generates image sequences using the active view and current time range in the Time Slider to determine the animation range*." (19) This paper includes several versions of this term, but all refer to the definition above.

In the context of computer animation, a *pipeline* is a set of sequential processes and tools that transport data through the various stages of production. *"Every major visual effects (VFX) or feature animation production studio utilizes a pipeline to manage data and workflows, enhance communication, increase efficiency, and aid troubleshooting when things go wrong."* (12)

The term *production platform* refers to an integrated 3D software suite that artists use to create 3D animation content.

2. BACKGROUND

2.1 Evolution of Pre-visualization

Film directors in the classic Hollywood era depended on the final-draft screenplay for guidance in set building and camera location. (16) As camera language, or how directors used cinematography and editing to tell the story, became more common, directors would use sketches of shots to help visualize their films.

Shooting a scene in one continuous shot from a static vantage point was the standard in early films. Innovators like D.W. Griffith and Sergei Eisenstein introduced new camera shooting and editing techniques which required more pre-production planning. These techniques became the standards for film production.

While pre-production sketches in live action were being used by a few directors, pre-production drawings and planning were widely used in the production of animated films. Webb Smith, one of the early Disney sketch artists, is credited as the father of storyboarding. During production of one of the Mickey Mouse short films, he mounted small sketches on the wall for Walt Disney and the other story artists to discuss. The small page-sized storyboards evolved into larger mounted storyboards.

Storyboards were crucial during the development and production of *Snow White and the Seven Dwarfs* (1937). They provided a site for creative convergence. Lasseter notes:

> *"Not only did this [large board] format allow a sequence to be restructured or refined on the spot, it made it easy for an artist to 'pitch' (describe and act out) the scene to others. Walt Disney's reputation as an unparalleled storyteller was cemented in front of such boards, as he analyzed sequences, suggested changes, acted out key moments, concocted gags, and tweaked shading."* (11)

Storyboards provided a creative site where dialogue and gag ideas originating from the storyboard artists were finalized. As ideas progressed through the production process, other artists and storytellers reworked them, facilitating the creative process and cooperation between studio departments.

Ray Harryhausen's innovation was to precisely integrate stop-motion characters into live-action settings. He would often draw directly onto location images then create rough drawings focused on character action rather than background detail. His storyboards provided enough specific information to serve as a useful starting, middle, or ending point to guide the stop-motion sequences, while being vague enough to enable him to improvise during the animation process. (19)

When George Lucas formed Industrial Light and Magic to create the visual effects for his *Star Wars* films, he established pre-production as the best planning method to meet the demands of special effects filmmaking. During the production of *Star Wars: The Empire Strikes Back* (1980), Lucas found that using storyboards as a basis to collaborate with writers and artists in different locations was particularly useful.

> *"Frequently wall-mounted, these storyboards established a space in which large-scale pre-visualization could take place; and, as is often the case where animation is concerned, they remained subject to constant revision and removal throughout production."*(16)

As digital technology progressed through the 1980s and 1990s, more emphasis was put on pre-production for planning sequences that would require computed generated or *CG* characters or environments. Using animatics of storyboards, stop motion animation, or early 3D animation became increasingly important in planning the expensive CG effects. The famous CG dinosaurs in Steven Spielberg's *Jurassic Park* (1993) were originally planned to animate in stop-motion, but once ILM proved that the dinosaurs could be created digitally, the stop motion work done by Phil Tippet's studio was used as the pre-vis and helped the animators at ILM with the dinosaurs' movement and camera composition.

Digital animatics became the primary method of previewing CG shots before going into final visual effects and sequence production. One of the first films to use digital pre-vis animatics was *Star Wars Episode 1: The Phantom Menace (1999)*. For most of these complex sequences, the pre-vis animatics were combinations of 3D animation, live action clips and puppets, and 2D sketches in 3D scenes. These animatics were instrumental for the transition from the storyboards to the final scene. Following the acclaim of the CG effects and successful implementation into the production pipeline, digital animatics and pre-vis became the most effective way of pre-production for CG sequences, effects, and characters.

2.2 Pre-visualization Today

With more demanding audiences, diminishing schedules, and ever-growing artistic ambition, pre-visualization has expanded into an important component of film production.

Pre-vis animatics perform a role in pre-production similar to the role editing plays in post-production: The ability to visualize sequences quickly and at low cost helps filmmakers and artists invent alternate ways of telling the story, explore ideas and choose the best ones. This can potentially save millions of dollars by testing scenes that might otherwise end up on the cutting room floor. The goal is to help filmmakers not only plan complex visual effects, but also to design sets and coordinate stunts. According to Chris Pallant:

> *"Pre-vis has become the step in the pre-production process that the film crew strive to reach as quickly as possible. As with the animatic and the storyboard before it, pre-vis presents the most accurate vision possible with the resources available of how the final film should look."* (16)

In the words of pre-vis pioneer Ron Frankel, pre-vis is becoming "a nexus of inter-departmental communication." With the technology currently available, departments are able to quickly create animatics and communicate shot information with each other during productions.

> "*New digital production demands a non-linear workflow that is available to all departments and to the director and all creative leads. It requires a collaborative workspace that threads through all of production and iteratively generates the metadata of any film from inception, and gathers and distributes information throughout the production. Pre-vis is perfectly placed to provide this virtual production space from early concept and design through locations, blocking, sequence, capture and post production, while managing technical data, assets and footage throughout production."* (3)

From its initial roots in the conceptual phase, pre-vis has now expanded into five roles. The first, *pre-vis*, is the conceptualizing of shots at the preproduction stage. The second, *on-set pre-vis*, creates real-time visualizations on location to help the director, visual effects supervisor, cinematographer and crew quickly evaluate captured imagery. Third, *post-vis* involves the compositing of pre-vis into the live-action footage for the verifying of final effects. *Tech-vis* incorporates and generates accurate camera, lighting, design and scene layout information to help define production requirements. Finally, there is *design-vis*, which provides a preliminary, accurate virtual design space in which production requirements can be tested, and locations can be scouted.

> *"Pre-visualization acts as the hub of a new digital process of making movies. Information from multiple departments coalesce within pre-vis and when orchestrated correctly, pre-vis provides an array of advantages (both technically and conceptually) to a director that were unheard of years ago."* (18)

The general pre-vis approach for both animated features and live-action films is similar. Larger animation studios like Walt Disney Animation and DreamWorks Animation use their own proprietary tools. At these studios, all of the branches of pre-visualization are done in the Layout Department. Storyboards are given to the layout artists. These artists then animate the characters with block poses and rough sets. The supervisor of the scene defines the animation detail needed for the layout phase, but the rule of thumb is "*Enough to convey the action and story.*" (4) The initial cameras are usually placed to establish focal lengths and shot compositions to match the storyboards.

At Disney, the layout department uses a custom camera toolkit called "QT" that *"uses actual movements of camera rig such as roll, pan, tilt, boom, swing, shake, and has 2D camera features inside 3D camera."* (15) Once cameras are placed, the Maya *Playblast* tool is used to generate the initial pre-vis animatics, which contain the rough sets, animation, lighting, and effects done by the layout artists in pre-vis. The initial layout sequences are then moved forward in the production pipeline by the respective departments such as Animation, Lighting and Effects. After the director and department heads approve the separate pieces of the shot, the layout team finalizes the cameras in post-vis and delivers the final shot sequence to the editing and compositing departments.[13]

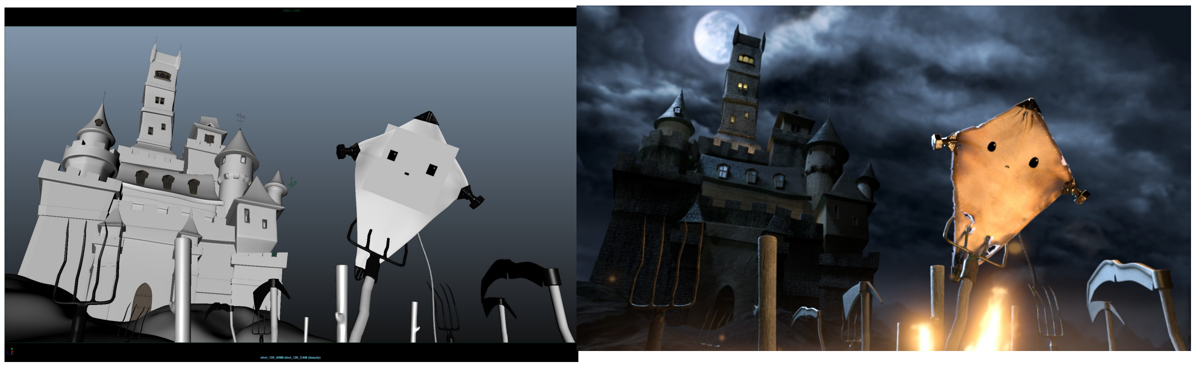


Figure 2.1: Frames showing the difference between a layout/pre-vis animatic frame and the final render. Taken from "*Frankenkite*" animation created at Texas A&M University (21)

2.3 Current Software and Tools

Today, films with multiple visual effects shots often contract pre-visualization to specialist studios such as The Third Floor, Pixel Liberation Front, and Halon. Artists at these studios build most of the pre-vis sets, characters, and animations in software packages such as Maya, Zbrush and Motionbuilder, or use libraries of pre-made rigged character models, sets, vehicles, and effects stand-ins.(8) Pre-vis and post-vis supervisor at The Third Floor Gerardo Ramirez explains:

> *"We used The Third Floor's standard pre-vis pipeline on this project: Autodesk Maya for modeling, rigging, animation and rendering; Adobe Photoshop for texturing; After Effects for the compositing of pre-vis and post-vis shots; and PFTrack during postvis to track the live-action plates. When creating assets, we would receive information from various departments. The visual development team would provide concept art for the characters, the art department would give us set designs and the location department would send reference images and survey data for the filming locations. Our asset team would then build and rig the models in Maya."* (5)

Depending on the budget and scope of the project, studios will use various software for pre-vis. Programs like ToonBoom's *Storyboard Pro* are used for creating storyboards and 2D animatics. Storyboard Pro allows the artist to create storyboards with layers using several brushes (similar to Photoshop) and manipulate objects and cameras in both 2D and 3D space. Storyboard Pro also features a tool which allows the user to watch the shots in sequential order and then adjust the timing between frames. While considered a useful tool for storyboarding, Storyboard Pro lacks full animation capabilities and is expensive.

Reallusion's *iClone* focuses on storyboards and real time 3D animation for tablet and smartphone mobile applications. iClone allows users to import and export content such as characters, props, and animation data from external 3D packages like Maya or the *Unity* game engine. While iClone can efficiently produce real-time animations, it depends on outside software (like Maya) to create the models, rigs, and more complex animation.

FrameForge 3D is a pre-vis centric tool with built-in or “stock” character models containing rigs, props and sets. FrameForge 3D allows for various cameras, lenses, and types of rendering and is thus a relatively quick process to create and share shots with the production team on low budget productions. FrameForge 3D lacks the tools to create complex animation and high quality models. Both iClone and FrameForge 3D are inexpensive options that allow independent artists to design and create accurate storyboards and pre-vis for live action production. (14)

Maya offers a complete creative suite for 3D computer animation, modeling, simulation, rendering, and compositing on a highly extensible production platform. Many of the deficiencies and shortcomings in the tools listed in previous paragraphs are addressed in the Maya environment. By enhancing the animatic creation capabilities within it, a new or modified animatic tool which takes advantage of the powerful features of Maya would help make it a preferred environment for animatic creation.

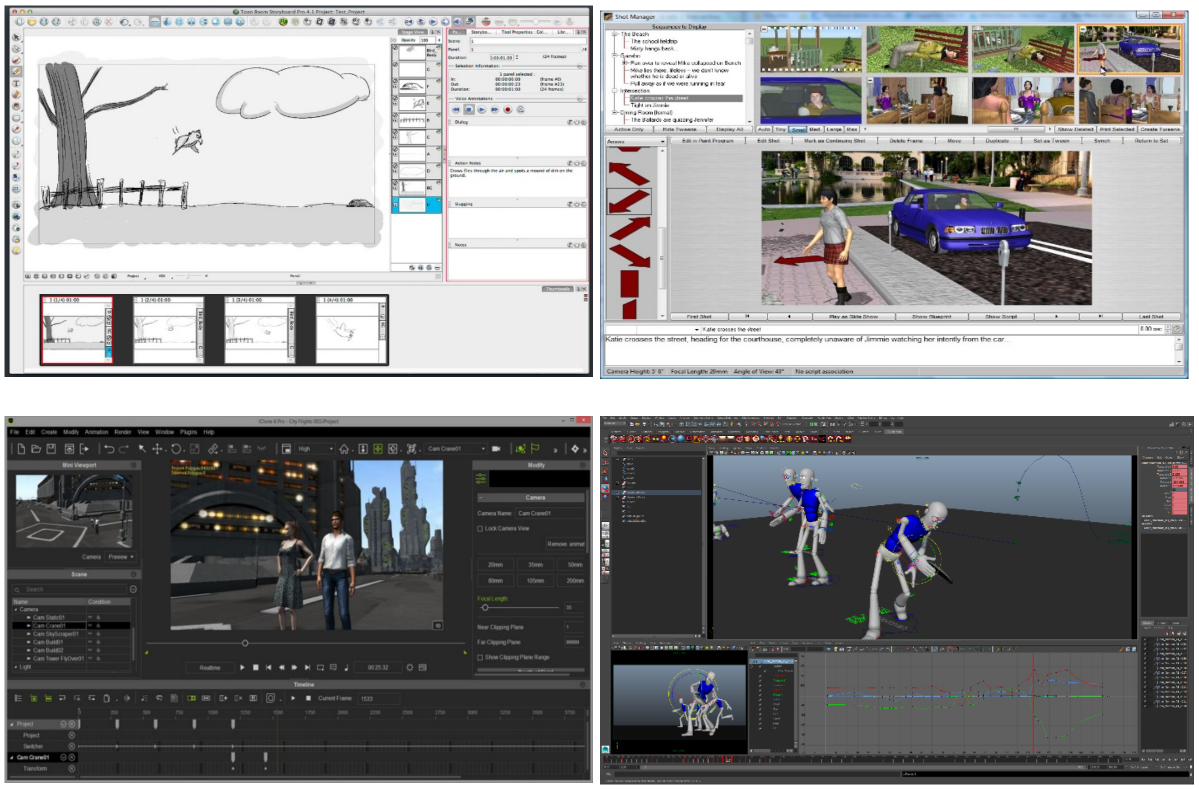


Figure 2.2: User Interfaces of the different types of software used for creating pre-vis animatics. Going across from top left: Storyboard Pro(2), FrameForge3D(6), iClone(10), Maya

3. METHODOLOGY

The approach of this research is to develop and demonstrate a modified animatic creation tool for Maya and to confirm that use of the new tool is a more effective and efficient way to create animatics. The background of pre-visualization software discussed in Section 2 was utilized to determine what improvements could be made to the Camera Sequencer. The description and functionalities of the Camera Sequencer are discussed in Section 4.

3.1 Goals and Objectives

The goal for this project was to research, develop, and demonstrate a more efficient and effective version of the Camera Sequencer. The project had the following objectives:

- To develop a clear understanding of the current Maya animatic creation tool and its limitations. This also determined how a new animatic creation process could be developed to improve its functionality and effectiveness.
- To design a MEL based animatic creation tool that is an improvement upon the existing Camera Sequencer.
- To develop a thorough understanding of the MEL script language and how to use it. This knowledge informed how to modify the Camera Sequencer so that the new tool follows the design specifications.
- To create the modified Camera Sequencer tool using MEL, based on the design specifications.
- To demonstrate the new tool by creating example animatics, showing how it improves the workflow, process speed, and provides more useful data for the artists.

The results of this demonstration will be determined by comparing the default Camera Sequencer to the modified version.

4. FUNCTIONALITIES AND DESIGN SPECIFICATION

4.1 Maya Camera Sequencer

Due to Maya's animation and camera capabilities, many artists use its Camera Sequencer to create their layout animatics.

> *"The Camera Sequencer is a tool that lets you create, rearrange, and manipulate camera shots of your scene. Each shot defines which camera is active at any particular time, and for how long. By adjusting the placement and timing of camera shots, you can effectively manage the 'filming' of the animation in the scene. When you finish manipulating camera shots in the Camera Sequencer, you can playblast the sequence into a rendered movie clip."* (1)

Shots are set up within the shot view area of the Camera Sequencer, in a fashion similar to a video editor like *Adobe Premiere* or *After Effects*. Camera shots are represented as a set of rectangles arranged on tracks in the clip editor. These represent the time(s) at which a particular camera is active in the viewport. Information presented in the rectangles include the shot's start and end frames, shot durations, the sequence time showing where the individual shots are placed, and the camera name. These attributes are set when the shots are created. Shots can be grouped together to make them easier to manage. For example, alternate versions of a shot can be collected together in a single group node and then moved and played together in the timeline through the Camera Sequencer window. When the layout and timing of the camera shots are set, the shots can be playblasted together into animation clips. Figure 4.1 shows a screen capture of the default Camera Sequencer GUI.

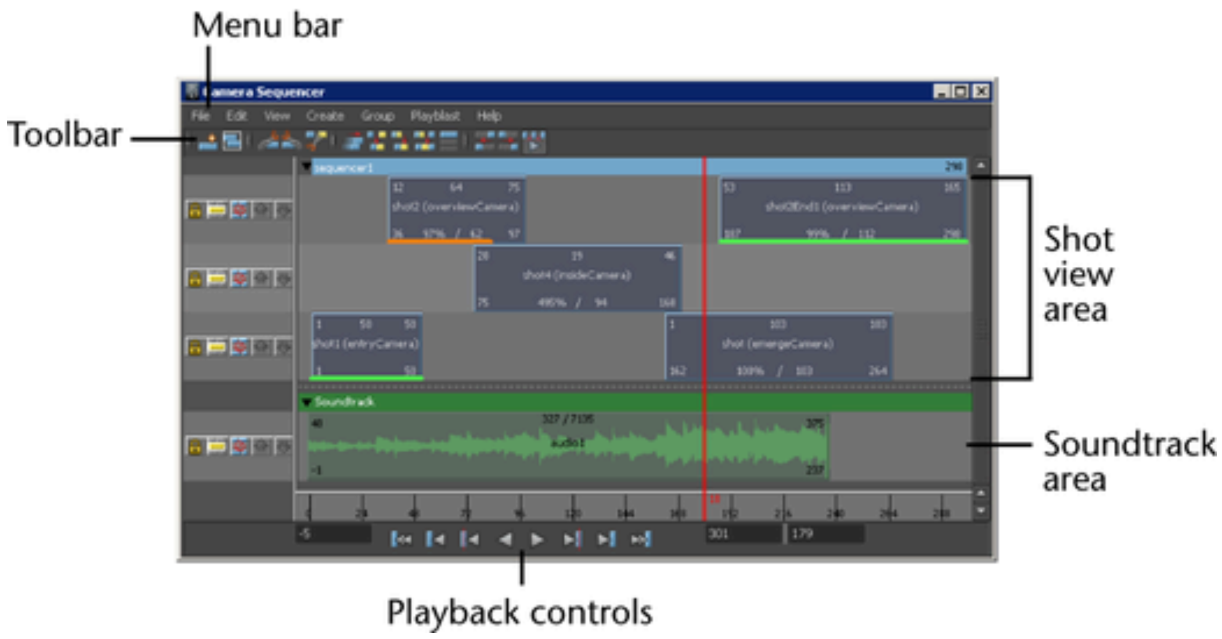


Figure 4.1: The default camera sequencer. From the Autodesk Knowledge Network (17)

The Camera Sequencer Menu Bar contains sub-menus such as File, Edit, View, Create, Group, and Playblast. The Toolbar is divided into collapsible sections and displays frequently used functions such as Create Shot, Ripple Edit, and Remove Overlaps as icons.

4.2 Design Specification

This project is intended to develop a modified tool in Maya that combines the functionalities of the Camera Sequencer with added elements that improve the animatic creation process. The modified tool will be developed using MEL script and be referred to as "jSequencer".

As discussed in Section 4.1, the Camera Sequencer is a tool that allows the user to create, rearrange, and manipulate camera shots in a scene. By adjusting the placement and timing of the camera shots, the user can effectively manage the "filming" of the animation in the scene. The basic idea behind this tool is that complex scenes with multiple cameras can be viewed all within a viewport. When you play an animation in the Camera Sequencer, as a default the animation is presented in the viewport that was last selected by the user.

Editing tools available in the Camera Sequencer toolbar include the Ripple Edit (which when selected, automatically adjusts the shots following a manipulated shot in the timeline), Split Shot (which cuts the selected shot at the current sequence time), and Remove Overlaps (which adjusts the track to remove any time overlaps between shots). Another Camera Sequencer functionality is the ability to import audio files that can be associated with specific shots or applied for an entire sequence.

Despite these functionalities, the utility of the Camera Sequencer for an experienced user is limited by constraints within the tool and within Maya's interface and settings. The jSequencer has been designed to address these limitations, as discussed in the following:

1. When using a multiple panel layout with several viewports in Maya, the Camera Sequencer automatically plays the shot sequence in the viewport panel last selected. The back-and-forth between the camera viewport and animation editor can often result in the wrong viewport being selected. The user then has to reset all of the viewports to the correct state, adding frustration and time to the workflow when editing animation and cameras. Viewport designation for the Camera Sequencer is available through the Ubercam tool (in the Menu Bar of the Camera Sequencer), but the Ubercam tool can only be used once the sequence is completed and ready for rendering. Since the primary function of the Sequencer is to quickly create and edit animatics, the capability to update in any viewport in real-time is needed. Enabling this functionality within the jSequencer should improve efficiency during animatic creation.

2. Unlike a standard shot-by-shot playblast, the Playblast tool in the Camera Sequencer cannot display the resolution gate mask. The resolution gate displays a border within the viewport showing the rendering area and resolution specified by the render settings. The mask surrounds the rendered area, framing the shot in an aesthetically pleasing way. It also provides a background for any display information needed in the viewport and animatic. The gate mask creates an effect similar to "letterboxing", or the practice of adjusting film aspect ratios to fit standard video formats. The resulting image has mattes (black bars) above and below. The matte areas are part of the overall image display and have an area for subtitles or text, avoiding overlap with the image. In the case of animatics, the mattes are where shot information should go. To display the resolution gate for individual cameras, the user must manually adjust the attributes of each camera by toggling the "Display Resolution" and "Display Gate Mask" boxes and adjusting the sliders for the "Gate Mask Opacity" and "Gate Mask Color" options. These functions are accessible via the individual camera's attribute editor and are not part of the Camera Sequencer. In the jSequencer tool design, manual adjustments are not required – the resolution gate and mask are automatically applied for each camera. By scripting these operations directly to the attributes of the cameras, the presentation in the camera viewport and the animatic will be more effective.
3. Specific shot information is missing in animatics created with the Camera Sequencer using the Playblast options. Modifying the tool so that the animatic will automatically display the missing shot information will improve the animatics created.
4. During playback within the Camera Sequencer, extraneous curves, joints, and other objects may be presented in the default viewport and can be distracting. The jSequencer

display will not include these types of objects and will save the user the time needed to manually turn them off.

4.2.1 Functionality Goals

The jSequencer will combine the tools and look of the Camera Sequencer with added elements that improve the animatic creation and review process. The new features of jSequencer are intended to resolve the limitations of the Camera Sequencer described in the previous section. They will be based on what is possible with the Maya procedures, commands, and interface. The added functionalities for the desired tool are the following:

1. The standard Camera Sequencer panel will be combined with a newly designed designated viewport into a window separate from the main Maya interface. The jSequencer window will have multiple user-adjustable panels and can adapt to different workspace layouts, giving the user the freedom of using the main display for other work without crowding the display. Because the jSequencer viewport will be assigned to the Camera Sequencer panel, the animation sequence in that specified location can be updated in real-time, no matter what edits and alterations are done to the sequence.
2. The jSequencer window will include a display of detailed shot information (such as user/artist, date, the camera name, the focal length of the camera, the frame number, and the overall scene time) in both the viewport and the playblasted animatics. This display also includes the resolution gate mask, providing a background for the shot information and presenting the animation sequence in an aesthetically pleasing way.

4.2.2 *jSequencer Display Design*

The image below is a visual breakdown of the sections and elements of the jSequencer display. These elements are described in Section 4.2.3.

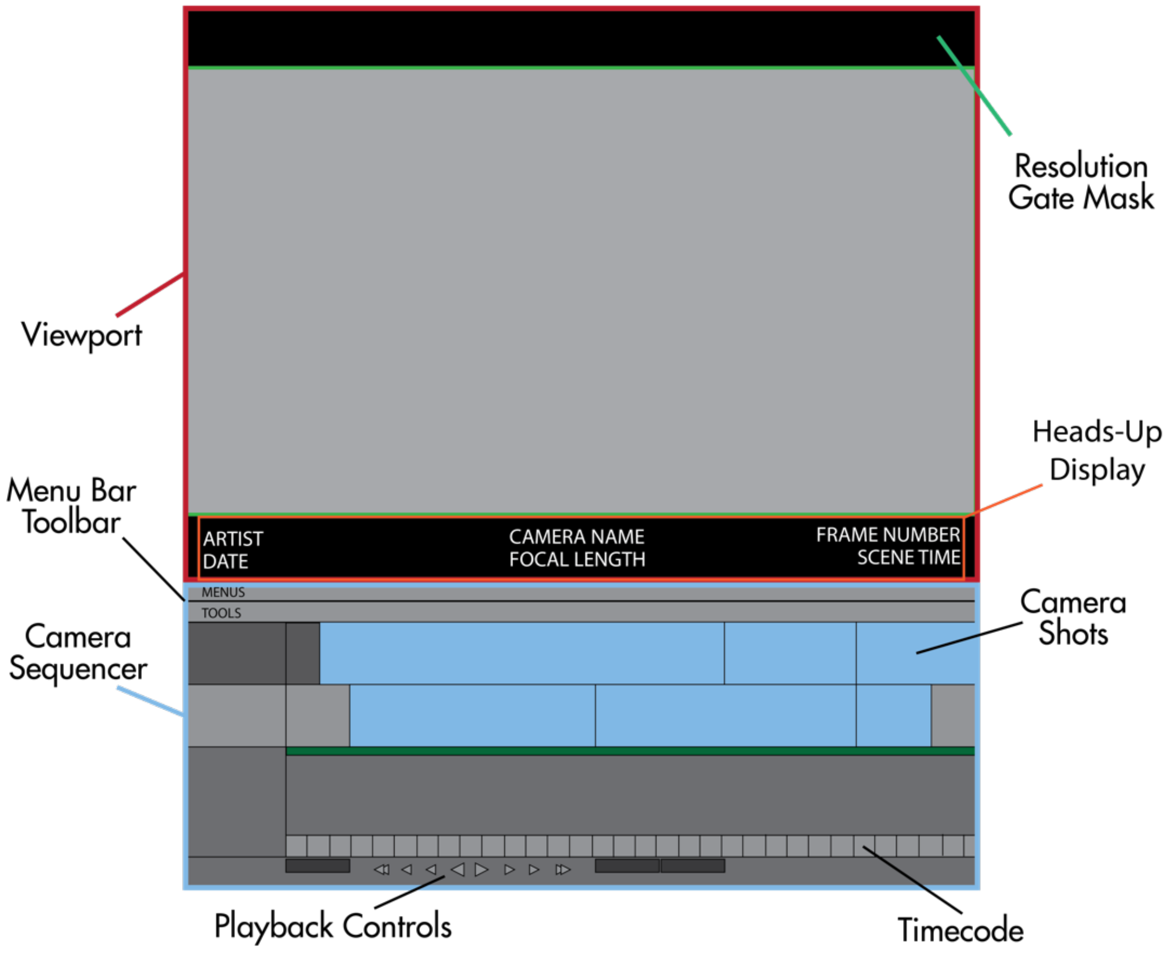


Figure 4.2: Representation of the modified Camera Sequencer display. This shows the separate elements of the jSequencer display specifying the separate elements and their locations in the window.

4.2.3 jSequencer Design Elements

The main design element in the jSequencer tool is the GUI equivalent of the "tear off" function in the Maya user interface. The "tear off" function creates a floating window (separate from the main display) of a specified *panel*, or a grouped collection of UI objects (buttons, fields, graphical views etc.). The state of its UI is maintained when it is relocated, resized, or recreated.

The display elements are created in Maya using MEL script. This section does not include specific scripts, but discusses MEL commands at a functional level. A more in depth description of the commands used are in Appendix A.

To create a customizable standalone window, the window() command is used. The window() command has flags which control the size and elements displayed within the window. The standalone window can be designed for either a single or dual monitor workspace and should include a layout with multiple sections and customizable size and layout type options. The jSequencer window displays the Camera Sequencer panel in the bottom section. (The contents of this panel are described in Section 4.1) The designated viewport created by jSequencer will display the sequence playback in the top section of the window.

The jSequencer tool uses the headsUpDisplay() command to display the important shot information as a 2D overlay plane on the 3D viewport. The specific information, such as user/artist, date, the camera name, the focal length of the camera, the frame number, and the overall scene time, will be visible in the designated viewport when working in Maya and also in the playblasted animatics created.

5. IMPLEMENTATION

5.1 jSequencer Development

The jSequencer was developed for Autodesk Maya 2015. Autodesk Maya is an industry standard animation suite which offers a powerful set of tools for 3D content creation and utilizes the Maya Embedded Language (MEL) for programming access and for scripting using built-in Maya commands. MEL is descended from UNIX shell scripting, which means MEL executes commands to accomplish functions and processes in the same way as UNIX commands. *"Most commands you use to control Maya act like UNIX command-line utilities: little stand-alone programs with many options that modify their behavior."* (13) Most menu selections within Maya's Graphical User Interface (GUI) can be completed at the command prompt by using MEL. In addition, because MEL is the foundation of Maya's infrastructure, tools and commands scripted with MEL should work in every version of Maya. Debugging and editing the scripts are also much easier than other scripting languages, because the Script Editor in Maya can show the specific MEL commands and errors using the "Echo All Commands" option. The reasons I wrote the script in MEL were my familiarity with the language and its specificity to Maya, making examples and tutorials easier to find through the MEL Command Reference and other online sources.

5.1.1 Separating the Camera Sequencer

Research to create jSequencer began with example scripts in the MEL documentation. Following the tool design specifications, the initial goal was the creation of the separate window. The "tear off" command in MEL creates a floating window, or a window not attached to the main display, for a specified *panel* or group of UI objects (buttons, fields, graphical views etc.). The panel takes care of maintaining the state of its UI when it is relocated or recreated. Following the echoed commands in the Script Editor when “tearing off” the Camera Sequencer panel enabled me to see the specific MEL commands used which led me to the correct commands and other resources to use in the jSequencer tool script.

One of the best examples I found focused on creating separate windows for Scripted Panels. A *Scripted Panel* (for example the Render View panel, the Graph Editor, and the Camera Sequencer) is a panel that is predefined in MEL and includes several pre-made tools, menus, and functionalities. The script example I found docks the scripted panel within a custom layout and adds buttons. In order to use the correct scripted panel, the panel type is sourced in the scriptedPanel() command. While it didn’t exactly fit the jSequencer design, this example became the basis for a window created with the Camera Sequencer. Several prototypes, as shown in Figure 5.1, were developed to separate the Camera Sequencer into its own window.

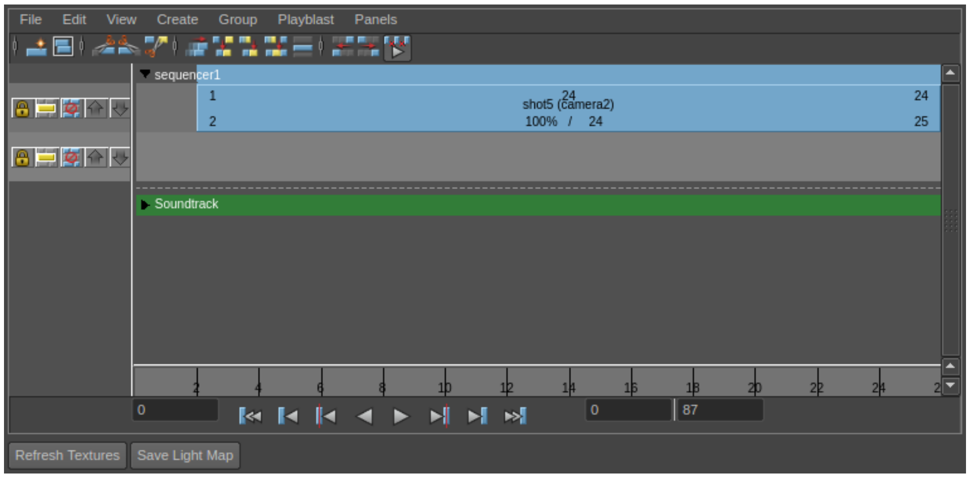


Figure 5.1: Screenshot of first prototype.

5.1.2 Viewport Editor

Once the sequencer was figured out, I focused on creating a separated viewport, referred to in MEL as a model panel. When a model panel is created, it includes a model editor and the modeling menu. These model editors show the default “perspective” views, but can be manually switched to other cameras. To create a model editor without the menu bar, the modelEditor() command was used. This command can also specify model shading modes, lighting and shadows, and cameras. For the jSequencer, a newly created model editor was used for the camera viewport. Figure 5.2 shows the “torn off” window of the new model editor. This new model editor was used in later prototypes.

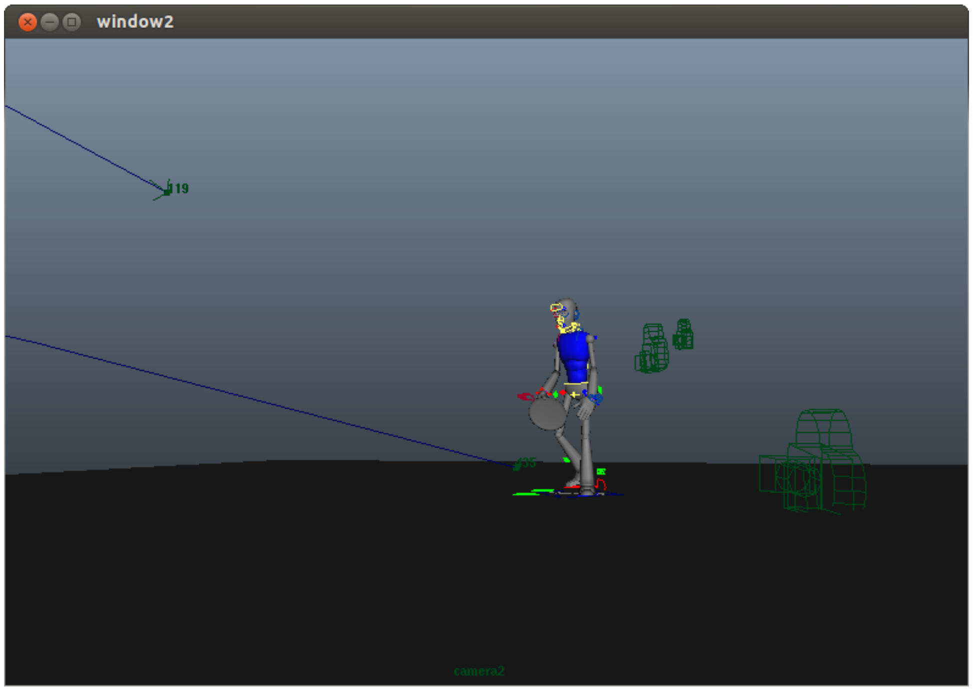


Figure 5.2: Screenshot of new editor window.

5.1.3 Window Layout

The next step was the development of the window layout. To get the layout correct according to the design specification, research in types of UI layouts was conducted. Many different layouts showing basic UI objects exist in MEL including column, row, form, and pane layouts. To use scripted panels like the model editor and Camera Sequencer, form layout and pane layout were prototyped to determine which was more effective for the jSequencer.

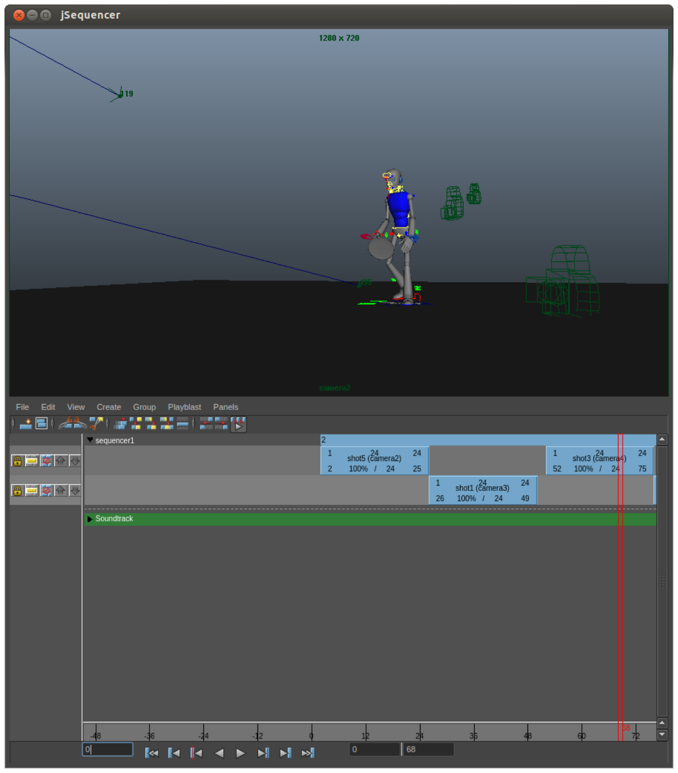
A form layout allows absolute and relative positioning of the UI objects (in this case the scripted panels) that are the layout's immediate children. In Maya, the parent is considered the main object and the children are considered sub-objects. Children's attributes, like position and scaling, are manipulated by the attributes of its parent. There is no default positioning relationship, so to have children appear in the form layout they must have at least one edge attached to another UI object in each direction. Because the *formLayout*() command is used to build most of the Maya user interface, it tends to function more predictably than many of the simpler, but less common, layout types. However, once the object positions are set, the size and relationship within the window cannot be altered without adjusting the script.

A pane layout creates multiple panels in a floating window, which are split into individual panes with separators. The panes are separate from each other and can contain anything created in MEL, such as custom controls, sub-layouts, or scripted panels. Configuration of the panes is also adjustable. Configurations control how many panes are created in the pane layout. For example, the "horizontal2" configuration creates two panels vertically separated in the middle of the window. To make sure the scripted panels are placed in the correct panes, and because the scripted panels return their own custom layouts, the *setParent*() command has to be used to signal the end of each pane. This command makes the scripted panels children of the pane layout. This information can all be set and edited within the *paneLayout*() command in MEL.

To compare the two layout types, prototypes of each were created. (See Figure 5.3). Although they seem almost alike in appearance, form layout uses absolute positioning for the UI objects, while pane layout can have adjustable panels. In the pane layout, the dotted line "pane separator" above the Camera Sequencer menus is the panel separator and can be dragged vertically to adjust the panel size. This became the preferred jSequencer layout, since it was user-adjustable.

5.1.4 Heads-Up Display

The next design element development was the shot information display in the viewport. This focused on the application of the *headsUpDisplay*() command, with additional research into existing plugins and examples. Tools like *animHUD* and *scHUDAnim* provided great examples on how to use the headsUpDisplay() command. Much work went into breaking down the code for these tools, focusing on common elements and how to use them in jSequencer. The main issue was figuring out what type of information to include. This problem was solved by looking at the types of information animators and VFX artists included in their reels and animatics. Common information includes camera name, focal length, and shot frame number. Basing my tool's HUD on the information displayed in these references, I programmed jSequencer to provide this useful information.



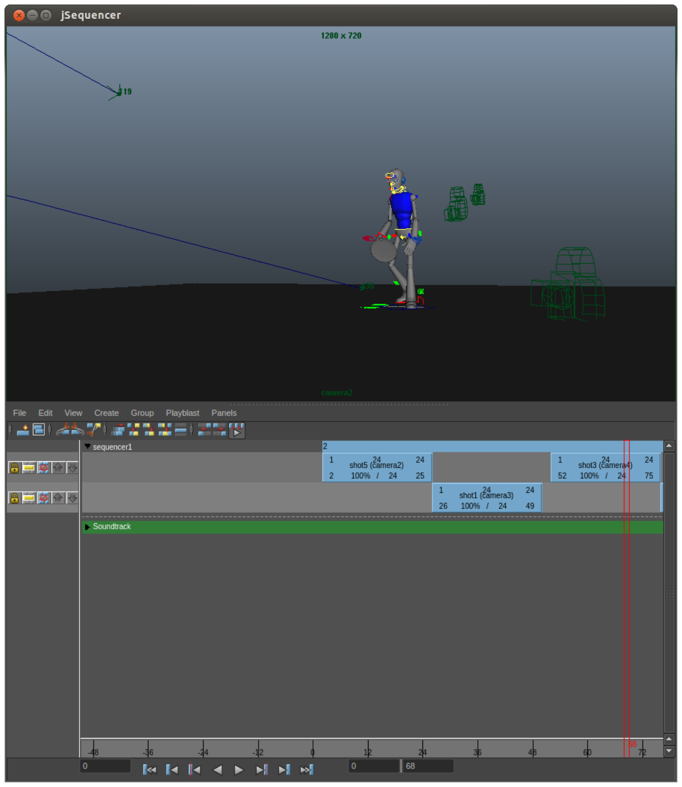


Figure 5.3: Both versions of the jSequencer window layout. Top is *form* layout, bottom is *pane* layout.

Turning the script into something that could automatically display this information exposed an issue with the headsUpDisplay() command. HUD objects depend on the placement flags, section and block. Each section is composed of a single column of blocks. If another HUD object or preset has the same placement as one of the newly created objects in the animatic tool, the script errors. There can only be one HUD object occupying a block at a time. The easiest remedy for this is using the *removePosition* flag, which removes the content of a specific block location in the HUD layout. This frees that space for the new objects created in the animatic tool's display. Due to the need for several HUD objects, the *removePosition* flag had to be used for each occupied block to free the space for each new HUD object. Once that was done, the HUD was correctly displayed. (See Figure 5.4)

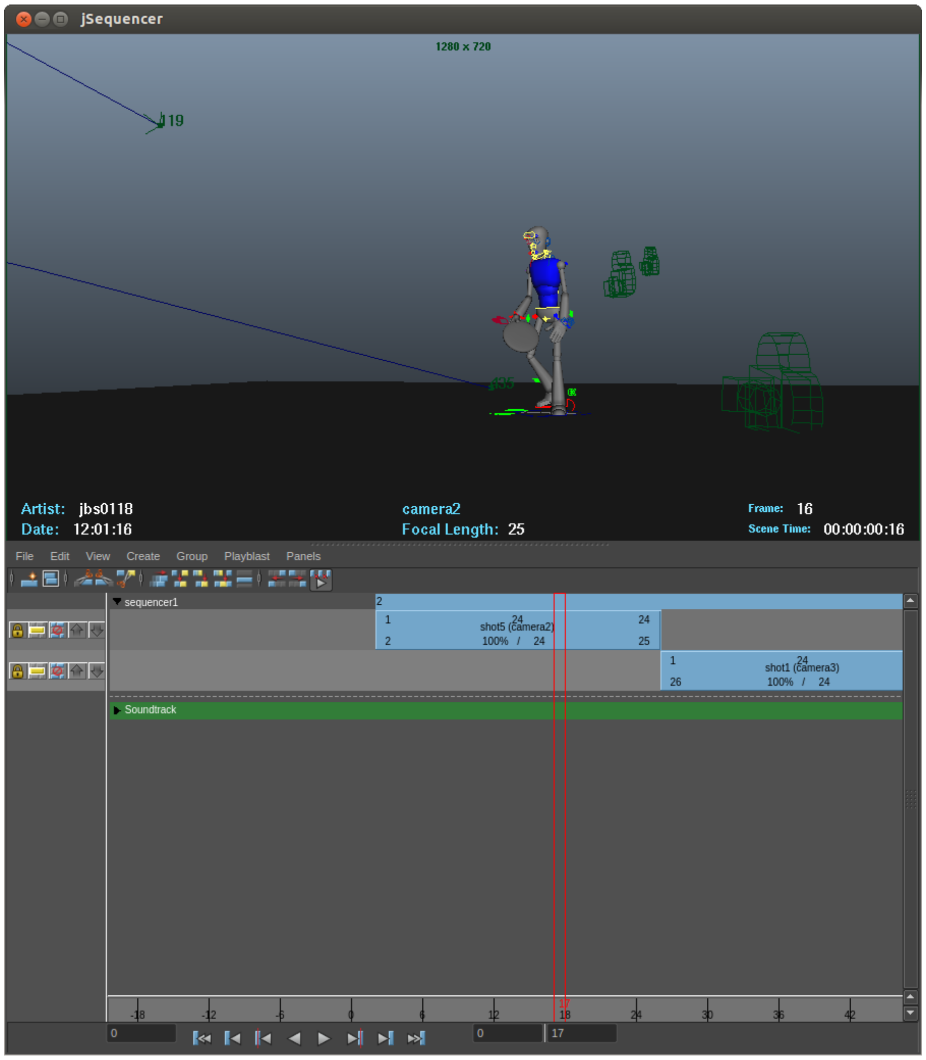


Figure 5.4: jSequencer prototype with the custom Heads-Up Display.

5.1.5 Resolution Gate

The final element in the jSequencer implementation was the resolution gate for the camera viewport. The resolution gate displays a border within the viewport showing the rendering area and resolution specified by the render settings. A mask surrounds the resolution border, framing the shot in an aesthetically pleasing way. It provides a background for the display information needed in the viewport and the animatic. The gate mask creates an overlay color outside the border set by the resolution gate, similar to the "letterboxing" technique in film. The gate mask can only be viewed when the gate is on. As discussed in the specifications, activating the gate mask involved manually going into the attribute editor of each individual camera to turn on the resolution gate and gate mask, and adjust the mask opacity and color values.

The final goal for jSequencer was to create a process which automatically completes these adjustments and speeds up the scene setup process. Using examples found in the MEL command reference and scripts accomplished for previous projects, this task was completed using the *setAttr*() command and determining the specific attributes. When combined with the information display, jSequencer was starting to look more like its intended design. (See Figure 5.5)

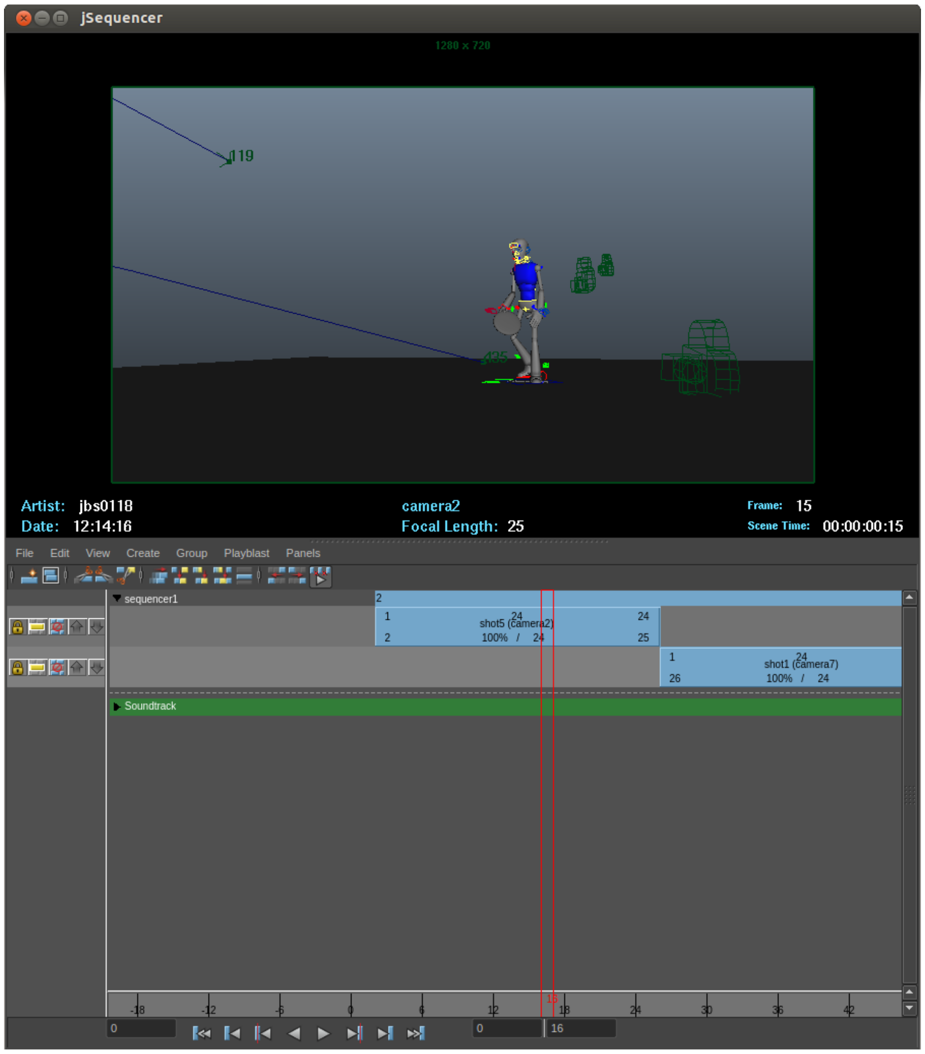


Figure 5.5: jSequencer prototype with the resolution gate.

5.1.6 Final Design Adjustments

Once the initial prototype was completed, I finished the design of the tool and fixed any other issues by scripting and editing flags. MEL commands such as paneLayout() and headsUpDisplay() have *flags* that are edited to fit specific designs. In MEL, flags are part of the commands and modify how they work. Apart from these scripting issues, other adjustments were needed to facilitate the animatic creation and Maya scene setup process and are discussed at the end of this section.

To get the desired design, and make the viewport an effective size with the resolution gate and heads up display turned on, the pane layout was adjusted. As explained in Section 5.1.3, using pane layout provides the ability to adjust the size of the panes after the script runs. Editing pane size was done using the "*paneSize*" flag in the paneLayout() command resulted in the viewport display take up 65% of the total window. Figure 5.6 shows how this adjustment to the pane size makes jSequencer appear closer to the original desired design.

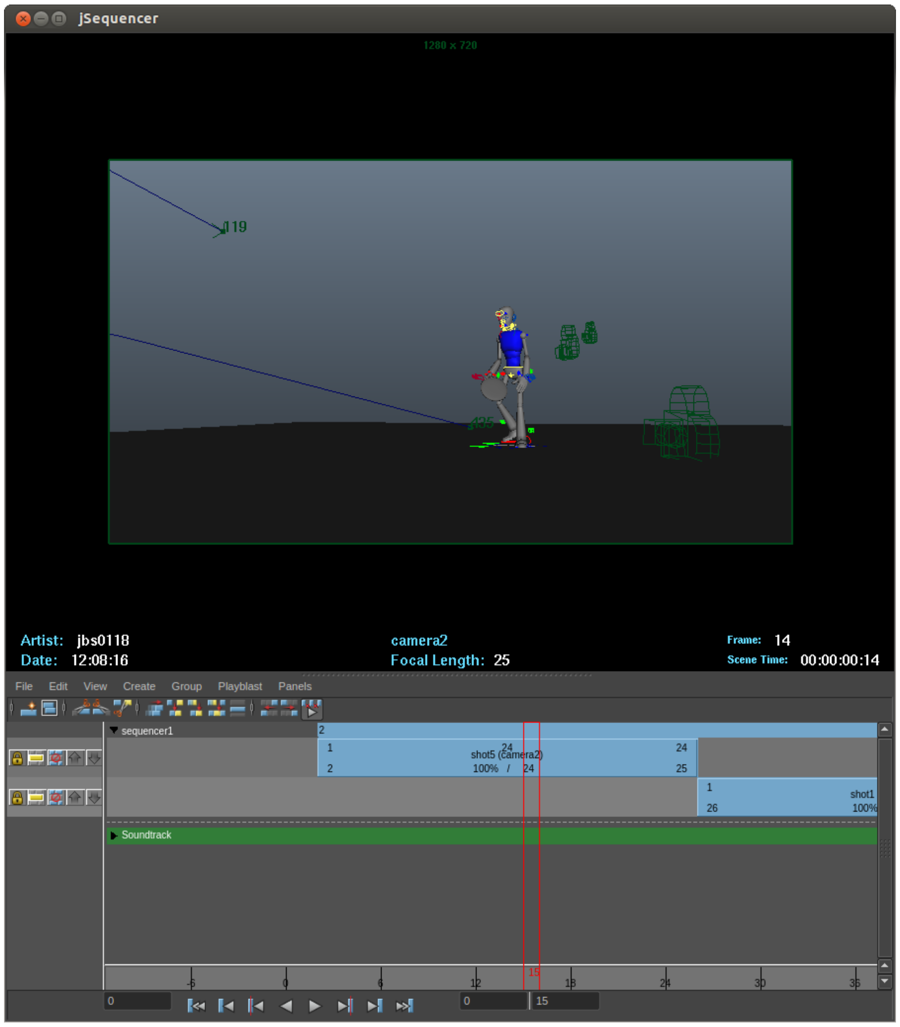


Figure 5.6: jSequencer prototype with an adjusted pane layout

In the initial prototype, the resolution gate and surrounding gate mask were activated for the selected camera when the tool was created. When switching to the next shot in the Camera Sequencer, the next camera did not have the attributes turned on. Since the goal was to have the script adjust every camera used in the scene, modifying the script so that it sets the attributes of every camera to be the same was very important. The solution was putting the *setAttr*() commands in a "for loop" that runs through all the cameras in the scene. The script finds every object in Maya that is labeled as a camera and automatically adjusts its attributes. This assigned the setAttr() commands to every camera, and saved the time needed to manually set the commands for each camera.

However, an issue surfaced during playback in the jSequencer. As the sequence played, cameras may have different viewport sizes and film gate values, resulting in jarring differences between cameras. Checking the Maya documentation and camera attributes, I discovered the issue was the overscan values. The overscan value is what adjusts the gate and mask size in the viewport. The default value automatically selects a horizontal and vertical fit so that the selected image fills the render frame. Like the other attributes for the cameras, using the *setAttr*() command was the easiest solution to correct for this. Putting this new attribute adjustment in the "for loop" forced all cameras to the same viewport dimensions and improved the video playback. As a result, the surrounding gate mask became more effective and the tool optimized the available window space. As seen in Figure 5.7, the camera frame goes to the sides of the window while still maintaining the resolution and mask on the top and bottom boundaries. This mask space allows the HUD to be more effective.

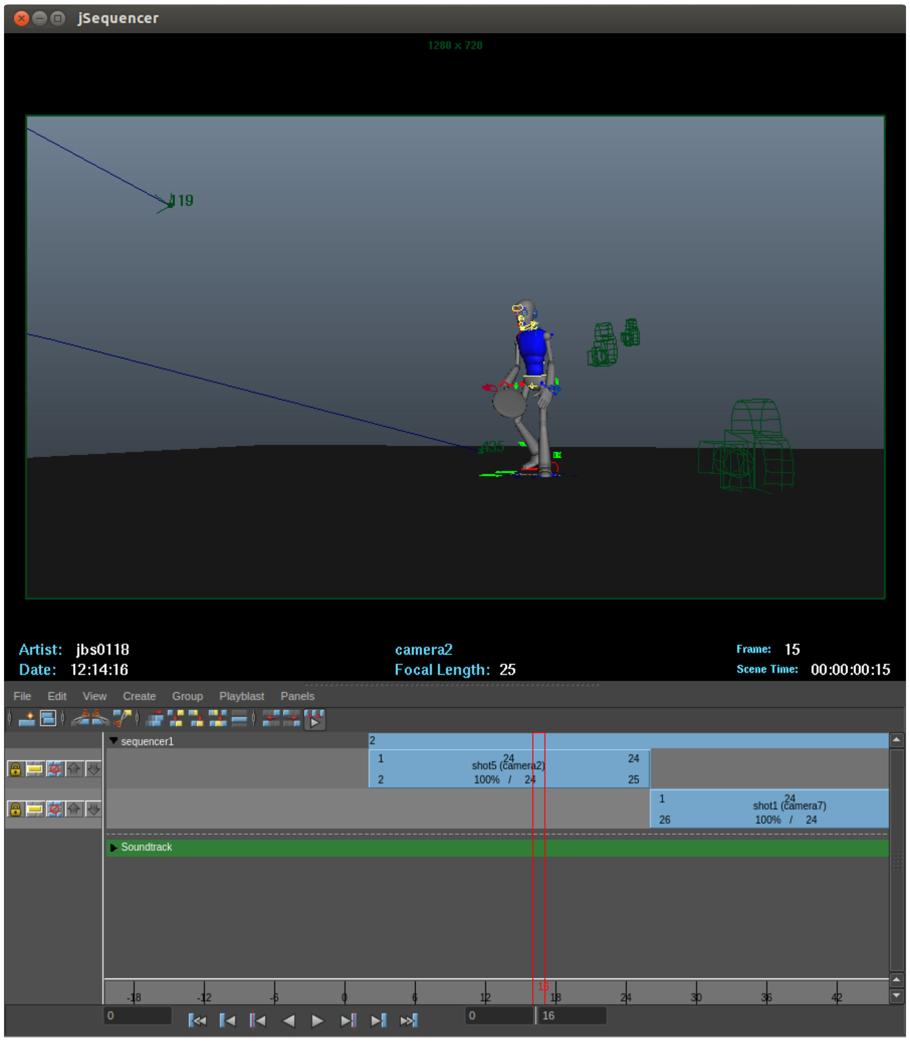


Figure 5.7: jSequencer prototype with an adjusted resolution gate

The viewport had another aesthetic problem due to the various objects cluttering the viewports and blocking the view of the animation. For example, NURB Curves (the shapes drawn in Maya commonly used as the controls for character rigs) are visible by default. In a scene with many characters, the NURB Curves and other objects can clutter the viewport and hide the action and camera composition. Visibility of these objects is controlled in the Show menu at the top of the model panel, where the user can select the types of objects they want to see in the selected viewport. As a default, every object type is visible. The user has to manually turn on/off the objects they want to see in the scene's viewport. Since the jSequencer uses a custom model editor without the Show menu, this problem was solved through additional scripting. Within MEL object visibility is controlled in the *modelEditor*() command flags. The flags used in the jSequencer are –cameras, -nurbCurves, and –joints. These flags turn off the display of the specific type of objects in the viewport. Figure 5.8 shows what a cluttered and an uncluttered viewport with many cameras, curves and other objects look like.

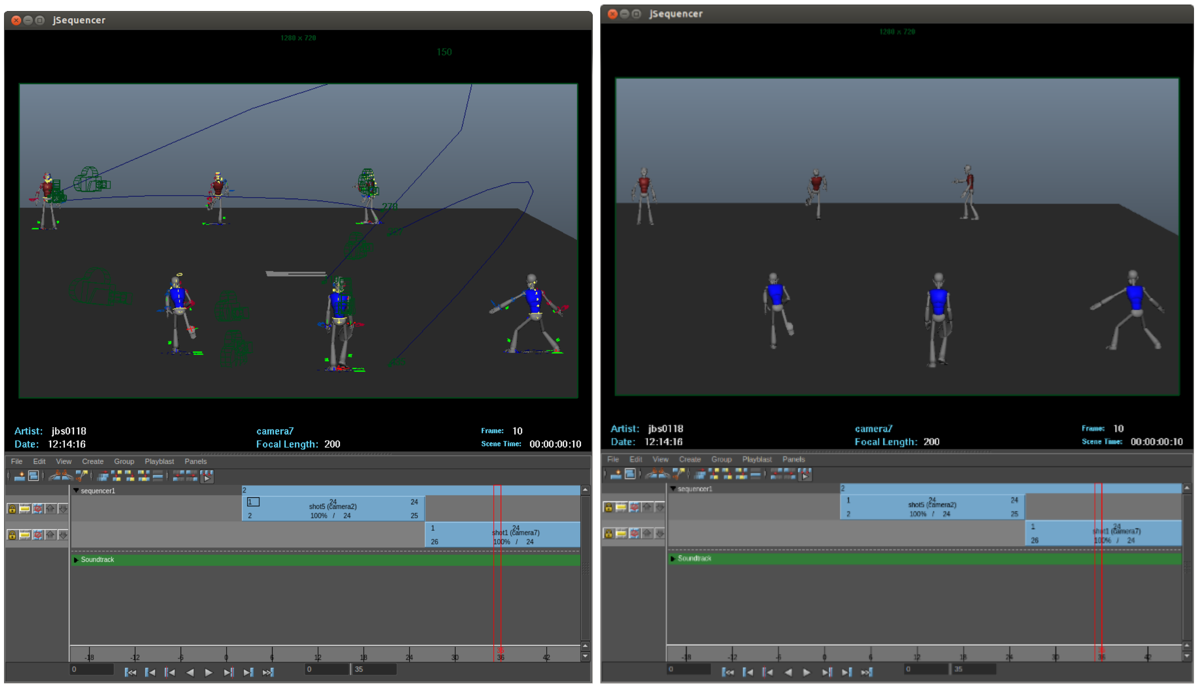


Figure 5.8: Comparison showing a view with different object visibilities. Objects' visibility turned on in the left, visibility turned off in the right.

An unexpected issue involved the Playblast tools in the Camera Sequencer. In the current version of Maya, the "Show Ornaments" option is missing from the options box of the Playblast tools under the Camera Sequencer menu. The "Show Ornaments" flag in the *playblast*() command turns on the Heads-Up Display during playback so it shows up in the resulting animatic. This option is available when using the Playblast tool in the Maya >

Create menu, however when using the Camera Sequencer, the option is automatically turned off. After extensive research, the easiest solution was to alter the flag in the base MEL scripts (doPlayblastSequenceArgList.mel and doPlayblastShotArgList.mel) to make the HUD display correctly. When the code is run in jSequencer, because the flag values are adjusted in global procedures, the old values in "Show Ornaments" are automatically overwritten.

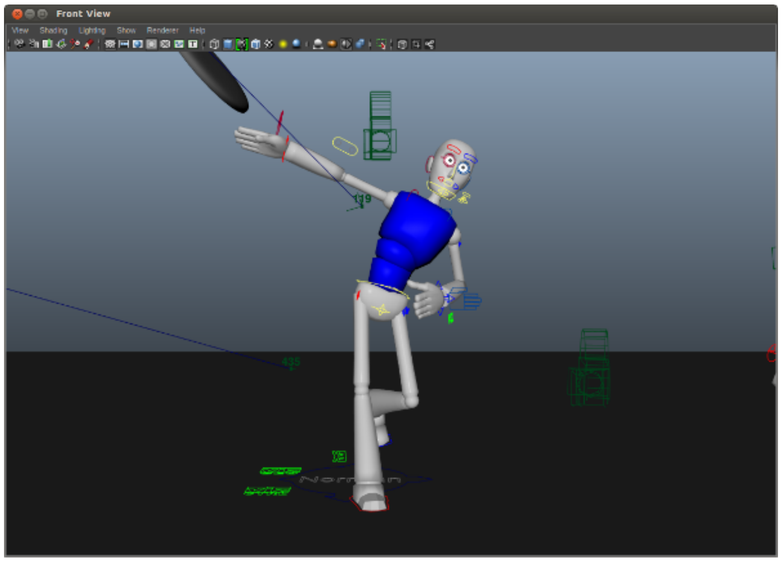
There were a number of minor scripting syntax errors made during the implementation of jSequencer. These were debugged in the script or solved by researching the specific errors and how to fix them.

5.2 Animatic Creation

5.2.1 Maya Scene Setup

The Maya scene setup depends on the user to create character animation and the cameras setups jSequencer can use. For demonstration of jSequencer, I created a simple character animation sequence with fourteen separate characters, all animated for 500 frames. Six of these characters have complex animations, while the others are in the background to represent movement and composition. Animation for these characters was done using basic keyframe animation techniques, with live action reference videos for guidance on timing and body placement.

Once the character animation was completed, I began setting up the cameras. In an attempt to mimic sports documentaries and films, multiple cameras were placed at angles and focal lengths that would capture the action. Like the animation, the initial camera placement and animation was done in the default Maya viewport. Once these cameras were placed, I was ready to start creating the animatic.



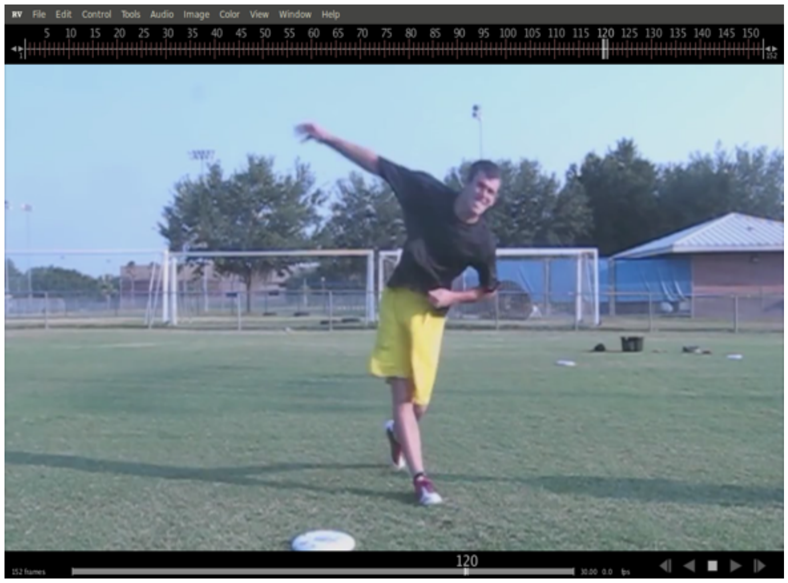

RV File Edit Control Tools Audio Image Color View Window Help
120

Figure 5.9: Screenshots of character animation and reference video.

5.2.2 Using jSequencer in Maya

To use jSequencer, the script has to be downloaded from the script/plugin database on Creative Crash, or highend3d.com. The script is then copied and pasted into the Maya Script Editor. Once the script is executed, entering "jSequencer" into the Command Line will activate the tool and the jSequencer window will pop up on the screen.

Adding the code into the Script Editor is advantageous because it will automatically save the state whenever the window is closed. As long as the code is still in the Editor's window, the script can be compiled without having to copy and paste it again. As long as the Maya session is on the same workstation, the only step is entering the same command and the jSequencer will be activated.

5.2.3 jSequencer Code Structure

The jSequencer script is comprised of three main parts: the GUI creation procedure (in MEL called *proc*), the display creation procedure, and display removal procedure. Procedures in MEL allow the user to collect a series of commands into a new function that will perform a new action. The procedures in the jSequencer script are global procedures, which can be called using the command line or by any other script, expression, or script node.

The GUI creation proc, called jSequencer(), creates the window layout, assigns the panels, adjusts the resolution gate and mask settings, removes the current display information and creates the new customized display. This proc is where most of the commands discussed in Section 5.1 are placed. The new display procedure (called jSeqHUD()) creates the HUD objects and places them in the designated sections and blocks vacated when the removeHUD() function is run. The removeHUD() function simply removes any heads-up display objects in the blocks and sections that are occupying the space new HUD objects will occupy. Both jSeqHUD and removeHUD are placed in jSequencer, so that they all run together. Once the script is compiled, the tool is created by entering *jSequencer* in the Maya command line.

The script runs and creates the jSequencer. The window pops up in the middle of the screen and is used with the current scene. The jSequencer is designed for a dual monitor workstation, so its most efficient placement is on the second monitor away from the main Maya window. In this setup the tool will not crowd the display and the user will have complete access to the main viewport to animate, model, or conduct any other type of work. This workspace management gives the user a place to edit the animation and immediately see how it looks in the jSequencer view.

To solve the Playblast tool issue in the default Camera Sequencer mentioned in Section 5.1.6, the installation scripts for Maya had to be altered. These MEL scripts are included in the jSequencer. Once the whole script is compiled in the Script Editor, the jSequencer is ready for use.

5.2.4 Creating and Adjusting Shots

By default, the Camera Sequencer creates a shot using the active camera in the current panel. The shot is inserted at the current sequencer playback time and sets the start and end frame range based on the Maya Time Range Slider. The Create > Shot option box in the Camera Sequencer Menu Bar provides the user more precise shot adjustments, like selecting the specific camera and setting the shot length. If the shot needs to be changed in any way, the attributes can be adjusted manually within the shot view area. The jSequencer sets the "current panel" to the modelEditor view in the window, so the shot being played is shown in the viewport. The jSequencer also provides the ability to see how the camera animation and compositions look when the sequence plays in real time. This capability makes editing the cameras dynamic and improves the animatic creation process.

I edited the animation and shot compositions by alternating between the jSequencer (to check the composition) and the perspective view (to edit the character and camera animation). This process continued until the cameras were finalized. For previsualization and layout animatics, the character animation isn't normally considered "final," but I wanted to make the animation on the main characters more dynamic.

5.2.5 Creating the Animatics

Once the cameras were set and animation was completed, the next step was creating the animatics. By default, the Playblast tool generates image sequences using the active view and current time range in the Time Slider to determine the animation range. The settings, such as file output and frame range, can be accessed in the options in the drop menu in the Camera Sequencer. The resulting animatic video is exported to the designated output file location (by default where the Maya scene file is located). The default Camera Sequencer and the jSequencer have different playblast processes. A more thorough explanation of these processes is discussed in Section 6.1.4

6. RESULTS AND EVALUATION

The results from this project compare the default Camera Sequencer to the jSequencer. The project aimed to improve the animatic creation capabilities, including the Maya workspace management, the character and camera animation editing process, and use of the Playblast and Heads-Up Display options. The following sections will discuss the comparisons between the default Camera Sequencer and jSequencer. The animatics use character animation created before the cameras were created and roughly placed in the scene. Then the animation and cameras were edited using both the default Camera Sequencer and jSequencer.

6.1 Comparisons

6.1.1 Workspace Management

The first comparison made was of the interfaces and workspace. When using the default Camera Sequencer tool, I had to either place the panel in the Maya display interface or "tear off" the panel into a separate window. To place the panel in the window, an acceptable panel layout must be selected from the Maya menus. I chose the quad layout preset to be able to have an effective layout. The Camera Sequencer takes up the bottom right panel, with the perspective view taking a majority of the display in the top right panel, the outliner in the top left, and a camera viewport in the bottom left (see Figure 6.1). When animating, the Graph Editor display was more needed than the Camera Sequencer, because being able to adjust the animation keyframes and components was more important. When I wanted to see how the animation would look in sequence I would have to switch the panels to playback the scene.

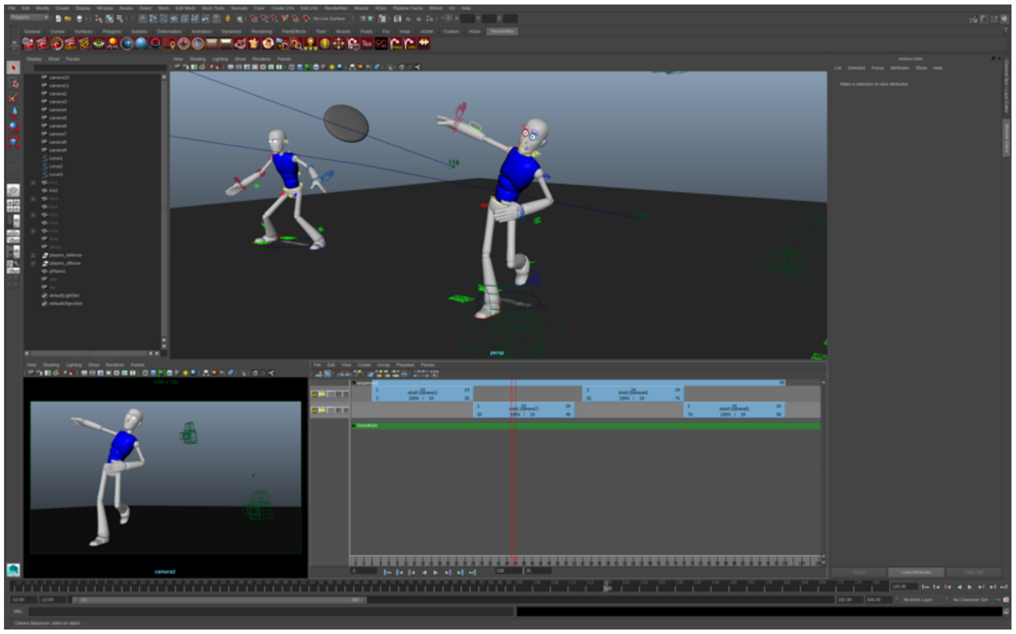


Figure 6.1: Maya workspace with the default Camera Sequencer

The jSequencer creates a new window with the Camera Sequencer and a dedicated viewport display. This creates a “tear off” copy of the Camera Sequencer, with the added benefits of the designated viewport and shot information display (which will be discussed in later sections). jSequencer is best for dual monitor workstations, so that the user can have the main Maya display on one screen and the jSequencer window on the other. This gives the user the freedom of having other types of panel layouts in the main Maya window, while still having the Camera Sequencer panel and camera view visible.

When I activated the jSequencer, I set up the main window with a large editing view and Graph Editor. If the user only has one monitor, the tool still works effectively, but adjustments, such as making the windows smaller so that both displays fit the screen, have to be made to the interfaces. Figure 6.2 is a screenshot of my workstation at Texas A&M University’s Viz Lab. The second monitor has smaller resolution, but is large enough so that tool still works effectively. jSequencer was designed for the workstations at Texas A&M

University's Viz Lab. If a user has monitors with different resolutions the height and width dimensions of its window can be altered in the script before execution.

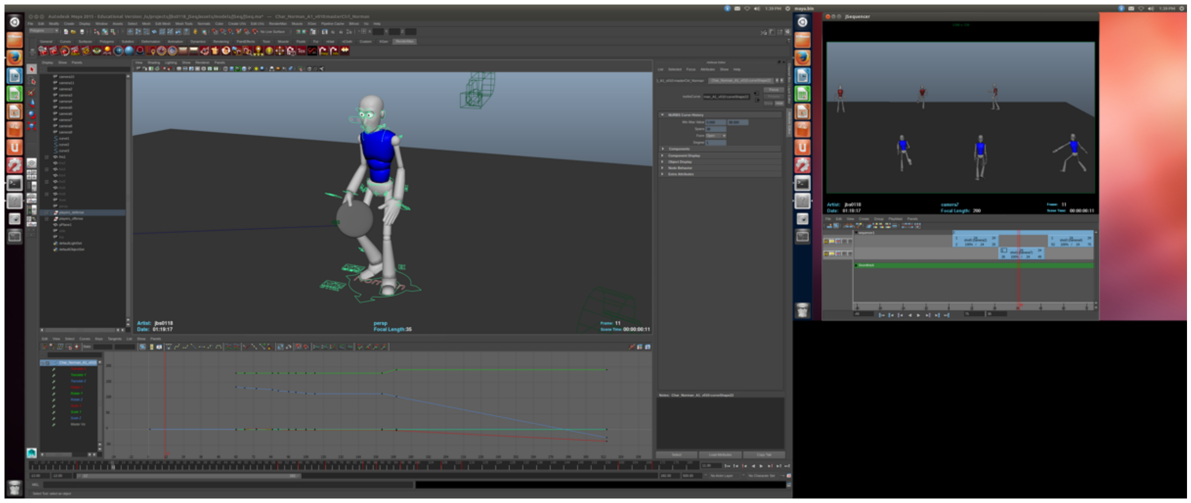


Figure 6.2 Maya workspace with jSequencer

6.1.2 Animation Editing

Inefficiency in the animation editing process was one of the main problems the jSequencer was meant to solve. When animating, there is a lot of back and forth between the main editing window and the Camera Sequencer to check the animation in the camera view. In the default Camera Sequencer, shot playback automatically uses the active viewport, or the viewport last selected, to show the sequence. If the main viewport is accidentally selected, the Camera Sequencer takes over the panel and changes the camera. The panel then has to be manually changed back to the original editor view, and the user must make sure to select the camera viewport before using the Camera Sequencer again. This process also has to be done when switching between the Graph Editor and Camera Sequencer panels. As mentioned in Section 6.1.1, whenever the user wants to edit the animation, they need to open the Graph Editor to edit the animation curves and keyframes. To see the shot sequence, then the user has to switch the panel back to the Camera Sequencer. Requiring these extra steps during the process of editing animation and reviewing in the Camera Sequencer quickly adds time to the animatic creation process and causes frustration for the user.

The jSequencer facilitates this process by creating an entirely separate window from the main Maya interface. When the jSequencer window is selected, Maya designates the viewport as the “active panel”, which means that the sequence will playback in the jSequencer viewport rather than in the main Maya window. The Camera Sequencer panel is placed in the jSequencer, and allows the user to place the Graph Editor or other scripted panels in the main interface. This solves the viewport selection and reassignment problem, and, as mentioned in Section 6.1.1, frees up space in the Maya display for other panels. During playback in the jSequencer window, the sequence only plays in the designated viewport. When the playback is paused, the main display updates automatically. When the animation is altered in the editing view, the jSequencer viewport updates in real-time. This makes the animation editing and review process more efficient and effective.

Figure 6.3 below shows the jSequencer being effectively used for character and camera animation editing. The Camera Sequnecer panel and camera view are separated in the jSequencer window with the main Maya interface having a large editing viewport and Graph Editor.

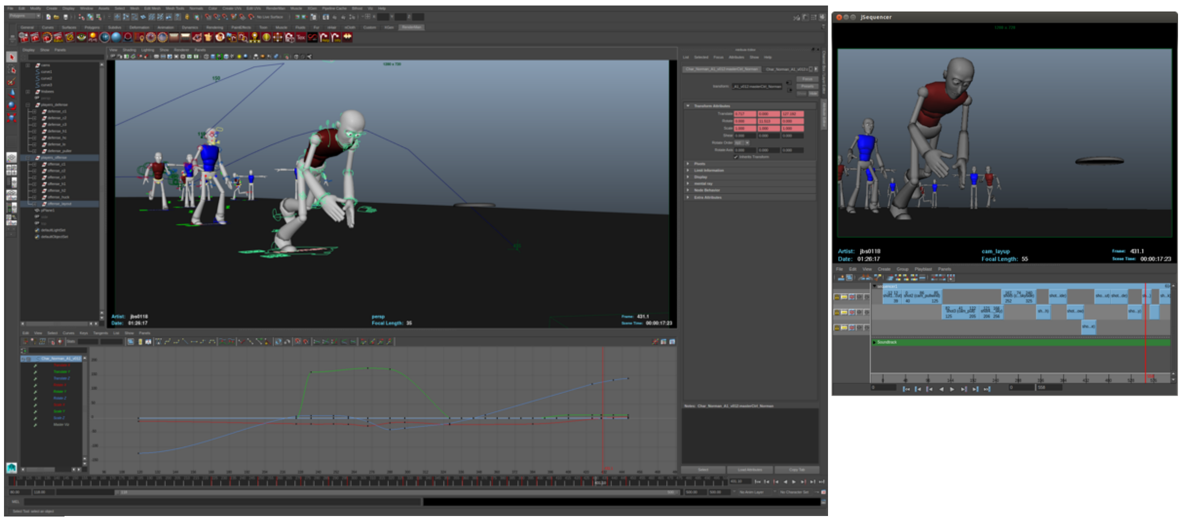


Figure 6.3 The animation editing process with the jSequencer

6.1.3 Animatic Displays

When using the default Camera Sequencer, the user has to manually adjust the Maya interface to have shot information and objects displayed in the viewport and activate the resolution gate mask. The information displays are small, difficult to read and grouped together, making them less effective and taking up too much screen space. Activating the Resolution Gate can help the visibility of the shot information, but they can still be difficult to review on smaller screens. The resolution gate mask has to be turned on and adjusted in each camera's attribute editor. Objects to be displayed in the viewport are selected using the individual panel's "Show" menu. However, these adjustments must be made before playblasting the animatic.

Using the jSequencer solves these issues by automatically adjusting the camera attributes and object visibilities, and by creating the desired information display. The shot information text is larger, easier to read, and evenly placed on the bottom of the frame. As mentioned in Section 5.1.4, the elements in the information display are pre-selected for their usefulness during the review process. The resolution gate mask creates the letterbox effect and provides a background for the shot information. Display objects, like curves and cameras, are turned off and create an uncluttered view of animation and composition. All of this is done automatically when the jSequencer is activated, saving considerable time during the animatic creation process.

6.1.4 Playblasting Capabilities

As discussed in previous sections, there are issues when playblasting animatics with the default Camera Sequencer and Maya interface. The default Camera Sequencer's playblast tools cannot create animatics that have the resolution gate mask or information display. Neither the "Playblast Sequence" or "Playblast Shot" tools include the "showOrnaments" box in the options windows, which turns on the Heads-Up Display during playback. However, when using the main Playblast tool in the Maya > Create menu, this option is available and works correctly.

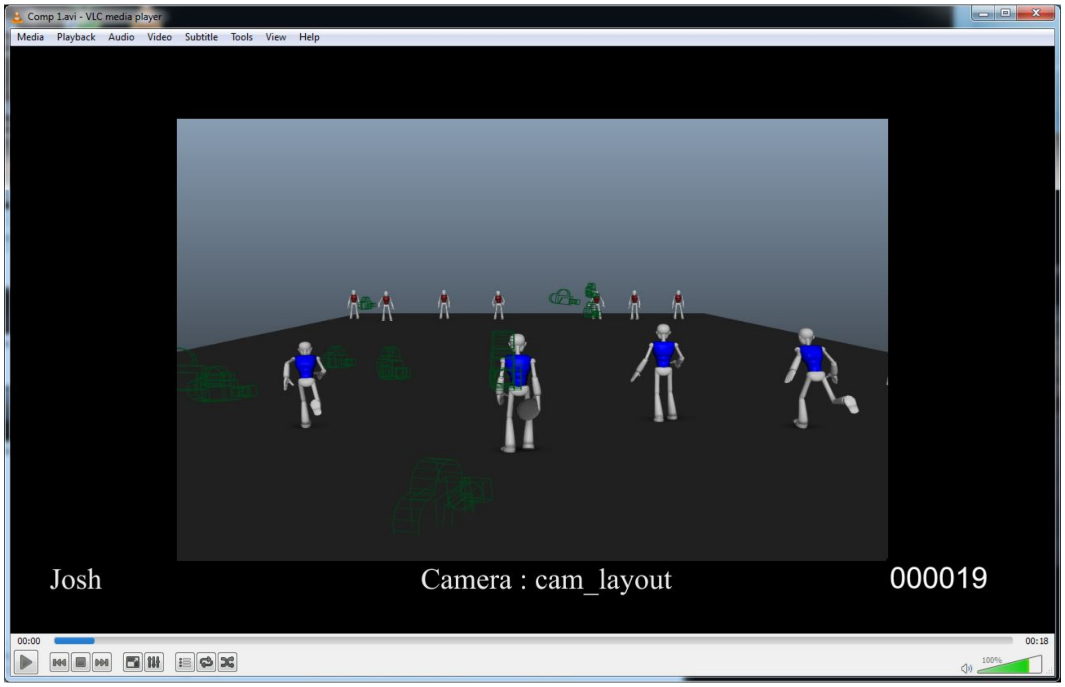
The jSequencer corrects this issue by altering the "showOrnaments" flag in the Camera Sequencer playblast installation scripts. The Playblast options in the Camera Sequencer remain the same, yet the correct value for "showOrnaments" is in place when the tool is activated and the animatics are created. The resolution gate mask and information display show up correctly in frame, and there is no need to use the other Playblast menus to create the animatics.

6.1.5 Finalizing the Animatic

The final comparison is made using both processes to create a final animatic. When using the default Camera Sequencer, the user has to export the playblasted sequence into a separate video editor such as After Effects to add the desired shot information. Once in After Effects, adding information like shot name, camera name and focal length is done by creating separate text boxes that have to be adjusted to appear in frame for each shot. For sequences with many shots and cameras, this can take a lot of time. Also, the dimensions of the imported playblast file have to be altered to allocate space on the bottom of the frame for the shot information. Once the video size is changed and shot information edited to appear on the correct shots in frame, the video is rendered. While the resulting animatic can have information not accessible in Maya, this process adds significant time and complexity to the animatic creation process.

Using the jSequencer playblasting capabilities automatically creates the final animatic. The letterbox effect is created by the uniform resolution gate mask across all of the cameras, and the shot information is displayed in an aesthetically pleasing way, without the distracting objects in frame. All of these actions are performed when the Playblast tools in the jSequencer panel is activated to create the animatic, saving the time needed for exporting and editing in another program.

Figure 6.4 shows the screenshots of the resulting animatics. The default Camera Sequencer animatic was created through After Effects and the jSequencer animatic was created completely in Maya.


Comp 1.avi - VLC media player
Media Playback Audio Video Subtitle Tools View Help
Josh
Camera : cam_layout
000019
00:00
00:18
100%

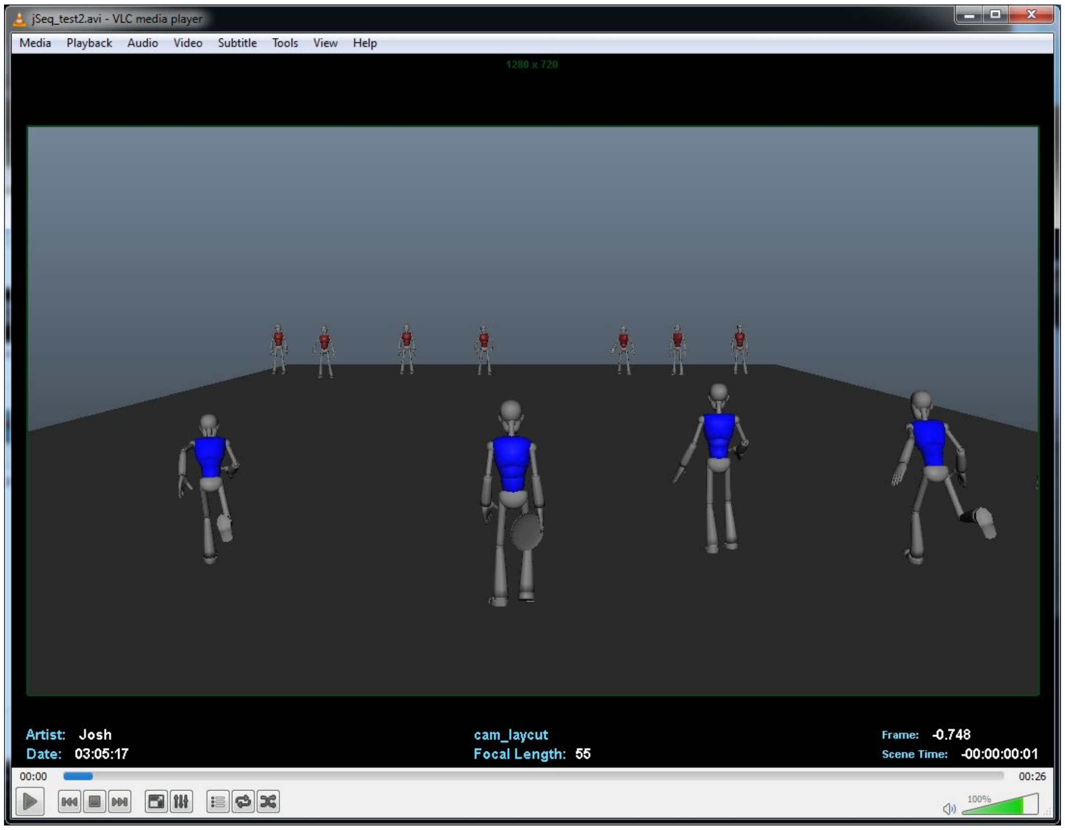


Figure 6.4 Screenshots of both resulting animatics. Top was created with Camera Sequencer and After Effects, while the bottom was created with jSequencer.

7. CONCLUSIONS AND FUTURE WORK

The default Camera Sequencer is a robust tool that is capable of creating useful animatics and shot sequences in Maya. However, the tool has several known issues that affect the efficiency of the animatic creation process. The user has to manually adjust the resolution gate, mask, and overscan values for each camera. The user also has to manually toggle which display objects are visible in each viewport. The Camera Sequencer plays the sequence in whichever viewport was last selected, resulting in time spent switching between viewports during animation editing. Animation editing also takes longer because of the back and forth between the Camera Sequencer and Graph Editor panels. Playblasts created from the Camera Sequencer cannot show shot information or have the resolution gate mask, and have to exported to a different software to add these missing elements. Editing the animatic in another program to include shot information adds considerable time.

The jSequencer tool design and functionality fixes these problems, and provides a more efficient and effective process to create animatics in Maya. The new tool keeps the main functionality of the Camera Sequencer, creating shot sequence animatics in a user-friendly interface, but adds elements that facilitate and improve the animatic creation, editing, and review processes. The jSequencer automatically adjusts all of the camera attributes and display object visibilities, and creates a custom information display. A designated viewport for playback separate from the main Maya window is provided, avoiding the hassle of viewport switching. The resulting playblast is the final animatic since it displays the resolution mask and shot information correctly, saving the time needed to export to another software. The jSequencer animatic display is much larger and cleaner with more information.

Maya's default Camera Sequencer has the basic capabilities to create animatics. The jSequencer tool enhances these functionalities and adds elements that improve the process effectiveness. jSequencer is therefore a better tool for creating animatics in the Maya workspace.

7.1 Future Work

The development of jSequencer provides a basis for possible future work. The tool itself could be rewritten in Python rather than MEL script. Python is primarily used for custom plugins that can be used and modified in many software packages. Python also has more direct access to C++ libraries and modules, which make reusing the code in other software easier.

A similar animatic tool could be developed for other software packages, such as Autodesk's *3dsMax* and *Motionbuilder*, SideFX's *Houdini* or Maxon's *Cinema4D*. 3DsMax has its own Camera Sequencer tool, but a better tool could be developed using the built-in script language MAXScript. The other packages lack a Camera Sequencer tool, so a basic sequencer could be developed to improve their animatic creation capabilities.

Providing the option to export the animatic sequence directly from the Sequencer into file formats for programs such as *RV* or *QuickTime* could be considered. If the sequence could be reviewed within RV or QuickTime the review of the animatics would be more efficient. This system could be setup within the Maya interface or added directly into the modified Camera Sequencer display. This would involve more precise editing of the Maya installation files and writing procedures to import and open the animatics in the programs.

This would be particularly useful for artists that use multiple shot sequences in one file and want to see alternate sequences.

REFERENCES

1. “Camera Sequencer Overview.” *Autodesk Knowledge Network.* 7 Sep. 2012. http://knowledge.autodesk.com/support/maya/learn-explore/caas/mne-help/global/docs/maya2013/en_us/files/GUID-756DCF5D-C74C-46E4-BE11-50E5C7FA3376-htm.html. Accessed 12 Sep. 2016.

2. Caron, Tammy. “Toon Boom Storyboard Pro Review.” 22 April 2014. http://www.imore.com/toon-boom-storyboard-pro-review. Accessed 12 Nov. 2016.

3. Desowitz, Bill. ‘A Previsualization Society Is Born.’ *Animation World Network*, 29 Sep. 2009. http://www.awn.com/vfxworld/previsualization-society-born. Accessed 12 Sep. 2016.

4. Hian Wong, Hock. ‘Previsualization: Assisting Filmmakers in Realizing Their Vision.’ *Siggraph Asia 2012 Courses.* 28 Nov. 2012.

5. Hogg, Trevor. ‘Precision Warfare: The Third Floor Visualizes’ Captain America: Civil War.’ *Animation World Network.* 30 June, 2016. http://www.awn.com/vfxworld/precision-warfare-third-floor-visualizes-captain-america-civil-war. Accessed 14 Sep. 2016.

6. “Introducing FrameForge Previz Studio Software.” *Creative Writing Software 101.com,* 2014. http://www.creativewritingsoftware101.com/graphics/frameforge/04-frameforgescreen.jpg. Accessed 12 Nov. 2016.

7. Jarratt, Brandon Lee. ‘From Production to Education: An Analysis of Pipeline Requirements and Practices.’ *Master’s thesis, Texas A&M University.* 15 Jan. 2013. http://hdl .handle.net/1969.1/149239. Accessed 10 Nov. 2016.

8. Lasseter, John. ‘Tell Me a Story’, in *Walt Disney Animation Studios – The Archive Series: Story.* New York: Disney Editions, 2008.

9. Lockwood, Noah and Patrick Coleman, Patricio Simari, and Karan Singh. *DirectCam: A Gestural System for Animatic Creation.* Poster presented at ACM SIGGRAPH Conference, San Diego, CA, August 2007.

10. Marques, Alan. “iClone 6.1 Review.” 4 June 2015. https://www.3dtotal.com/interview/558-iclone-61-review-by-alan-marques-keywords-reallusion. Accessed 12 Nov. 2016.

11. McEachern, Martin. ‘Orchestrators of the Dream.’ *Computer Graphics World.* Oct. 2013. http://cgw.com/Publications/CGW/2013/Volume-36-Issue-6-Sept-Oct-2013-/Orchestrators-of-the-Dream. Accessed 14 Sep. 2016.

12. “MEL for Programmers.” *Autodesk Knowledge Network.* 11 May 2016. http://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/Maya/files/GUID-0F7C50D1-FF45-4868-8EBC-FE044F54B82E-htm.html. Accessed 4 Nov. 2016.

13. “MEL Command Reference.” *Autodesk Knowledge Network*. http://help.autodesk.com/cloudhelp/2017/CHS/Maya-Tech-Docs/Commands/index.html. Accessed 16 June 2016.

14. “An Overview of Previsualization Software and Methods.” *Wolfcrow,* http://wolfcrow.com/blog/an-overview-of-previsualization-previz-software-and-methods. Accessed 12 Sep. 2016.

15. Paik, Jiwon and Cheeyong Kim. ‘Research on Disney’s 3D Animation *Wreck-It Ralph’s* Style, Layout Pipeline, and Camera Capture System.’ *Journal of Korea Multimedia Society.* Vol. 16, no. 11, Nov. 2013.

16. Pallant, Chris and Steven Price. *Storyboarding: A Critical History.* New York: Palgrave Macmillan, 2015.

17. “Playblast an Animation.” *Autodesk Knowledge Network.* 26 Oct. 2016. http://knowledge.autodesk.com/support/maya-lt/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/MayaLT/files/GUID-1C6EDC8D-DA67-490E-81F1-1205336DEBD9-htm.html. Accessed 15 Dec. 2016.

18. Pohl, Brian. ‘Previs: Love It or Leave It…It’s Here to Stay.’ *Animation World Network*. 4 April, 2011. http://www.awn.com/blog/previs-love-it-or-leave-itits-here-stay. Accessed 15 Sep. 2016.

19. Ross, Aaron. ‘Production Rendering From the Maya Camera Sequencer.’ *Lynda.com-Article Center*. 1 April, 2014. http://www.lynda.com/articles/production-rendering-maya-camera-sequencer. Accessed 15 Sep, 2016.

20. Schauer, Bradley. ‘The Auteur Renaissance, 1968-1980.’ In *Cinematography*, edited by Patrick Keating. New Brunswick: Rutgers University Press, 2014.

21. Seal, Joshua, layout lead. *Frankenkite.* Texas A&M University Department of Visualization, 2015. http://vimeo.com/168993074. Accessed 30 June, 2016.

Supplementary Sources

Primary Plugin References

“animHUD 1.0.5 (maya script).” *Creative Crash*. 6 April 2010. https://www.creativecrash.com/maya/script/animhud. Accessed 10 July 2016.

Ewert, Bryan. “How Do I Dock Maya’s Graph Editor, HyperGraph, Render View, et al, in My Own Window?.” *MEL How-To*. 5 Aug. 2002. http://ewertb.soundlinker.com/mel/mel.053.php. Accessed 16 June 2016.

“scHUDAnim.mel 1.2.0 (maya script).” *Creative Crash*. 22 May 2012.https://www.creativecrash.com/maya/script/schudanim-mel. Accessed 10 July 2016.

“shotView 2.3.0 (maya script).” *Creative Crash*. 2 Feb. 2011. https://www.creativecrash.com/maya/script/shotview. Accessed 16 Nov. 2016.

Animation Reference Videos

TheAUDLChannel. “Championship Weekend V | Top 10 Presented by Spinlister.” Online video clip. YouTube. YouTube, 15 Aug. 2015. https://www.youtube.com/watch?v=ImPM3NOpch8. Accessed 1 Sep. 2016.

Smith, Brodie. “How to Throw a Forehand Far | Brodie Smith.” Online video clip. YouTube. YouTube, 18 April 2011. https://www.youtube.com/watch?v=NM3tVxL4Q70. Accessed 25, Aug. 2016.

Smith, Brodie. “How to Throw a Frisbee Far | Brodie Smith.” Online video clip. YouTube. YouTube, 20 Jan. 2011. https://www.youtube.com/watch?v=Trm_XZR3dA0. Accessed 18, Aug. 2016.

APPENDIX A

MEL Scripting

MEL scripting provides access to the base commands in Maya. Different types of MEL commands and parameters were used in jSequencer implementation. Below is are explanations of the MEL commands used in the script. All definitions are from the MEL command reference in the *Autodesk Knowledge Network.* [26]

MEL Commands Used

The following commands are listed in the order that they appear in the script. Not included are the commands used in the MEL installation procedures that are copied into the jSequencer file.

window() - This command creates a new window but leaves it invisible. When combined with the showWindow() command, a new blank window is created. Everything that is inside the window has to be defined between these two commands.

paneLayout() - This command creates a pane layout. A pane layout may have any number of children as determined by the current configuration. In the jSequencer script, a horizontally split pane is used with a visible separator between the two. The dimensions of the children can be altered by this separator or in the code.

formLayout() – This command creates a form layout. A form layout allows absolute and relative positioning of the controls that are its immediate children. There is no default positioning relationship so to have children appear in the form they must have at least one edge attached in each direction. This command is present in the script, but it is commented out because it is not the layout used.

getPanel() - This command returns panel and panel configuration information. This function includes the flag -scriptType and returns the scripted panel “sequenceEditorPanel”.

scriptedPanel() - This command will create an instance of the specified scriptedPanelType. A scripted panel is a panel that is defined in MEL, with all the required functions and commands available as MEL’s procedures.

modelEditor() - This command can create, edit or query a model editor. This is used to create the separated viewport in the jSequencer. The signficant difference between this and modelPanel(), is that the modelEditor() only includes the viewport without the tool and menu panels provided by modelPanel().

sequenceManager() - This command is used to manage sequences, shots, and their related scenes. This is used to specify the dedicated panel that the sequencer will control.

setAttr() - This command is used to set attributes for objects in the scene. For example, setting the position of the camera. MEL script is especially useful with this command because the script editor echoes the commands when the user edits something in the viewport. This way the user can see the specific attributes they want to edit.

setParent() – This command changes the default parent to be the specified parent. One special parent string “..” indicates one level up in the hierarchy. A control, menu, or panel must be parented to a control layout or window.

displayColor() - This command changes or queries the display color for anything in the application that allows the user to set its color.

headsUpDisply() - This command creates a Heads-up Display (HUD) object which is placed in a 2D inactive overlay plane on the 3D viewport. It is used to display information designated by a user script. The only mandatory flags on creation are the section and block flags, which designate where the object appears in the viewport.

All of the commands have several flags. Flags modify how a command works. A flag comes after the command name, is proceeded by a dash (-), and is followed by a parameter. It is through the use of flags that options and values in the commands are edited in the following example.

```
paneLayout -configuration "horizontal2" -paneSize 1 50 65;
```

A pane layout is created with two panes split horizontally, the top pane taking up 65 percent of the total layout.

APPENDIX B

Software Packages

These are brief descriptions of the software packages referenced in this paper excluding Autodesk Maya and the other pre-visualization packages described in the background section.

Adobe After Effects – After Effects is a digital visual effects, motion graphics, and compositing application used in the post-production process of film making and television production.

Adobe Premiere – Premiere is a timeline-based video editing software application.

Autodesk 3dsMax – 3dsMax is a professional 3D computer graphics software for making 3D animations, models, games and images.

Autodesk Motionbuilder – Motionbuilder is a professional 3D character animation software used for virtual production, motion capture animation, and traditional keyframe animation.

Maxon Cinema4D – Cinema 4D is a professional 3D computer graphics software primarily used for motion graphic animation.

SideFX Houdini – Houdini is a professional 3D computer graphics software that exclusively uses a suite of procedural generation tools. Its primary uses are for procedural modeling and visual effects. It is also bundled with its own renderer, *Mantra*.

APPENDIX C

jSequencer Script Installation

jSequencer will only work for Maya 2011 and newer versions, because it depends upon the Camera Sequencer panel which was introduced in the 2011 version. Without that panel, the jSequencer will not work. Installing the script into the user's Maya workspace requires a few simple steps, provided by the read-me text file on the download page. These steps are described in detail here.

1 Go to the website highend3d.com and search for jSequencer in the Maya Scripts/Plugins section. If not found, check in the "Free" section of the same area.

2 Once the page is found, download the files.

3 Copy the entire MEL script into the Script Editor in Maya.

4 Execute the script in the Script Editor. This will compile all the procedures in the script and make them usable commands in MEL.

5 Input the command "jSequencer" into the Command Line in the Maya interface.

6 The jSequencer window will pop up and be ready for use.

The "jSequencer" command has to be entered into the Command Line each time you open a new session of Maya. The Script Editor automatically saves its state, or what was in the editor when the window was last closed, so the jSequencer code will still be there. The sequence created and the camera attributes and display objects are saved from the previous session, so activating the tool will bring up the same window as before and be ready for use.

If the command does not work during the new session, follow the same instructions above to activate the tool.