12-2011

Rigging and Texturing Considerations for the Short Film Spider Fight

Casey Johnson
Clemson University, casey5@clemson.edu

RIGGING AND TEXTURING CONSIDERATIONS FOR THE SHORT FILM *SPIDER FIGHT*

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Fine Arts
Digital Production

---

by
Casey Johnson
December 2011

Accepted by:
Dr. Timothy Davis, Committee Chair
Dr. Jerry Tessendorf
Dr. Brian Malloy

Abstract

This paper delves into two separate production areas of the short film, *Spider Fight:* the approach used to solve the problem of rigging eyelids for 3d characters, and the pipelines that were implemented for texturing 3d assets. Typically, problems exist in the rigging of eyelids for 3d characters. Productions rely on a series of blend shapes, or a combination of joints and blend shapes, to produce satisfactory eyelid movement. *Spider Fight,* however, used a series of curves and joints to control the movement of the eyelids, which allowed a high degree of control over eyelid movement, and also fast editing if any problems were detected. Another challenging area for computer-animated productions is texturing 3d assets. In order to accommodate the mix of characters and props in *Spider Fight*, a pipeline was developed for props that included Photoshop and Maya, while a second pipeline was created for characters that involved UVLayout, ZBrush, and Photoshop. The end result was an efficient system, which produced satisfactory results starting from the creation of UV shells and ending with completed textures.

Table of Contents

Page

List of Figures

Chapter 1: Introduction

In the short film, *Spider Fight*, many areas of the production required special attention due to complexity or efficiency concerns. One such area with complexity issues was rigging eyelids. Another area with efficiency issues due to of repeated usage was texturing pipelines.

Rigging was a major concern because all of the animation in the film relied on proper rigging. If the rigging was not implemented correctly, the characters would not perform as expected. In addition, the rigging had to accommodate changes introduced after the start of animation without affecting animation already completed. Specifically, the rigging for the characters' eyelids had to be completed in a way that was efficient to set up, and was amenable to changes after animation. To meet the requirements of efficiency and flexibility, a curve-based rigging system was implemented to control the eyelids of the characters in *Spider Fight*.

Texturing was a second major concern because of the sheer number of 3d objects that required textures. Naturally, some of the 3d objects would require more texturing work than others. In general, the props required less texturing work than the characters. A great deal of time was saved by streamlining through developing separate pipelines to accommodate various 3d objects. Two texturing pipelines that were developed for this project will be discussed in detail. Additionally, the curve-based rigging system for character eyelids will be presented and explained thoroughly.

The remainder of this paper will expand upon and discuss the topics mentioned. Chapter 2 will give an overview of rigging and blend shapes, preview the eye-rigging approach used in *Spider Fight*, and cover texture-mapping concepts. Chapter 3 will explore the eye-rigging system and texture pipelines used in the short film. Chapter 4 will discuss the results of the curve-based eye rigs as well as the two texturing pipelines. Finally, Chapter 5 will evaluate

lessons learned during the production, including the benefits and disadvantages of the rigging and texturing practices used in *Spider Fight*.

Chapter 2: Background

In computer animation, rigging refers to binding an object's mesh to a joint-based control structure. Maya provides two options for binding an object to a rig: smooth bind and rigid bind. The difference between these two methods is based on the way individual points are weighted to skeletal joints. In a rigid bind, the points on an object are weighted 100 percent to the closest joint in the skeletal hierarchy. In a smooth bind, the points in an object are weighted across a specific number of joints designated by the artist. The result of each bind is immediately obvious when the joints are rotated, as shown in Figure 2.1. An object that has been bound using rigid bind will move only the points nearest to the joint, which will cause the geometry to shear near the joint. An object that has been bound using smooth bind move the points over the joint, but will have a fall-off effect for all of the points that are just outside of the joint. A real-world example of each of these methods is a robot as a rigidly bound object, and a human as a smoothly bound object.

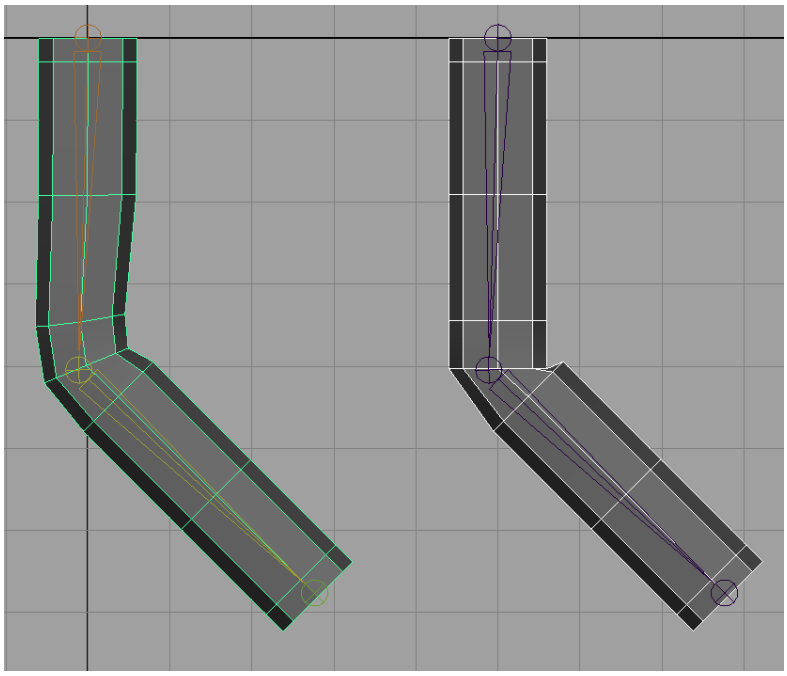

Figure 2.1: Smooth bind (left); rigid bind (right).

While joints can control an object, objects can also control other objects by establishing child-parent relationships. An important difference exists between binding an object to a joint-based control structure and parenting an object to another object. An object that is bound to another object will have all of its points weighted to the joint hierarchy, which is its only means of transformation. When an object is parented to another object, the pivot point of the object will follow all transformations performed on its parent object, but can still be translated, rotated, or scaled, without having to perform any such operation on its parent [LUHT10 pg. 110]. Also, when an object is parented to another object, the child object utilizes its original pivot point for local transforms, but when an object is bound to a joint hierarchy, the bound object's pivot point no longer has any influence over the object's transformations. Once an object has been rigged, the object can no longer be manipulated with the translate, scale, or rotation keys: object transformations must now be performed by translating the rig. The bound

object's geometry, however, can still be manipulated in a few different ways that do not involve the use of the joint hierarchy.

2.1 Blend Shapes

One method for moving geometry points without joints is blend shapes. Blend shapes can be used to shift the object's points, which is useful for creating facial expressions, such as smiling, frowning, or wincing around the eyes. Blend shapes are also useful for correcting problem areas that arise from moving the rig. A couple of common examples include adjusting the fat under the chin of a human character when the mouth is opened, and correcting elbow-pinching in a human character when the forearm is moved.

Blend shapes, as shown in Figure 2.2, are a set of poses, such as facial expressions, that can be used as targets in animating a sequence in a semi-automatic way. Blend shapes typically solve only one specific problem. As a result, if the artist wants to create ten different expressions for the face, ten specific blend shapes must be produced. Multiple blend shapes can be used in combination, but if they operate on the same points in an object, conflicts may arise over the movement of overlapping points. Using two blend shapes together, shown in Figure 2.3, results in double deformations along the middle of the shape, producing a deep divide.

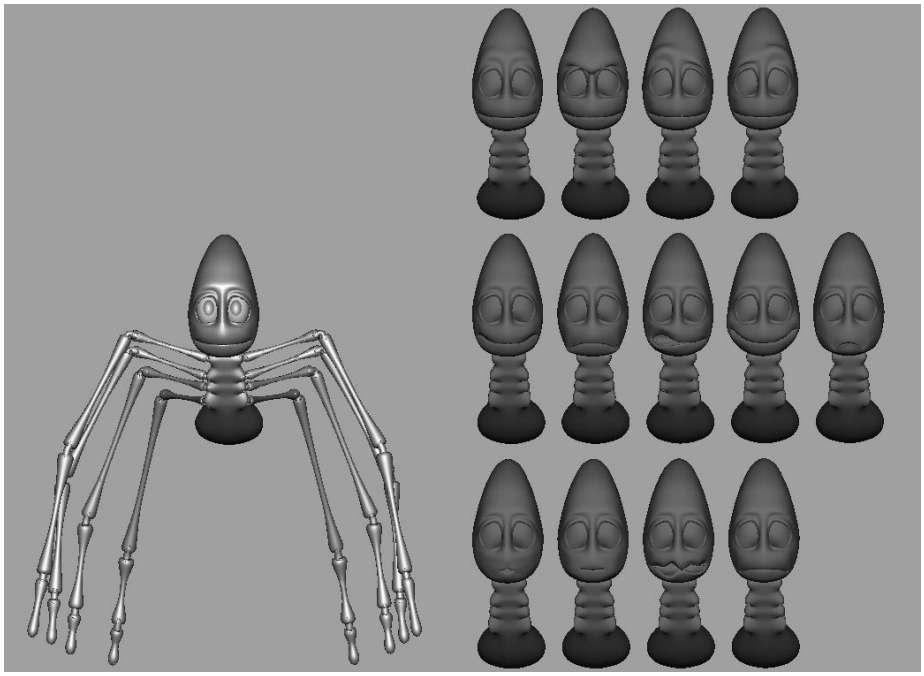

Figure 2.2: Character with blend shapes.

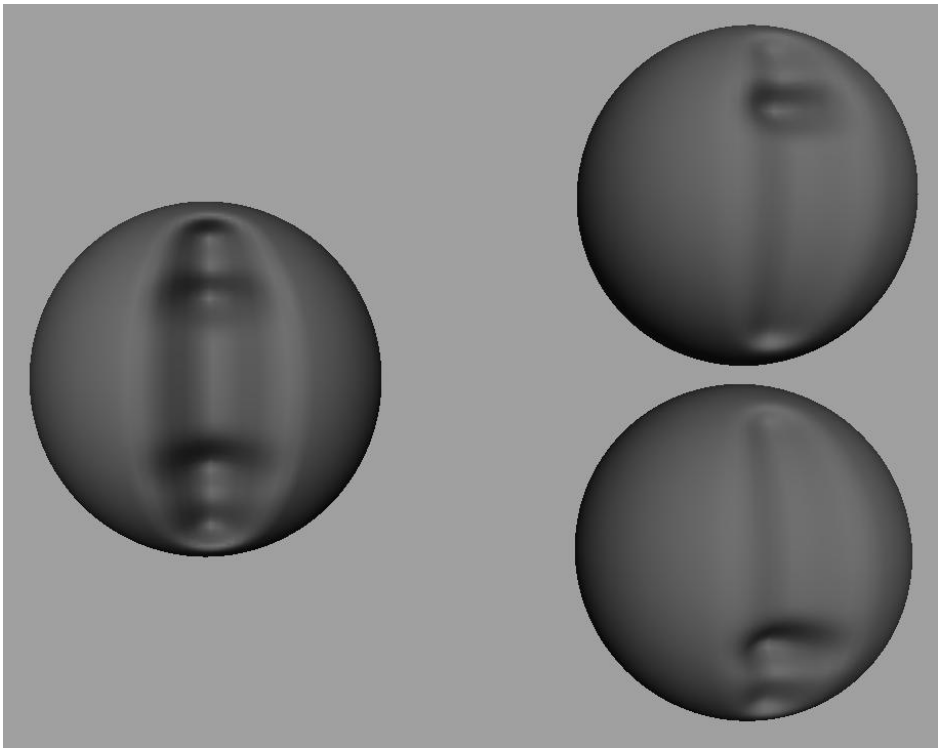

Figure 2.3: Using two blend shapes, shown at right, which both affect the center points of the geometry, produces a multiplied deepening effect if used simultaneously, as seen on left.

Another method for manipulating points on a rigged object is to use clusters, as demonstrated in Figure 2.4. Clusters allow an artist to control a specific set of points on an object. Unlike binding an object to a joint hierarchy, which weights all the points of an object to the joint or joint hierarchy, clusters bind only selected points on an object. One advantage of this method is that the artist can manipulate the cluster, and hence the points of that cluster, in any way. Unlike blend shapes, which only blend between the original geometry and the blend shape, clusters can be freely transformed through any manipulation operation. These methods will be discussed further when rigging the spider characters.

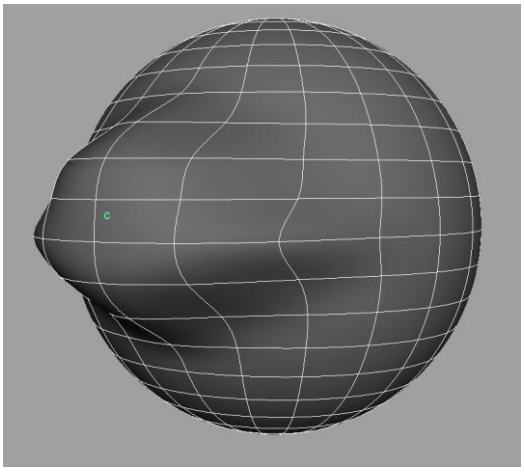

Figure 2.4: Object with a cluster.

2.2 Eye Rigging

An area that requires specialized rigging is the eyes, especially since the eyes convey a great deal of information about the character's mood. An especially difficult feature to model and animate is the eyelid. When rigging eyelids, special attention must be given to eyelid deformation when the underlying joint hierarchy is manipulated. In some cases, e.g., when the eye is perfectly round, two or more joints can be placed at the center of the round eyeball, as demonstrated in Figure 2.5, with the eyelids weighted to these joints [WARD05 pg. 620].

Several joints are necessary because the top and bottom eyelids must each be attached to at least one joint. Due to the position of these joints in the center of the eye, when the eyelid geometry is rotated around this central point, after being properly weighted, the eyelids will appear to open and close [OSIP03 pg. 252].

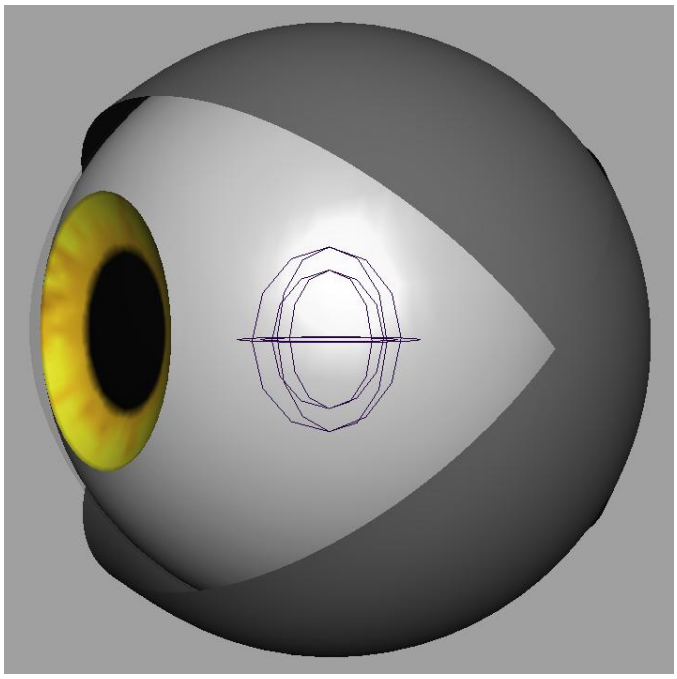

Figure 2.5: Eye with joints at the center.

Even if the eye is perfectly round, joints based in the center of the eye will work only in non-realistic setups, such as cartoons. The problem is that the eyelid needs to change shape as it rotates around the eyeball. The upper and lower eyelids must also conform in such a way that when the eyelids meet, they meet completely. If these considerations are not addressed, the eyelid geometry may overlap or leave a gap when the eyelids are closed.

If the eyes are not perfectly round, then the issues surrounding eyelid transformations become more complicated. If the character's eye is an oval shape, placing two joints for eyelid rotation becomes problematic. A great deal of difficulty is also introduced in ensuring the eyelids follow the shape of the eye, since a simple joint rotation will not work. To deal with

these problems, corrective blend shapes are often used [ALLE08 pg. 179]. These corrective blend shapes can be used in conjunction with rotating joints that control the eyelid geometry. Another approach to closing the eyelids, and hence blinking, is to use blend shapes entirely, and not bother with setting up joints to control rotation. In this case, several blend shapes are required, especially if the eye shape is oval, in order to prevent the geometry of the eyelids from penetrating the geometry of the eye.

Another consideration regarding the use of blend shapes to fix eyelid rotation is that the blend shapes are specific to a given model. If a model is changed after blend shapes are created, problems will arise with the deformations. As such, if a model is changed during production, the blend shapes must be completely recreated. Also, blend shapes cannot be transferred across multiple models; thus, if a production contains several similar human models, blend shapes must be set up for each model, since they rely on an identical number of points for the blend shape and the blend shape target.

A final disadvantage to blend shapes is they cannot be set up through a script. Blend shapes for eyelids must be visible to the artist to ensure that the geometry does not penetrate the eye geometry, and that the blend shape follows the geometry properly. As such, blend shapes require human interaction.

2.3 Texturing

Aside from rigging the eyelids, a second major consideration for the *Spider Fight* short was texturing. Texturing is the method by which various types of maps, such as diffuse and bump, are applied to the surface of geometry [KOND05 pg. 86]. An example of a texture map is shown in Figure 2.6. The texturing process involves both creating UV maps, and creating texture-based maps for the geometry. A UV map is a 2d layout of the points on a surface, and

once created, a texture map is assigned based on the UV layout. Depending on the outcome desired, the texture map brought into Maya can be mapped to any number of relevant channels, the most common of which is the color channel.

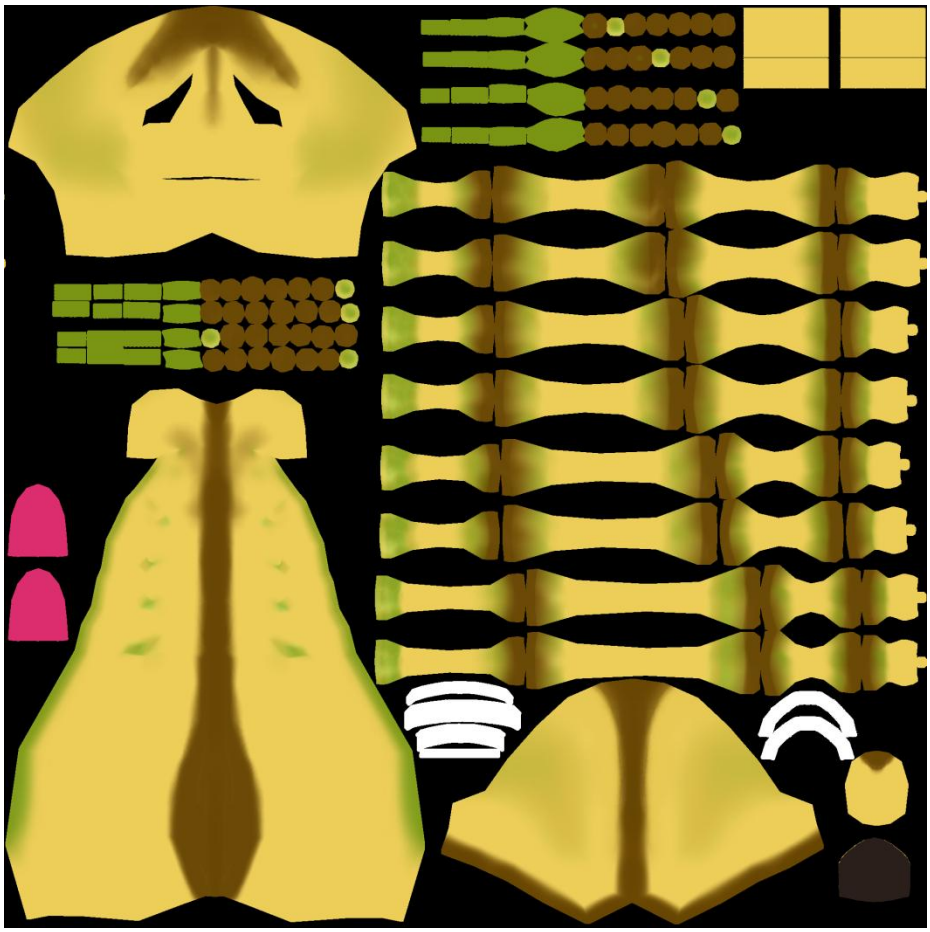

Figure 2.6: Texture map.

Geometry often must be separated into pieces to accommodate the application of a texture map. To texture a can of soda, for example, the top and bottom of the can of soda would be separated and flattened. The remaining cylindrical body of the can would be split from top to bottom, and unrolled such that the body of the can was flat. At this point the altered can would consist of two flat round circles (the top and bottom of the can), and one long rectangle (the body of the can), as shown in Figure 2.7. The disassembly and layout process

described is similar to the disassembly and layout method that an artist would use to create a UV map for a 3d can of soda.

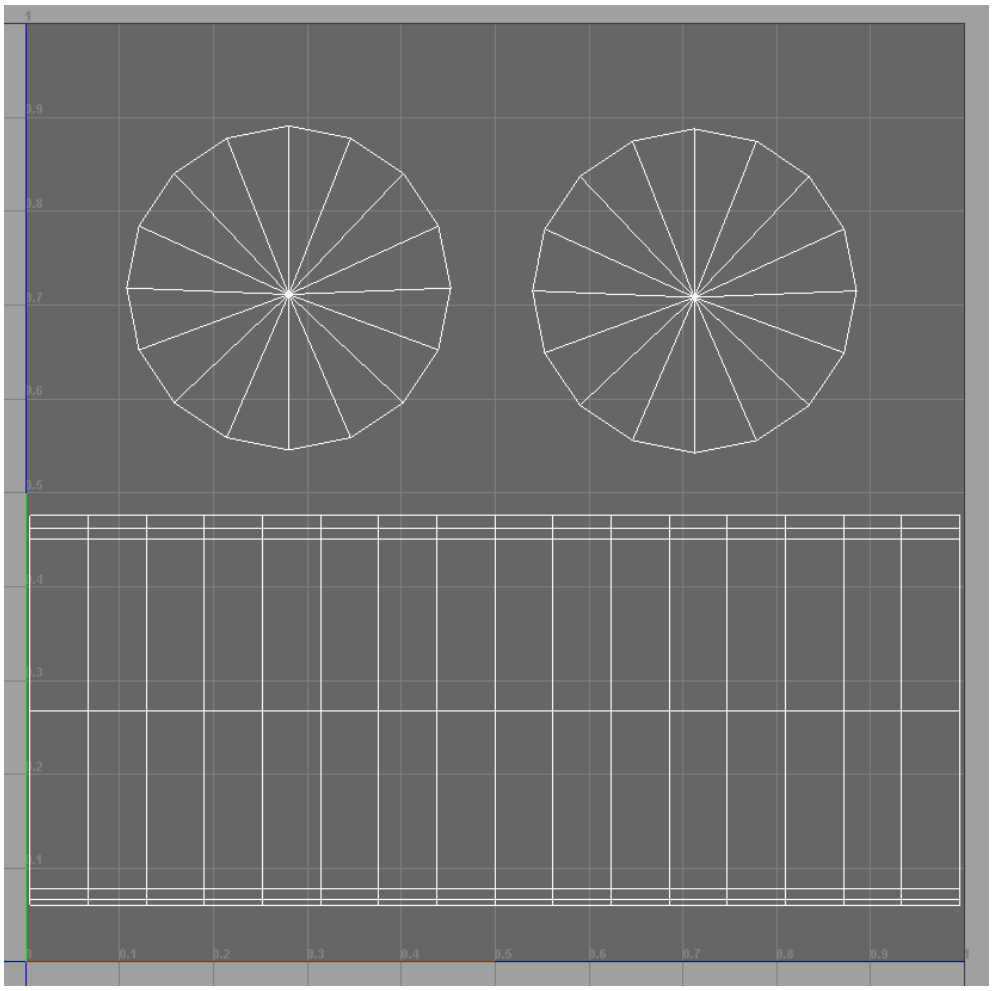

Figure 2.7: UV's for a can of soda.

For the purposes of *Spider Fight*, geometry is composed of polygonal faces, the corners of which contain UV coordinates. The outer lines of a polygon constitute the face's edges. To be a valid surface, a polygon must have at least as many edges as vertices. In general, when an edge of one polygon matches the edge of a neighboring polygon, the corresponding vertices and edges are welded together to produce a single edge. Although the two polygons share vertices at each end of the edge as well as the edge itself, each will retain individual UV coordinates for

the vertices. Each polygon will therefore contain the same number of UV's, regardless of any type of weld operation with any number of other polygons.

In order to lay out a UV map, the user must decide how the geometry will be split. To cut geometry into pieces for a UV map, the artist must instruct the software where to cut the geometry by selecting the edges, or the polygons, that will serve as the seam for the cut. The method for making the cut depends largely on the software used. Maya, for example, requires an artist to select the type of projection to use to flatten the geometry (planar, cylindrical, or spherical, in addition to more advanced options). Maya will unwrap any part of the geometry that is selected, or if no part is selected, Maya will unwrap the entire piece of geometry. Using a different program, UVLayout, requires the artist to select edges to act as the seams, which are used to separate a piece of geometry from the whole. Once the pieces are separated, UVLayout uses an interactive pelt-mapping method to flatten them. The planar unwrap and pelt map methods are shown in Figure 2.8.

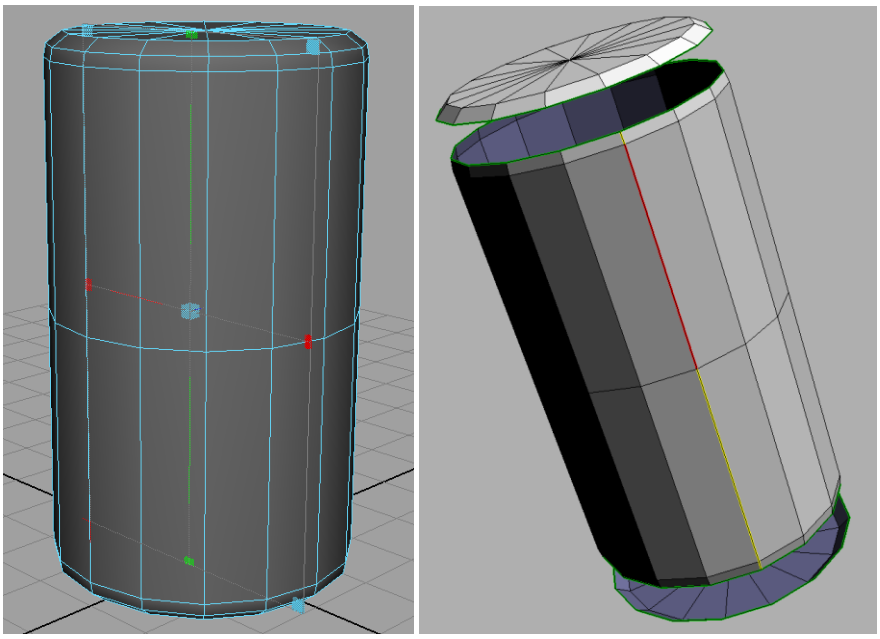

Figure 2.8: Planar unwrap in Maya (left); pelt map in UVLayout (right).

2.4 Texture Creation

A texture map is a collection of colors and texture images. The process of creating a texture map usually involves collecting images or textures from photographs, and pairing those images with textures created inside of a program capable of manipulating a texture map. A texture map can be created in a number of different programs, some of which are specific for image editing, and some of which are not. For *Spider Fight*, two programs were used for creating texturing maps: ZBrush and Photoshop. These two programs vary greatly in their approaches to texture map creation. ZBrush allows the user to paint directly on a 3d model, and is also capable of using an existing UV map. As such, when a texture is painted on an object and the texture map is saved, the texture map will be created utilizing the 3d object's UV map.

A texture created in Photoshop, as shown in Figure 2.9, follows a completely different workflow from a texture map created in UVLayout. Photoshop allows the user to paint on an object, but not to manipulate a 3d object with the same degree of control as ZBrush. As a result, texture maps created in Photoshop are made by drawing on the model's UV map. The first step in this process is to create a 2d image from the UV map. In Maya, this task can be accomplished through the UV Editor and the "Render UV's" command. The 2d map is then opened in Photoshop (or any other image editing program) and effectively used as a drawing canvas where the UV's act as a guide for placing textures.

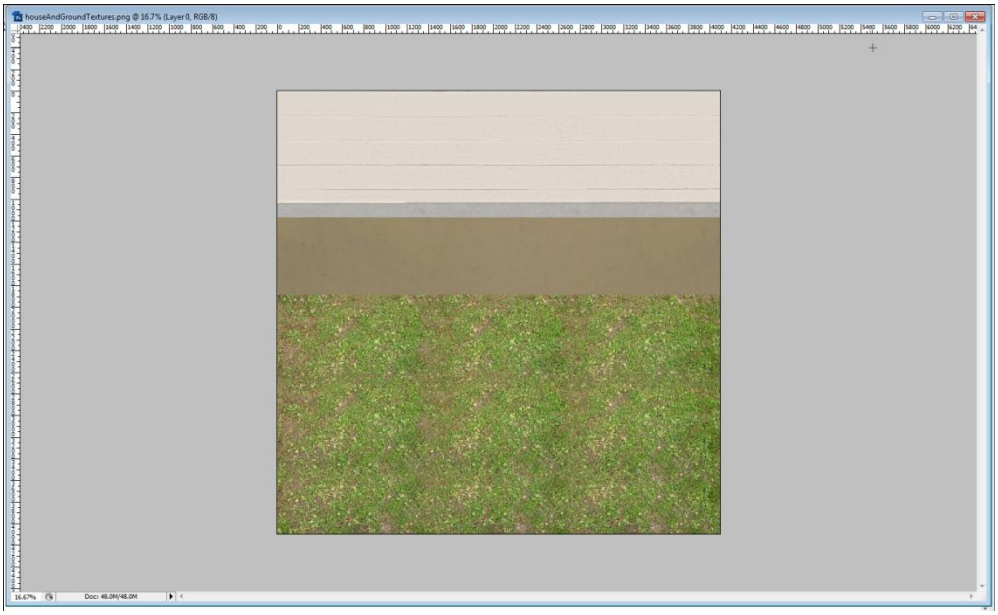

Figure 2.9: Texture map editing in Photoshop.

One disadvantage of using a 2d paint program to create a texture is matching opposite sides of a UV shell at the seam. A good practice is to lay out a model's UV's in such a way that the seams of the UV shells are in places that will most likely not be seen by the camera. A few common locations of UV seams for characters are under the arms, at the back of the head, and on the inside of the leg. If the two opposite sides of a UV shell are solid colors, the seam is easy to hide. If the two opposite sides of a UV shell are a gradient or mixed pattern, however, the seam is difficult to hide in an image editing program such as Photoshop. ZBrush, however, can solve such a problem because the program allows the artist to paint directly on the object; consequently, the artist can paint over the seam and ZBrush will correctly match the texture on opposite sides of the UV shells, creating a seamless texture. Based on advantages of each software package, some workflows use Photoshop for creating the initial texture and ZBrush for fixing seam issues.

2.5 Texture Resolution

Another important aspect of texture mapping is texture resolution. Since UV's are based on a 3d model that has no physical size and hence no true size, the UV's themselves have no true size either. Additionally, a UV map is infinitely scalable. When the "Render UV's" command is used in Maya, the artist must specify the resolution of the 2d UV image output. Common texture sizes are 512 x 512, 1024 x 1024, and 2048 x 2048 pixels. Since 3d software packages typically set the range of both U and V as [(0,0), (1,1)], UV space is a perfect unit square. As a result, UV textures are most often square as well.

The most important element in determining the resolution of a texture for a short film is the distance from the texture to the camera. The texture map must be sized to avoid pixelation when the texture is at its closest point to the camera. For example, if the resolution of a film is 720 x 480 pixels, and a certain 3d object nears the camera and fills the entire frame, the texture resolution of that area must be at least 720 x 480, thereby affecting the resolution of the entire texture map for the object. A texture map larger than 720 x 480 may not be adequate since the UV shells that correspond to the surface that fills the frame must be at least the resolution of the camera.

After a texture map has been created and saved in a compatible image format, the texture map is ready for use in Maya. Maya can read a wide variety of image formats, but for *Spider Fight*, only PNG and TIF were used. Texture maps created in ZBrush were saved as PSD files, opened in Photoshop, and saved in PNG format. The next step was to apply the texture to a shader in Maya, since each object in Maya must have an attached shader in order to use a texture map. Though many shader types exist, e.g. Blinn, Lambert, Phong, etc., all shaders use texture maps in the same way. The texture map is mapped to the desired channel of the shader

via a file node. Once the texture has been mapped, the texture will be applied to any 3d objects that use that shader. The texture map will then be applied to objects according to the UV map for each individual object. In the next chapter, we will see how such textures are used.

Chapter 3: Implementation

With a thorough understanding of rigging systems available in Maya as well as competence in texture creation, an application of these principles began. Rather than use accepted conventions for manipulating the eyelids, new techniques were developed and applied. In addition, separate pipelines were established for the creation of textures, dependent on the type of model that was being textured. A discussion of these concepts follows.

3.1 Eyelid Rigging

This paper tests the notion that inefficiencies exist with standard eyelid rigs. As discussed in the previous chapter, eyelid setups typically use blend shapes or joints at the eye's center point. Our method, however, allowed the eyelid geometry to move along the surface of the eye geometry, where the eyelids are either rotated based on joints located at the eye's center point, or the eyelids are sculpted in positions along the eye using blend shapes. This method involved creating several curves along the surface of the eyelid. The number of curves depended on the complexity of the eye. Since this method was experimental, testing was performed to determine the number of curves needed; ultimately, four curves per eye were sufficient, based on the eyelid geometry itself. Wherever an edge loop from the eyelid geometry touched the top or bottom of the eye, a curve was needed. In this approach, four edge loops touched the top of the eye, and four edge loops touched the bottom of the eye. The result appeared to indicate that eight eye curves were needed; however, only four were necessary because the curves could be shared. Four eyelid curves were created and each curve

was modeled to fit the shape of the eye. After modeling, these curves were moved slightly away from the eye to provide separation.

Once the eye curves were positioned in a satisfactory manner, locators were attached to each curve. The locators were used as drivers, but the locators had to follow, or be constrained to, the shape of the curve to accomplish that task. In Maya, a number of constraints are available to the artist to establish a control relationship between two objects. In the eyelid setup, the most effective constraint was a geometry constraint. When a locator was constrained to an eye curve using a geometry constraint, the locator would follow the shape of the curve exclusively. Since the shape of the eye curve matched the eye geometry, the locator would effectively traverse the surface of the eye geometry. After the four eye curves had been created, four locators were created, with each attached to its respective curve using a geometry constraint, as shown in Figure 3.1.

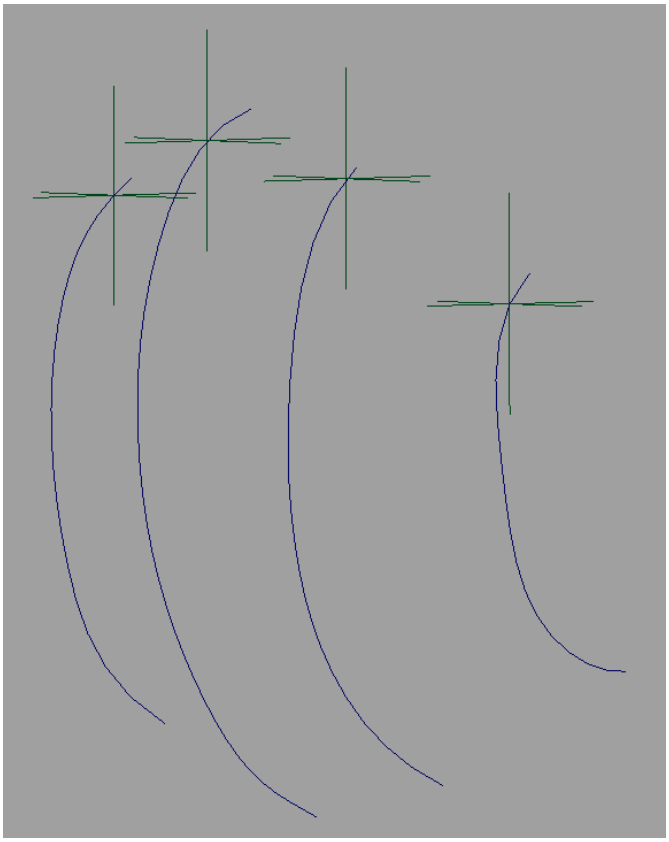

Figure 3.1: Eye curve with locators.

As the locator moved closer to the center of the eye, the geometry of the eyelid was pulled through the eye geometry. To correct these penetrations, one locator was used for every three edge loops, across which weights could be adjusted with bound joints. Three locators were constrained to each eye curve. These curves were shared by the top and bottom eye locators, which resulted in six locators constrained to each eye curve. At this point, four eye curves had been created, each with six constrained locators, as shown in Figure 3.2, but no joints had been attached. The locators could not yet affect the eyelid geometry.

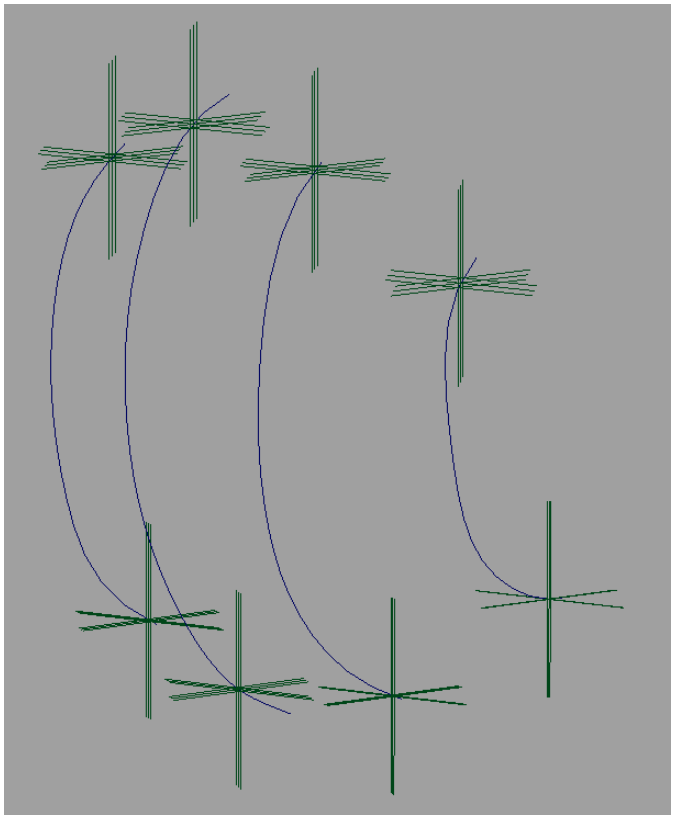

Figure 3.2: Eye curve with six locators per curve.

To resolve this issue, one joint was created for each locator, each of which was point-constrained to a locator, as shown in Figure 3.3. Following the constraint operation, as a locator was moved along a given eye curve, a joint followed it. The joints could have been constrained to the curves directly; however, the joints were instead constrained to locators to facilitate manipulating the eyelid rig. The locators were positioned on the curve, based on the location of the three edge loops that the joints were to control. The joints were bound to the eyelid geometry and weighted in a manner that provided a given joint 100% control of the nearest eyelid surface point, 75% control of the second closest eyelid surface point to it, and 50% control of the next closest eyelid surface point.

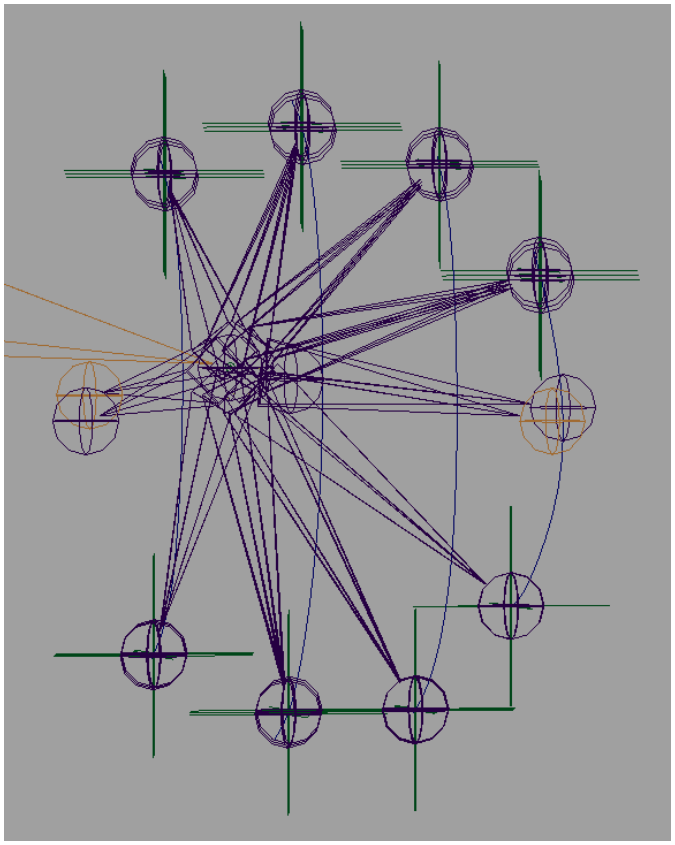

Figure 3.3: Eyelid curves with locators and joints.

3.2 Controlling Locators

The next step in the eyelid rigging pipeline involved controlling the locators in a weighted manner. If all the locators on the eye curves were moved down at the same rate, the geometry eyelid would become folded or bunched. To prevent this issue, locator positions were adjusted to facilitate animation. The first set considered included the eye curve, locators, and joints that were positioned at the top half of the middle eye curve. Locators controlling the eyelid geometry were named to reflect the part of the eye they controlled. A joint was constrained to each of the three locators, where one locator (1B) controlled the bottom part of the eyelid, one (2M) controlled the lower middle part of the eyelid, and one (3U) controlled the upper middle part of the eyelid. The six locators of each eye curve are shown in Figure 3.4.

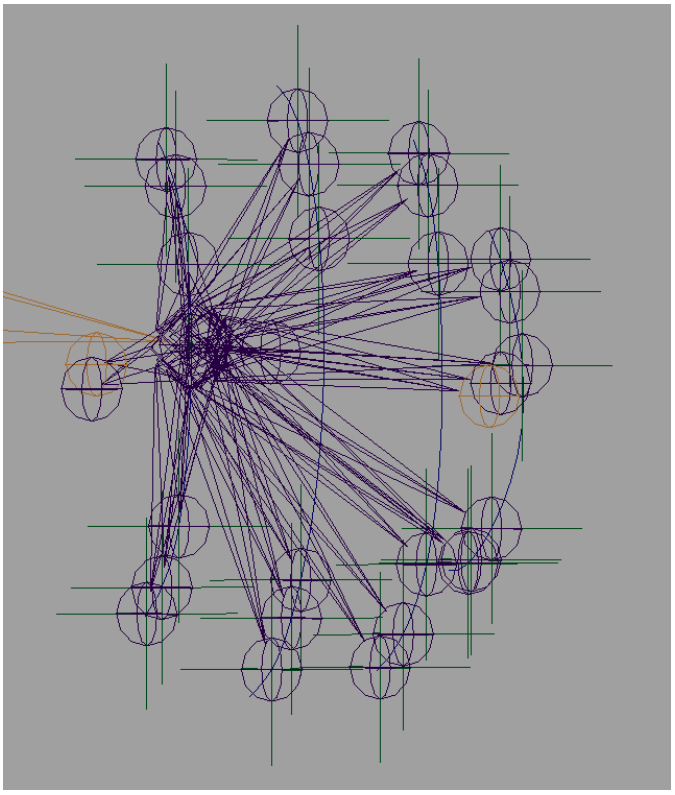

Figure 3.4: Locators driving the joints along the eyelid curves.

To adjust the movement of the joints, 1B was moved to the middle of the eye, which moved the bottom part of the eyelid geometry to the middle of the eye. As noted, the geometry for the bottom part of the eyelid appeared correct, but since other parts of it remained in place, the eyelid geometry cut through the eye geometry. Moving 2M to a position about 80% closer to 1B and 3U to a position 50% closer to 2M eliminated this problem. Finally, the top-most and bottom-most vertices of the eyelid shared weighting with the joint in the center of the eye, such that the top-most and bottom-most vertices of the eyelids would not fully move. Figure 3.5 demonstrates the result of weighting the joints.

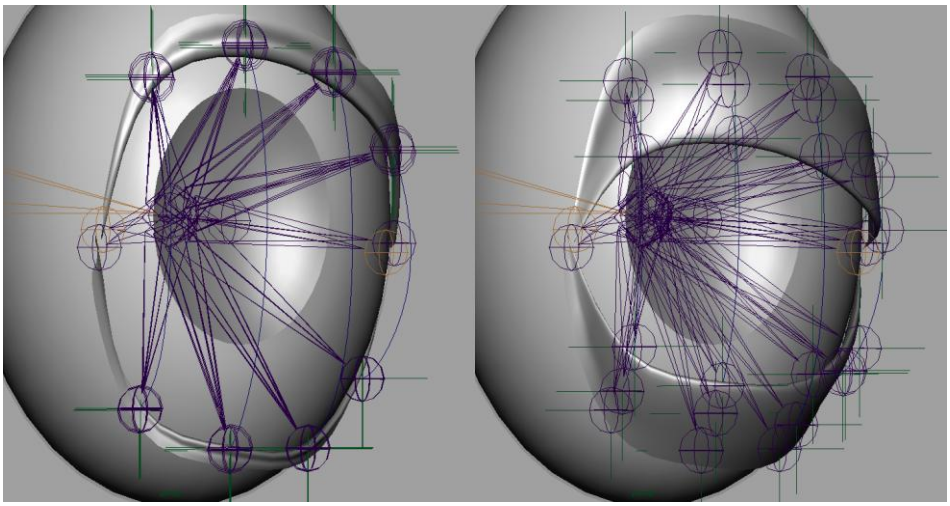

Figure 3.5: Eyelid geometry driven by weighted joints.

Once the locators were moved to visually appealing and collision-free positions, set driven keys were created. Keying all 24 locators when the character blinked would have been impractical; hence, a system of set driven keys was paired with a GUI which controlled the eyelid movements. First, the translation channels of all locators were keyed while the eyelids were in a default state. Second, these channels were keyed while the eyelids were in a closed state. Finally, the translation channels were keyed while the eyelids were in a widened state. At this point, the set driven keys were created and attached to the GUI that allowed the animator to move a curve-based visualization of the eyelids up and down to control the movement of the eyelids, as shown in Figure 3.6. Additional set driven keys were created for angry and startled expressions.

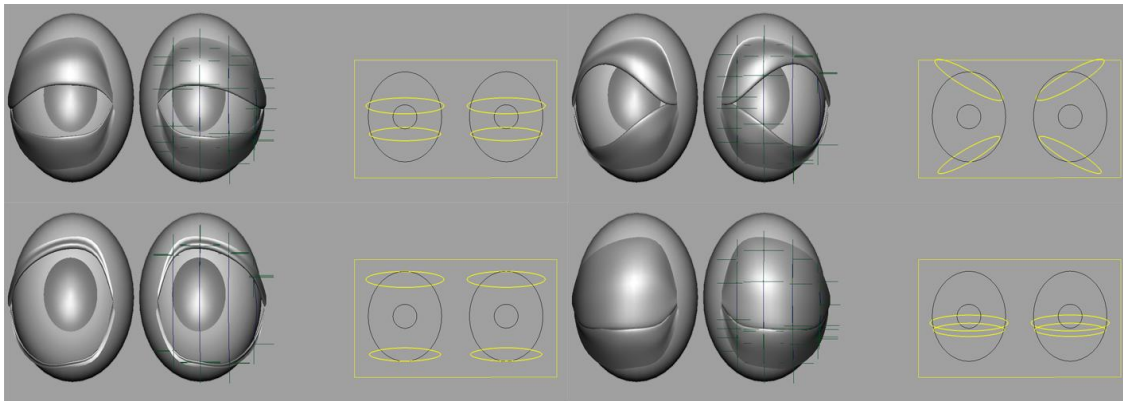

Figure 3.6: Eyelids and locators controlled through a GUI.

The eyelid rig was applied to both of the main characters in *Spider Fight*. After several simple test scenes were animated, the rig was production-ready, and no further methods were required to control or shape the movement of the eyelids. Some advantages and disadvantages of this system are presented in the next section.

3.3 Discussion of Curve-Based Rigging Systems

One advantage of the curve-based rigging system is that large portions of the rigging system can be automated through scripting. Once the curves are in place, MEL (or Python, or C++) can be used to create the desired number of curves per eye, which can then be geometrically constrained to the eye curves, as shown in Figure 3.7. Further scripting allows selected points of the eyelid geometry to be controlled by a cluster, which is then attached to a selected locator constrained to an eye curve. This ability to script portions of the eyelid setup is an advantage over the use of blend shapes, which appear to lack such capability. The only 'sculpting' time required by the curve-based eye rigging system is in matching the eye curve to the shape of the eye.

```
string $theCurves[] = `ls -sl`;
string $theCurve = $theCurves[0];
CreateLocator;
string $theLocator[] = `ls -sl`;
setAttr ($theLocator[0] + ".sx") .15;
setAttr ($theLocator[0] + ".sy") .15;
setAttr ($theLocator[0] + ".sz") .15;
rename ($theLocator) ($theCurve + "L01");

duplicate -rr;
duplicate -rr;

int $i = 1;
for ($i = 1; $i < 4; $i++)
{ select -r $theCurve ;
  select -tgl ($theCurve + "L0" + $i);
  geometryConstraint -weight 1;
  string $locatorName = ($theCurve + "L0" + $i);
  float $locatorPosition = `getAttr ($locatorName + ".ty")`;
  $locatorPosition += float($i/8.0);
  setAttr ($locatorName + ".ty") ($locatorPosition);
}
```

Figure 3.7: MEL code for creating, naming, and constraining locators to eyelid curves.

Another advantage of the curve-based eye rigging system is that adjustments are non-destructive. If the eyelid penetrates the eye, the issue can be resolved by grabbing the necessary vertex or vertices of the eye curve and adjusting them such that the eyelid geometry no longer intersects the eye geometry. The locators that are driving the eyelid geometry follow the shape of the eye curve, hence the locators will now follow the adjusted path, as shown in Figure 3.8. In contrast, if a problem is discovered with a blend shape, an artist must re-sculpt the blend shape in a way that prevents the eyelid from intersecting the eye geometry.

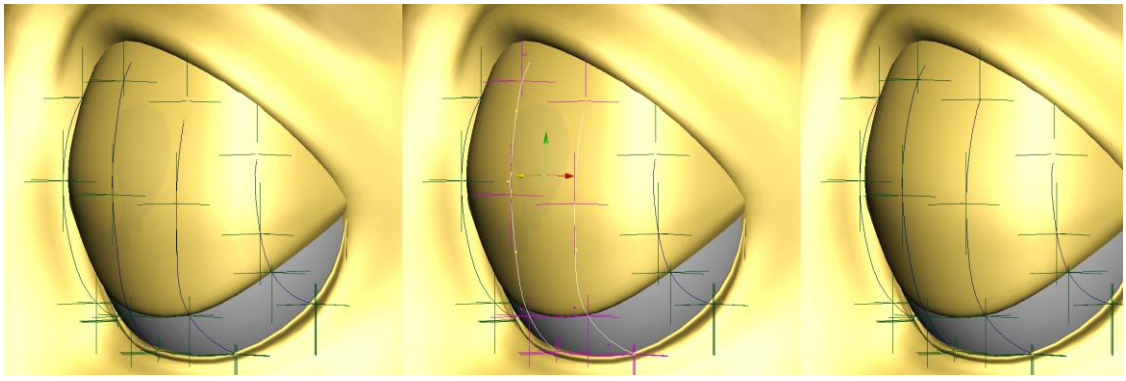

Figure 3.8: Geometry collision fixed by adjusting the eye curves.

A final advantage of the curve-based eye rigging setup is that the eye curves can fit any eye shape. The curve-based eye rigging setup can handle ovular, wavy, or heavily angled shapes. The curve-based eye rigging setup would function as expected if the eye curve had a sufficient number of points to mimic the eye shape.

Drawbacks to the curve-based eye rigging setup should be noted. One disadvantage is that the curve-based eye rigging setup is not as precise as blend shapes. Imperfections or wrinkles can be sculpted into a blend shape without much difficulty, but for a curve-based system, adding wrinkles would require special curves, with accompanying locators and joints. The curve-based eye rigging setup, however, can be paired with blend shapes to achieve additional details such as imperfections and wrinkles.

A second drawback of the curve-based eye rigging setup is that all locators must be keyed and controlled through a set driven key setup. A script can be written to assist in the process, but an artist is needed to key the locators and move them to correct positions. After the work of keying the locators is completed, the system can be controlled through a GUI.

3.4 Prop Texture Pipeline

Another major issue in the creation of *Spider Fight* was prop textures. The short film featured two main characters, two secondary characters, and an outdoor environment that included a house, bug zapper, and natural environment. In order to effectively complete texturing work on all of the objects that were in the film, two texturing pipelines were developed. One pipeline was set up to handle the four animated characters, and another pipeline was created to address the texturing needs of all the other 3d objects.

A prop pipeline was developed in which all the props were first given a solid color shader. After a render and evaluation was made of the scene, a texture was created matching

the original solid color, as shown in Figure 3.9. The first object textured was the side of the house with a solid cream color.

Figure 3.9: Testing the color palette by rendering with flat colors and simple textures.

The next step in the prop-texturing pipeline was to gather textures for each individual object. After initial approval, textures were sought for the side of the house. The primary source for these textures was a website that specializes in textures [VIJF11]. The site features a search function for finding textures, which are royalty-free, high-resolution, and in some cases tiled. Each prop used approximately two to three different textures.

After gathering textures, a UV map was created for each prop in Maya. Planar projection and atlas map were the two most commonly used methods for creating the UV maps. Planar projection was used for the ground plane, because the ground plane was created from a polygon plane, and the ground plane was only slightly bumpy. In addition, using the planar mapping solution kept all of the faces of the ground plane together. Planar projection was also used for the side of the house, which largely resembled an upright polygon plane.

Both planar projection and atlas map algorithms were used on the bug zapper geometry. The four side faces of the bug zapper were each mapped using a projection that was

oriented to the normal direction of the side. The top and bottom of the bug zapper were mapped using separate planar projections. The remaining pieces, which consisted of the individual bars on the bug zapper with normals not facing outward, were mapped using an atlas map. This method sped up the mapping process on these pieces since it required no artist intervention. Once all of the parts of the bug zapper were mapped, the various UV shells were fit onto one UV map, with careful consideration given to maintaining consistently sized UV shells. UV maps were not created for the trees or the grass, as both objects were produced with Maya paint effects and thus already possessed sufficient mapping. The texture map and repeating values on the trees, however, were adjusted for consistency.

After creating UV maps for all of the props, an appropriate texture resolution was determined. As a base, no texture map was smaller than 1024 x 1024. Once the closest position of each object to the camera was found, the necessary texture size for each given prop was calculated. Two props received special attention: the bug zapper and the side of the house. Since several shots in the film included very close shots of each, their texture was set at 4096 x 4096 pixels to provide high-quality results.

Due to the proximity of the camera to the side of the house in several shots, a stand-in object was required to avoid a single impractically large texture. The house wall was approximately 166 times as wide as either of the spider characters. Since some shots depicted one of the spider characters close to the wall, a texture size of at least 2048 x 2048 pixels was required for a very small section of the side of the house, perhaps an area only three percent of the total size. Creating a texture sufficient for this situation would result in a file size impractical for interactive use and rendering.

Although a repeating texture could have been used on the wall, it would have been noticeable in many of the distant shots of the side of the house, and could potentially disrupt the viewing experience. The solution was to model one small section of the house wall. To make this piece as versatile as possible, it was textured with a siding texture as well as a foundation texture. In addition, the shape of this small wall piece allowed several to be stacked in such a way as to cover the foundation texture of each piece, and allow a wall of any size to be 'built' for a shot. To guard against pixelation, a texture size of 4096 x 4096 was employed. This approach kept the wall textures from becoming pixelated in close shots, while eliminating the need for a single massive texture for the side of the house.

3.5 Character Texture Pipeline

Once all the texturing for the props was completed, texture work for the characters began. The foundation for the character pipeline was built upon work from past projects. The spider characters were organic and hence required a different UV workflow. Although planar maps were an option, molding such maps in a way to minimize stretching and overlapping of UV's is difficult. Another issue was that painting the textures in Photoshop would result in texture seams that could not be hidden, as the spider characters were shot from various angles throughout the film; therefore, the common UV trick of hiding the seams of the UV shells on the back of the head, or on the inside of the legs and arms, would not be sufficient.

In order to achieve an organic UV layout for the characters, a program called UVLayout was used. One feature of this software is that it allows an artist to select the location of the seams, and then visually separates the 3d model along these seams for viewing. UVLayout also utilizes a pelt mapping algorithm, which is suited for UV mapping of organic shapes. Pelt mapping works in a similar fashion to the way clothes are made. If a shirt or a pair of paints

were cut along the seams, and the pieces laid flat, the resulting shapes would be similar to those produced by pelt mapping.

The first character from *Spider Fight* to undergo pelt mapping was Larry, the yellow spider. Seams were created around Larry's face, down the back of his head, under his body, and on the inside of all of his legs. The parts were then visually separated in UVLayout, before being placed on the UV map and flattened. An advantage of UVLayout is that the program color codes polygons on the UV map, which indicates whether stretching is occurring. In addition, when UVLayout places the various UV shells onto the UV map, the program displays all of the pieces in proper proportion to each other, as shown in Figure 3.10. As a result, once all of the pieces of the geometry are flattened on the UV map, the pieces are all of relative size; hence, a texture using the map will not be stretched. After all of the UV shells for Larry were flattened, the model was exported, and moved into the texturing phase of the pipeline.

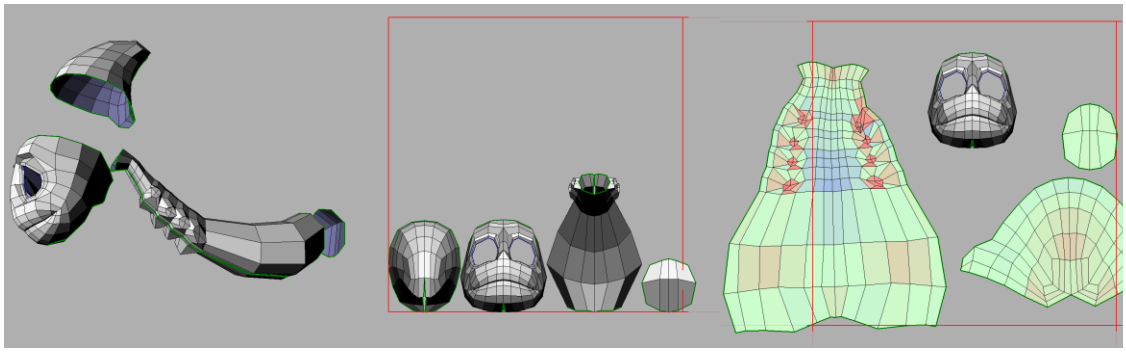

Figure 3.10: Unwrapping and flattening in UVLayout.

Rather than texturing the models with Photoshop, both spider characters were textured in ZBrush. ZBrush was used in order to seamlessly lay out the textures of the spiders across their respective UV shells because the textures used several gradients that cross UV seams. Although Photoshop could have been used to create the textures, Photoshop does not have an accurate method for maintaining continuity across UV shells. As a result, the gradients in the

textures would not have been aligned at the seams and would have produced a line discontinuity at that edge. In ZBrush, an artist can paint directly onto the 3d model, from which the software creates a texture map in accordance with the UV map of the geometry. As a result, painting the model in ZBrush eliminates the textures seams on the geometry.

Figure 3.11: 'Larry' character textured in ZBrush.

Another advantage of a texture map created in ZBrush is that the resulting texture map is resolution-independent. An artist begins by painting a texture onto the model without importing a texture map. Once the object is painted, the artist then creates a texture map within ZBrush. The artist specifies a texture resolution, and ZBrush converts the painted texture to a texture map of that resolution. In addition, the artist can save several texture maps at various resolutions and test each map by rendering scenes in which the texture is used. If the texture map appears pixelated, the artist can return to ZBrush, and save the texture map at a higher resolution without the need for more painting operations. Once a satisfactory texture

map was created for each of the spider characters, the texture maps were edited in Photoshop to correct any small painting mistakes, or fill in each area designated a solid color.

The only parts of the characters in *Spider Fight* that did not utilize a painted texture map were the eyes. The eyes for all of the characters were fractal-based textures. Since a viewer tends to focus on the face, and particularly the eyes when watching a character, the eyes must be textured in a high-quality fashion that resembled human eyes. Fractal-based ramp shaders served as the base, since ramp nodes allow for precise control of color values placed on a surface. White was ramped to the majority of the shader, followed by a thin black line, then an area of color that represented the iris, and finally a solid black area that represented the pupil. The resulting shader resembled an eye, but required fractal values to control the iris color, as shown in Figure 3.12. After the fractal values were mapped to the ramp, the fractals were re-colored according to the desired iris coloration. The completed shader was imported into the reference files of all the characters and edited for coloration.

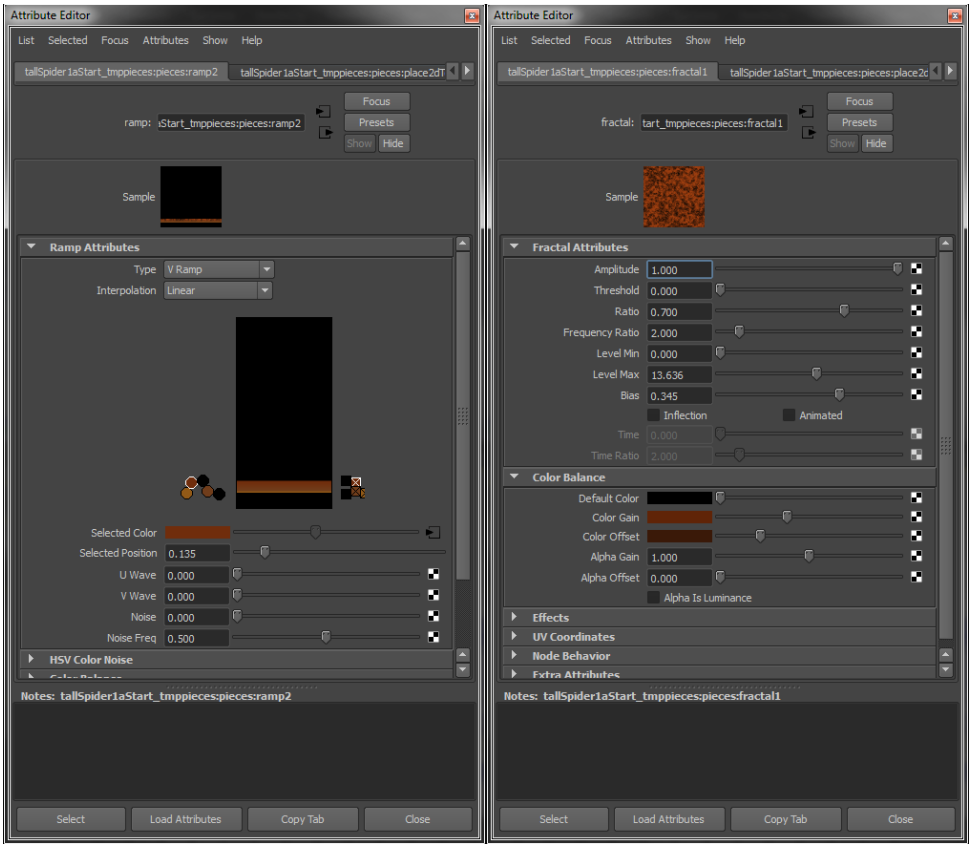

Figure 3.12: Eye ramp shader editing.

3.6 Discussion of Texturing Pipelines

A major advantage of using two pipelines was that both were open to a great deal of flexibility. If needed, ZBrush could have been used in the pipeline for props since it was already used in the pipeline for characters. In addition, UVLayout could have also been used for UV map creation for the props, since it was already being used in the character pipeline. The two individual pipelines produced the results desired; therefore, altering the pipelines on an asset-by-asset basis was not necessary.

Though the texturing pipelines worked well in the production of *Spider Fight*, some drawbacks were noted. The first was that each of the respective pipelines relied on several programs in order to achieve the desired result. Unfortunately, a satisfactory 'one program'

solution for creating UV maps and texture maps was not found. ZBrush was capable of creating UV maps and texturing them, but using UVLayout for UV map creation seemed to provide a more efficient method. Additionally, a newer version of Photoshop has improved its ability to paint directly onto 3d models since the production, which could possibly result in better seam correction for texture maps. Other programs, such as 3d-Coat, also provide techniques for creating UV maps and texturing that were not evaluated. Given more time, this project could have benefited from the opportunity to test new elements for possible inclusion into one or both of the texturing pipelines.

Chapter 4: Results

After the completion of the characters and textures, the necessary rendering of the short film commenced. *Spider Fight* contained almost 70 shots, many of which employed the eye rigging and texturing pipelines for rendering. As shown in Figures 4.1 and 4.2, the closing of the eyelids for each of the main characters showed no penetration issues.

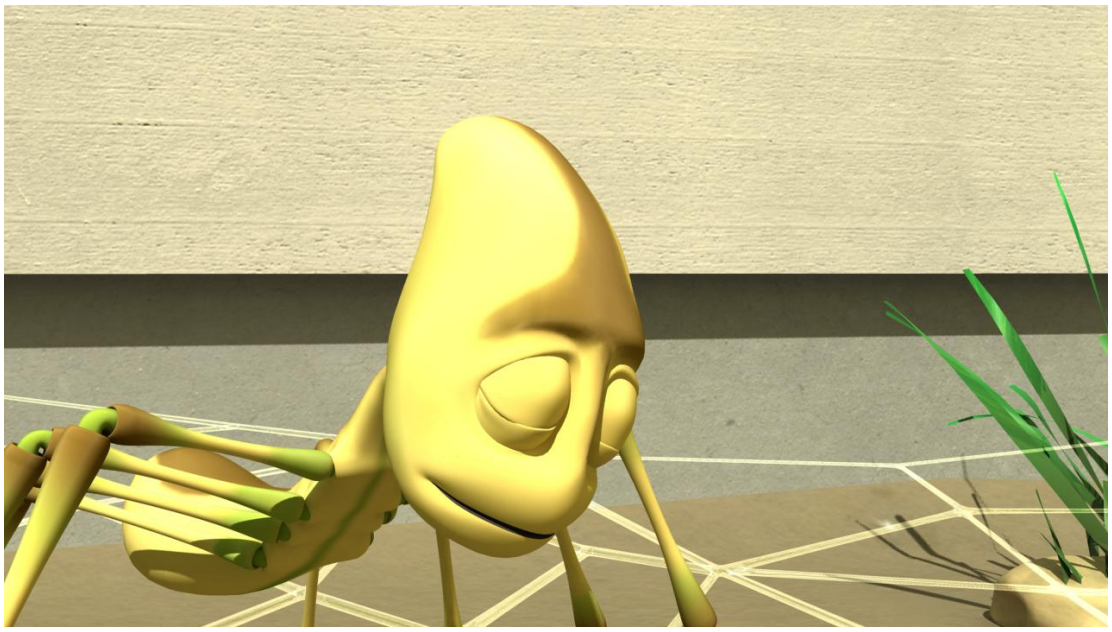

Figure 4.1: Close-up of 'Larry' with closed eyes, showing no penetration of eyelid geometry.

Figure 4.2: Close-up of ‘Merry’ with closed eyes, showing no penetration of eyelid geometry.

With the eye geometry showing no penetration issues when closed, the next part of the curve-based rigging system to evaluate was the appearance of the eyelids when moved into other positions. Scenes that used ‘emotional’ eyelid positions, such as ‘angry’ or ‘startled,’ were targeted for evaluation. As show in Figures 4.3, 4.4, and 4.5, the eyelid geometry performed without error in these cases.

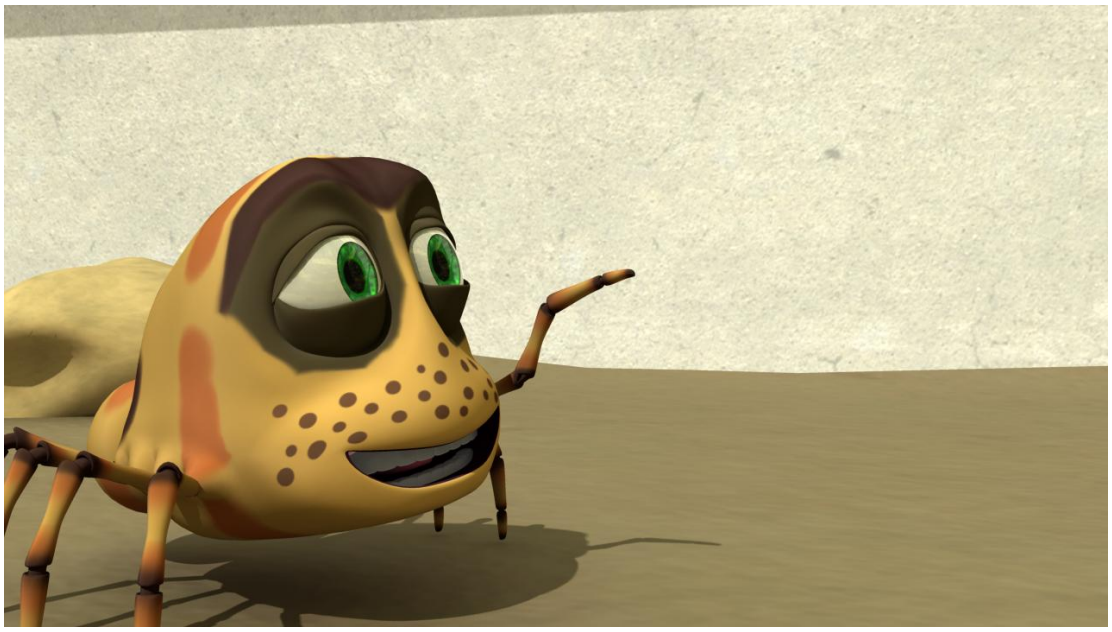

Figure 4.3: Close-up of ‘Merry’ with no eyelid geometry issues.

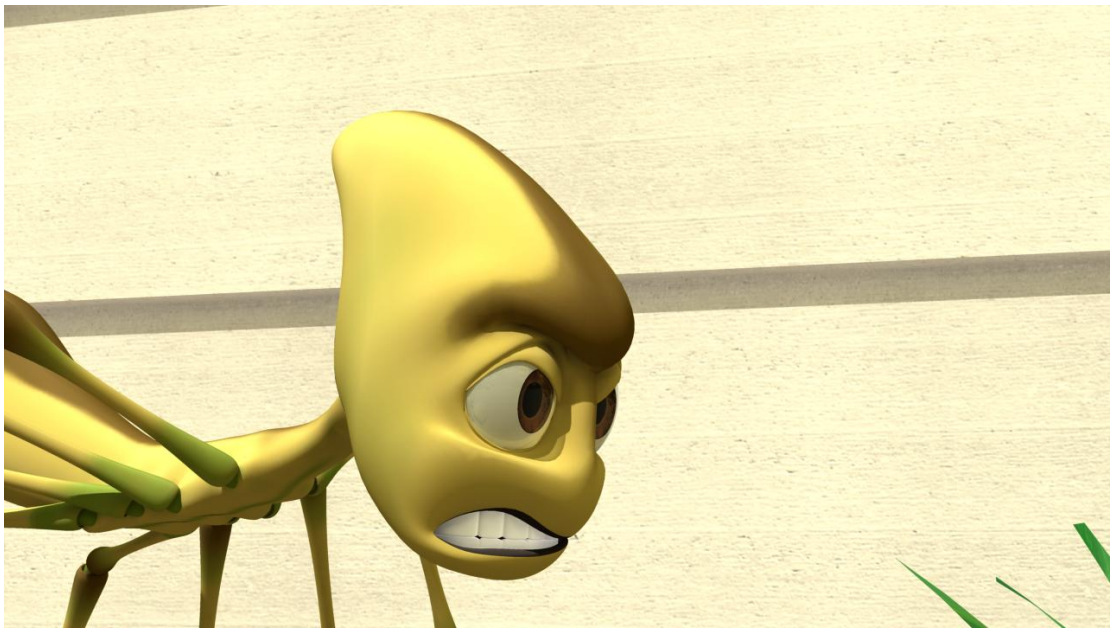

Figure 4.4: ‘Larry’ character with ‘angry’ eyelids.

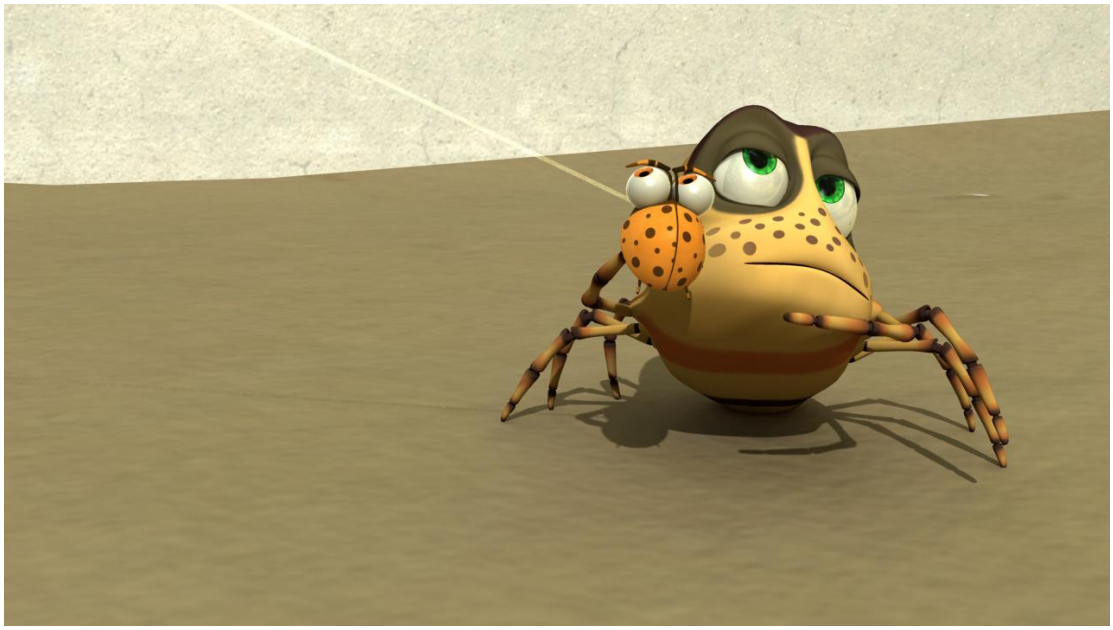

Figure 4.5: ‘Merry’ with upper eyelids down and no geometry issues.

The satisfactory results of the curve-based eyelid rigging system allowed for the textures to undergo evaluation. The first textures to be examined were the eyes. Various scenes were evaluated, including far, near, and close-up shots. Figures 4.6 and 4.7 show near and close-up shots, respectively, with the final version of the eye texture.

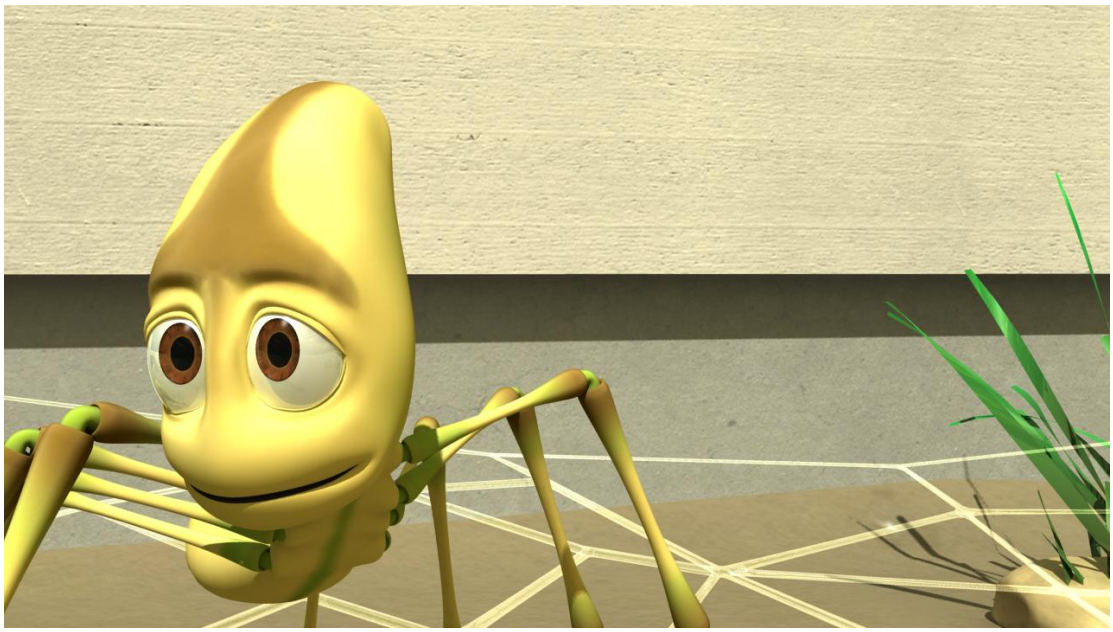

Figure 4.6: Near shot of ‘Larry’ showing ramp-based eye shader.

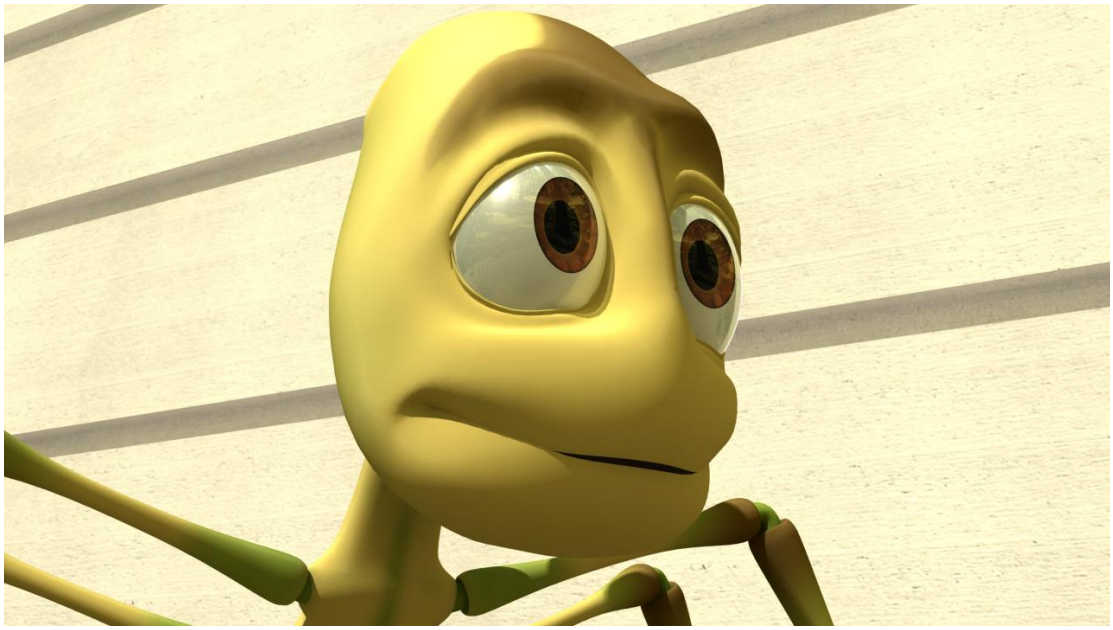

Figure 4.7: Close-up of 'Larry' showing ramp-based eye shader.

The final examination of the renders involved the texture maps. The unpixelated texture of the stand-in wall geometry can be seen in the background of Figures 4.8 and 4.9. Specifically, Figure 4.8 shows the base of the stand-in wall texture whereas Figure 4.9 demonstrates the siding texture.

Figure 4.8: 'Merry' and bug character, with wall piece in background.

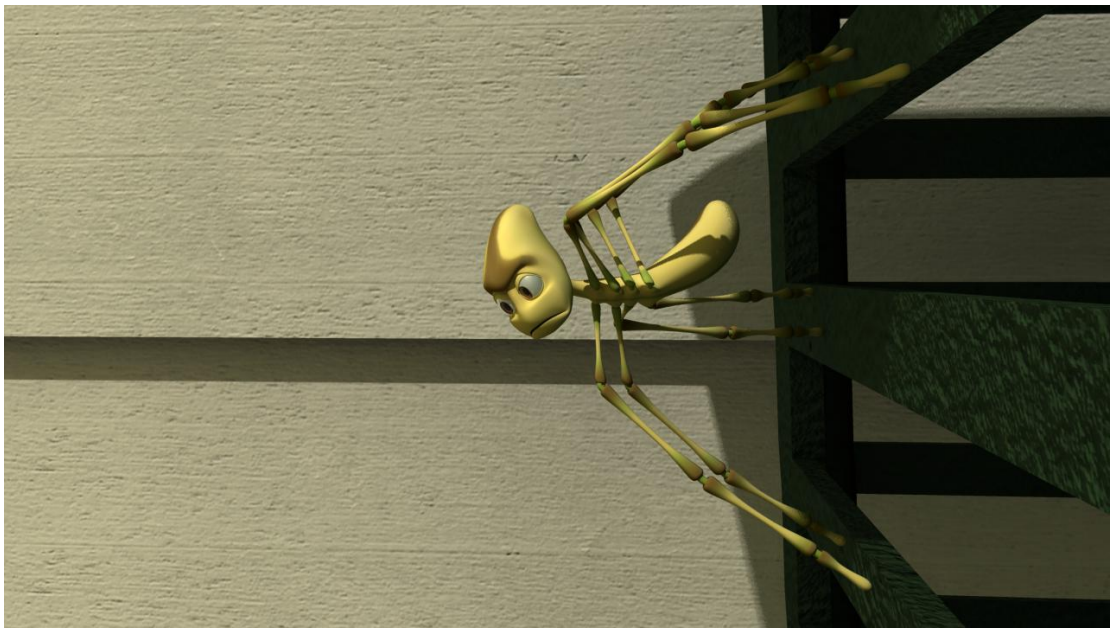

Figure 4.9: 'Larry' with wall piece and bug zapper.

Shots with the spider characters occupying a large amount of screen space were a concern during the texturing phase; however, because consideration was given to screen space early in the texturing pipeline, both spider characters received a sufficiently large texture map. As a result, neither character displayed pixelation or texture blurring in close-ups, as shown in Figures 4.10 and 4.11.

Figure 4.10: ‘Merry’ shown close to camera.

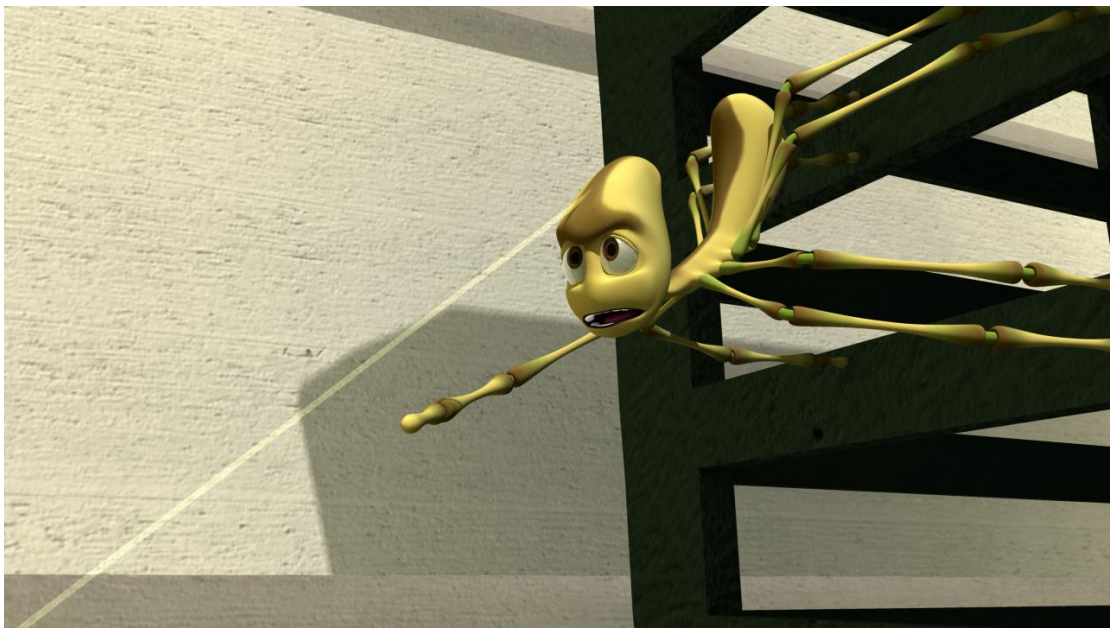

Figure 4.11: ‘Larry’ shown close to camera.

Chapter 5: Conclusion and Future Work

The curve-based eye rigging setup that was created and implemented in *Spider Fight* provided solid results, and proved to be capable of handling the eyelid movements of the two main characters. Though the curve-based eye rigging system was entirely conceptual at the onset of the project, preplanning guaranteed that the system was integrated without delaying the production. In addition, only minor tweaks were required to the first iteration of the curve-based eye rigging setup, which helped facilitate the expressions that were necessary for the short story.

Due to the success of the curve-based eye rigging setup, this approach may hold potential for facial rigging. By this process, a curve-based system could provide similar results as a blend shape, but perhaps with a higher degree of realism. Unlike blend shapes, a curve-based rigging system applied to a facial setup could be tailored in such a way that the curves could function as muscles under the skin. Conceptually, if a model of a skull were created, curves could be created that, instead of following the shape of an eye, could be situated to simulate the possible motion paths of facial muscles. In this way, locators constrained to the curves could drive joints or clusters to represent the muscles moving under the skin.

An advantage of a curve-based system is that curves can be set up to move along and around the skull, mimicking real muscles. Blend shapes are not well suited for mimicking such movements around shapes, and thus multiple blend shapes would be required per facial movement. The curve-based system would be capable of using any number of moving locators to mimic the movement of muscles, and would retain the advantage of curve modification to refine the animation. In addition, scripts could be used to assist in the setup.

The curve-based eye rigging setup used in *Spider Fight* could be improved in future implementations. The first change would be to use clusters instead of joints. The issue with using joints is that when the joints are bound to a piece of geometry, they are bound to the entire piece of geometry, which often results in zeroing the weights on points not affected by the joint. Clusters solve the problem because they only affect the points that are selected when the cluster is created. Of course, some weight adjustment would still be required, but only on a limited number of points.

Secondly, further implementation of the curve-based rigging system need not use a constraint to attach the locators to the curve, but use the curve as a motion path. Attaching the locators via a motion path would provide an advantage in the way the locators on the curve are controlled. In the current method, the locators are moved to arbitrary values in order to open and close the eyelids. The destination values of these locators differ from one model to the next, even if the eyes are of the same shape. Using a motion path instead of a geometry constraint, however, would move all the locators in a normalized space, regardless of the size or shape of the eyes. As such, scripting the location of the locators would become simpler. For example, to close the eyelids, three upper eyelid locators can be set to values such .1, .4, and .5, whereas three lower eyelid locators could be set to .5, .8, and 1. Using a curve-based eye rigging setup that utilizes normalized space for the translation of its locators would therefore be uniform and more efficient.

Evaluating the texturing pipelines used in *Spider Fight* reveals several positive production aspects. First, the use of two pipelines allowed for maximum efficiency when texturing objects. Employing one texturing pipeline targeted at props and another targeted at characters helped minimize the time required to texture any particular asset, as no unnecessary

steps were taken in the texturing process; each texturing pipeline used only the programs that were absolutely necessary to texture the given 3d object. Another benefit of the two texturing pipelines was that either of the texturing pipelines could be easily adjusted or extended, if such additions were necessary.

Some areas in the texturing pipelines could be improved upon for future projects. The first issue that should be addressed is the evaluation of additional packages that might enhance the efficiency of one or both texturing pipelines. Several software programs are available that can handle multiple aspects of each of the respective texturing pipelines. As previously mentioned, 3d-Coat has a great deal of potential in a texturing pipeline. In addition, a re-evaluation of the latest version of Photoshop could reveal improvements to Adobe's software that would make Photoshop more valuable in one or both texturing pipelines.

The creation and utilization of the eyelid rig, and the setup and use of two texturing pipelines in the *Spider Fight* short film helped the production of the project flow smoothly, and also allowed more focus to be placed on dealing with other problems that arose in the production. Creating and utilizing a rigging method that controlled the eyelids without the need for blend shapes, and allowed corrections to be made without requiring additional repairs, was invaluable to the production. In addition, the use of two separate streamlined texturing pipelines was valuable in maximizing efficiency of resources and time. Further evaluation of 3d software could lead to even more efficiency in the texturing pipelines. In addition, further exploration of curve-based rigging systems could lead to more realistic animation in the realm of facial animation and beyond.

References:

[ALLE08] Eric Allen and Kelly L. Murdock, *Body Language, Advanced 3D Character Rigging*, Sybex, 2008.

[KOND05] Katsunori Kondo, *Discover the Game with Alias*, Alias, 2005.

[LUHT10] Eric Luhta, *How to Cheat in Maya 2010*, Focal Press, 2010.

[OSIP03] Jason Osipa, *Stop Staring: Facial Modeling and Animation Done Right*, Sybex, 2003.

[VIJF11] Marcel Vijfwinkel, "[CG Textures] – Textures for 3D, graphic design, and Photoshop," http://www.cgtextures.com, January 2011.

[WARD05] Anthony Ward, *Game Character Development with Maya*, New Riders, 2005.