5-2015

Artist-Oriented Surfacing Workflow

Virginia Bailey Nearing
Clemson University

Artist-Oriented Surfacing Workflow

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 Arts

---

by
Virginia Bailey Nearing
May 2015

Accepted by:
Dr. Jerry Tessendorf, Committee Chair
Dr. Robert Geist
Dr. Timothy Davis

Abstract

Surfacing is the art of creating materials for digital objects, and it is an incredible technical and artistic challenge. Creating materials and textures for 3D meshes involves managing potentially hundreds of image files, moving data between multiple software packages, and building shaders that will work consistently through a digital production hierarchy. All while attempting to create a piece of art that will tell a story of what an object is, how it was made, and where it has been. Balancing these technical and artistic components is an extraordinary challenge, and it is easy for an artist to get overwhelmed or distracted by the technical demands of surfacing at the cost of the quality of their art. The topic of this thesis is a suite of tools designed to allow artists to intuitively manage these technical demands by streamlining the most time-consuming aspects of surfacing into condensed one-click operations.

Acknowledgments

I would like to thank my family, Nancy, Steve, and Elizabeth for their unconditional love and support. Thank you to my friends for their constant encouragement and humor, especially Mandy, Ben, Anna, Kristen, and Sarah. And to my undergraduate advisors, Dr. Lindsay Jamieson and Dr. Alan Jamieson, for encouraging me to apply to the Digital Production Arts Program, for their mentorship, and their friendship. Finally, I would like to thank my advisor, Dr. Jerry Tessendorf, and my committee, Dr. Robert Geist and Dr. Timothy Davis for their instruction and guidance throughout my time at Clemson.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The art of texturing digital objects is a fundamental design process in computer graphics. Whether it is for creating a photo-realistic digital object to fit into a live action scene, or creating a fully animated digital film, the artistic choices and quality of the surfacing are critical to the look and impact of the final frame. In order to meet the artistic needs of this technical and complex work, surfacing artists can benefit greatly from a workflow that allows them to focus on their art and design, rather than on repetitive, tedious actions that can be automated.

While an artist's skill is not defined by the quality or complexity of the tools at their disposal - the right set of tools can give the artist the freedom to create more sophisticated work. Having the right tools enables artists to spend their time refining their work and honing in on their artistic vision. Modern productions are massive operations with many artists, many objects, and chaotic deadlines. In this situation the artist's tool kit is every bit as critical as a well prepared canvas. A well developed workflow and tools can give artists the freedom to spend their time on improving their craft, rather than fighting the package or process.

To design and implement a toolset that gives artists this kind of freedom one must understand the medium and the art. Artists in any medium have a workflow, or a set of steps they take to help construct a creative environment. Painters keep their paintbrushes located in a comfortable location so they don't have to think when they reach for them. Canvases are framed, stretched, and primed. Particular paints are selected for their material properties. As digital artists we create and manipulate our environments as well: windows are positioned in specific locations on the screen, we have our favorite digital brushes, our files are stored and named according to personal or production

preferences. Many of these habits will vary from artist to artist in digital and traditional media. Such as what brushes they prefer or how their tools are visually arranged. And some of these things will be the same across a studio - where old paints are disposed of, where the solvent is stored, procedures for ventilation. By understanding these routines as an artist we can build more intuitive tools.

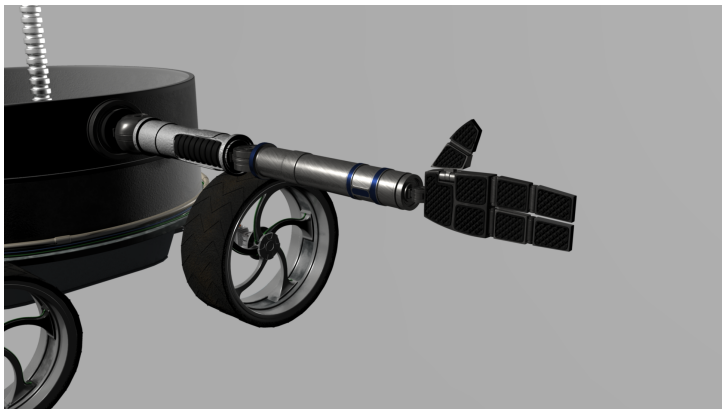
Without a well designed workflow, texture artists will spend the majority of their time manually saving files and building shading networks. When working with complex objects this can become extremely tedious and repetitive, leaving the work prone to human error such as overwriting files, saving to the wrong location, inconsistent naming conventions, and bad file permissions. At the least, this makes it challenging to have multiple artists working on an object, and at the worst can completely break renders.

To manage the technical process of a surfacing workflow across a production one solution is to give the artist a set of tools that distances them from the actual file system. This enables artists more time to focus on their art. By automatically generating names, file paths, permissions, and settings, we can reduce human error significantly and ensure consistency across multiple artists. We can further improve the artists productivity by identifying other repetitive tasks such as shader network generation and transferring objects between graphics packages. The less time an artist spends interacting with the files, file system, and shader networks, the more time they will be able to spend focused on the materials of the object. This enables more iterations of texturing, which in turn means more refined texturing. Figure 1.1 shows how dramatically iterations of textures can improve the quality of surfacing, and figure 1.2 shows the impact that the quality of the surfacing has on a completed frame.

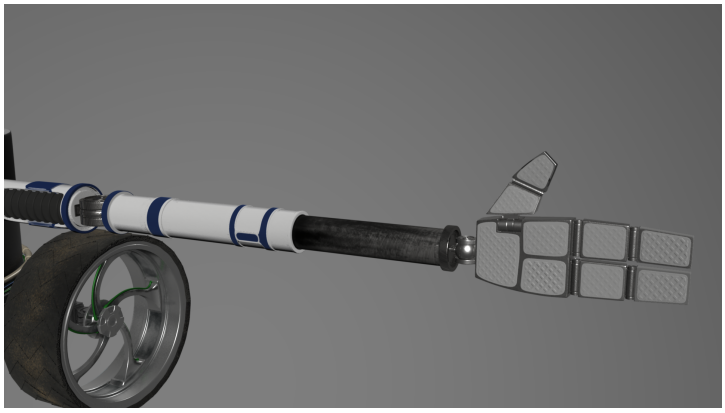
In order for artists to communicate critiques and goals to each other, it is important for us to use a common language. This helps to troubleshoot tools, and to communicate their technical needs.

When surfacing an object, the shader is the center of the artist's attention. A shader defines how the surface of a mesh will look when rendered. It calculates how the light bounces off the object and what color it will be when it reaches the camera. Artists use several attributes to manipulate the shader to achieve their artistic intent.

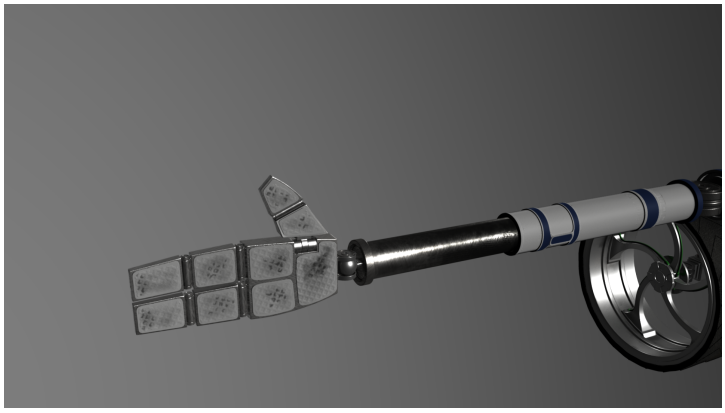
Some common attribute types include diffuse, specular, bump, and displacement. Diffuse attributes relate to the basic color of a material - what it looks like without any reflection, refraction,



(a) Early iteration - metallic arm and black hand-pads



(b) In progress - arm changed to plastic, and pads white for visibility



(c) Final iteration - hand pads and arm worn down and roughed up from use

Figure 1.1: Surfacing Iterations for Alien Oasis

Figure 1.2: Alien Oasis Completed Frame

or shadow. Specular attributes control how the light bounces off of the material to create the shiny glint on an object that helps to visually define it's 3D form for an artist. Bump and displacement perform similar functions in that they alter the surface structure of an object, warping the surface of the geometry without additional modeling. The difference between them is that bump attributes do not actually distort the mesh, they just tweak the calculation of how the light reflects from the surface, giving the appearance of variations in highlights where there would normally be a flat surface. The result is that the silhouette of the geometry is unchanged. Displacement distorts the geometry, affecting the silhouette, and generating shadows.

Each of these attributes are defined by many variables. The simplest is diffuse because the artist only needs to define a color and intensity. Specular properties vary dramatically based off of the equations being used. For example, in a Blinn shader in Maya the artist has control over the specular fall off, eccentricity, and color as shown in figures 1.3. Figure 1.4 shows the variations of the fall off and eccentricity values: the top row shows eccentricity values of 0.1, 0.3, and 0.7, the lower row shows roll off values of 0.3, 0.7, and 0.9[2].

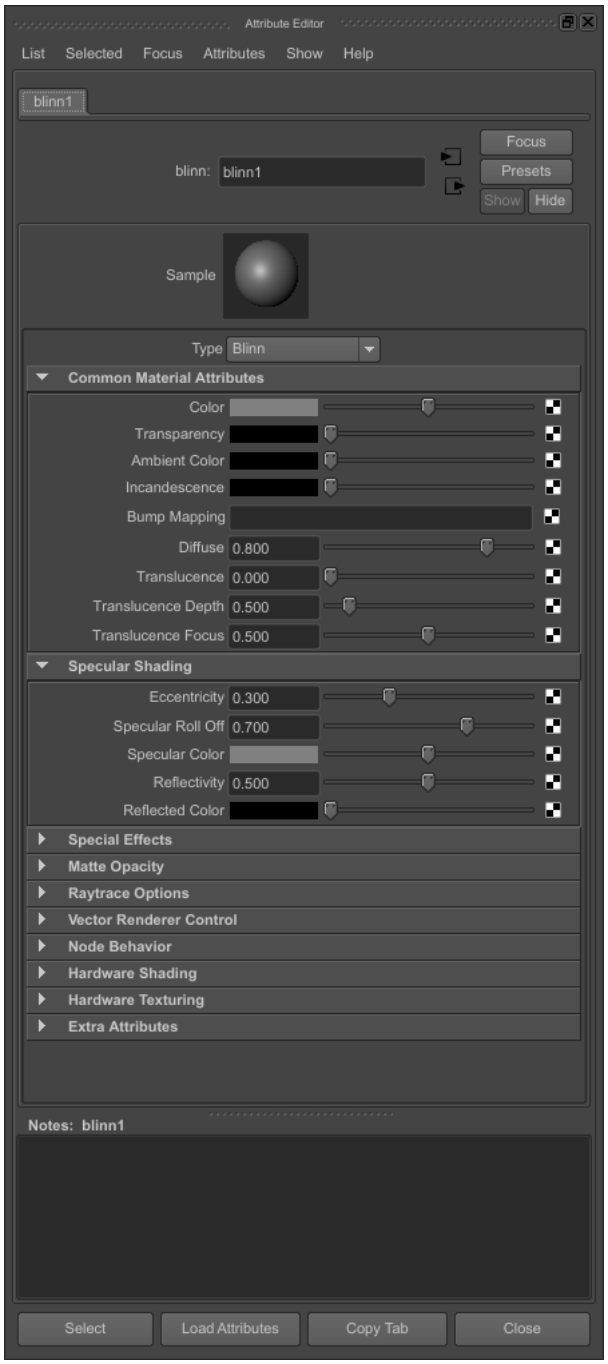
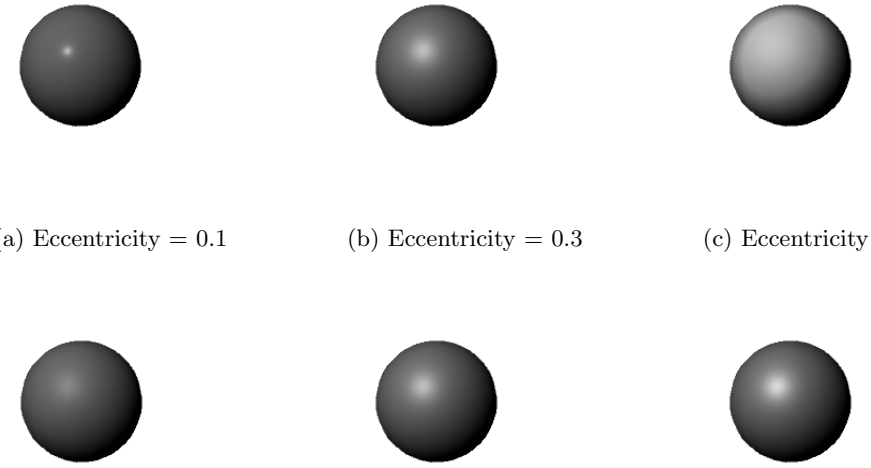


Figure 1.3: Blinn Shader Options



(a) Eccentricity = 0.1 (b) Eccentricity = 0.3 (c) Eccentricity = 0.7

(d) Roll Off = 0.3 (e) Roll Off = 0.7 (f) Roll Off = 0.9

Figure 1.4: Blinn Shader Options

The majority of these variables can be controlled either with static input - such as a numerical value or a color - or with variable input. Variable input can come in the form of procedural or painted textures, where the input is determined by the value of a pixel on the texture. Procedural textures are generated programmatically, and painted textures are created using photographs or manually created images.

Choosing what pixel in the texture input affects which vertex or face on the geometry is called mapping, and it can be done programmatically or using UV coordinates on the geometry. UV mapping is the act of mapping a 3D object onto a 2D plane in which U and V denote the axis of the 2D plane. It is used by surfacing and modeling artists to determine how the pixels of the 2D image will be rendered on each vertex of a 3D object.

The production pipeline is the sequence of processes a production goes through from the initial concepts to the final cut of the film. This pipeline is divided into three major stages: pre-production, production, and post-production. Pre-production largely consists of design work and research, and is where the core story is developed. It is also where some early research is done on a small scale to ensure the ideas are technically achievable. Production is where a team of artists create the content seen on the screen: CG models, effects, lighting, animation, and texturing are all

done in production. The post-production stage is is where all the content generated in production is assembled and any final edits or color correction is done.

Within each major stage of the production pipeline there several departments called workflows such as: modeling, surfacing, rigging, animation, lighting, and effects. Each of these workflow areas contain work items and assets. Work items are the files used to create content such as Autodesk Maya files for modeling, rigging, and animation or Houdini files for effects work. Assets, also known as products, are the files generated by the artist that are used elsewhere in the pipeline such as texture map image files, shader networks, geometry caches, or rendered frames.

To push data from one workflow to the next we use a publish and subscribe system. When artists are ready for their work to be used down the pipeline they will publish it, then the next workfow can subscribe to the necessary data. For example, when modelers are ready to pass their work down to the surfacer and rigger they will publish the Maya file or the mesh as a geometry cache. The surfacer sees published work that is available for them to subscribe to and work with. Ideally data will flow smoothly from one stage in the pipeline to the next as shown in figure 1.5. However in practice the pipeline function is described in figure 1.6. Figure 1.7 shows the pipeline in action on a single shot of *Life of Pi* and the extraordinary quantity of data that the system needs to manage. Both artists and tools must be flexible enough to function in semi-organized chaos.

For the purposes of this thesis, we focus on the surfacing workflow in the pipeline. Figure 1.8 shows the data flow within surfacing, as well as with other adjacent stages of production. Figure 1.9 shows how the tools described in this thesis facilitate the practical flow of surfacing data through the pipeline. The four tools shown are the OBJ Export, DPA Create Project, DPA Export, and Generate Shaders. The OBJ Export publishes data from modeling for access in other workflows. The DPA Create Project subscribes to that OBJ data within the surfacing workflow and sets up a new Mari project for the artist. The DPA Export tool publishes painted texture files for other working files and workflows to access. And finally the Generate Shaders tool subscribes to the texture files and creates shader node networks within Maya for the artist.

Determining how successful a surfacing workflow is for artists is challenging. The most useful indicators in the current Digital Production Arts studio are the consistency of naming conventions, the quantity of surfacing assets generated, the number of iterations or versions created for each map, and a visual assessment of the quality and evolution of the surfacing.

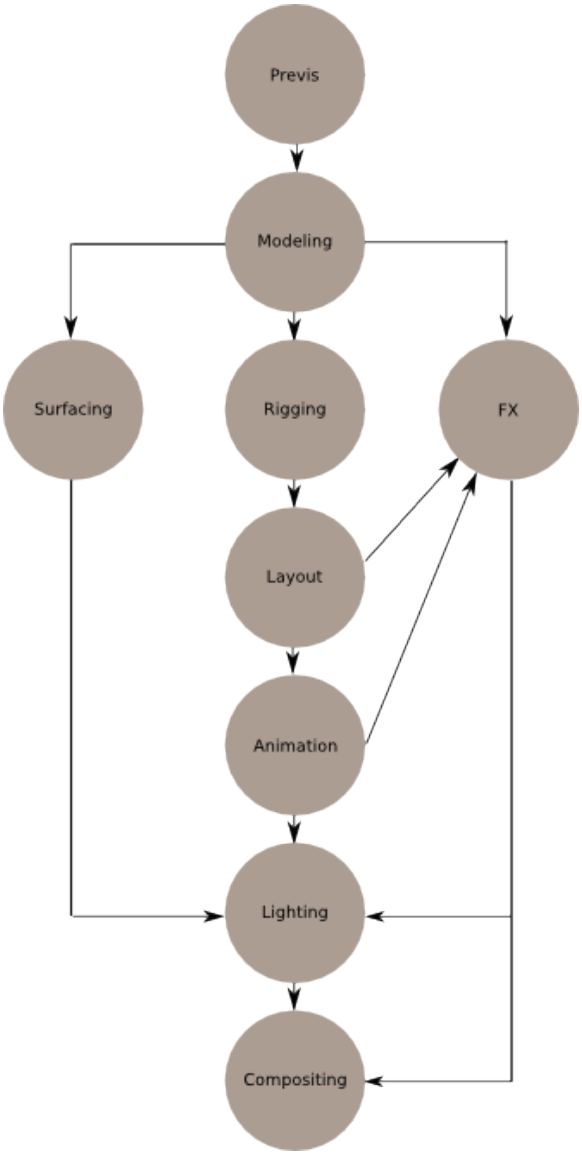


Figure 1.5: An ideal production pipeline

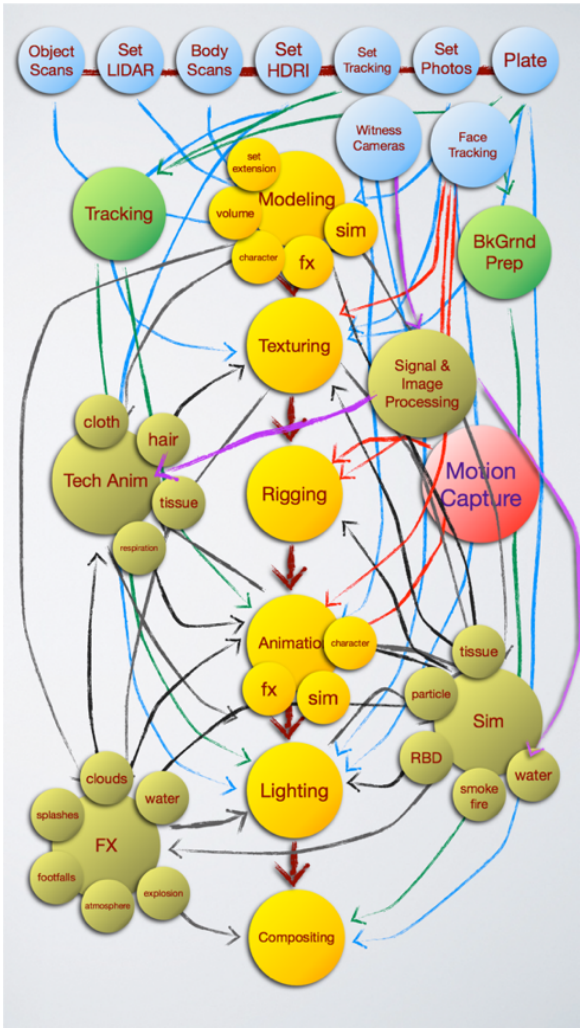


Figure 1.6: A more realistic production pipeline. Courtesy of Dr. Tessendorf[7]

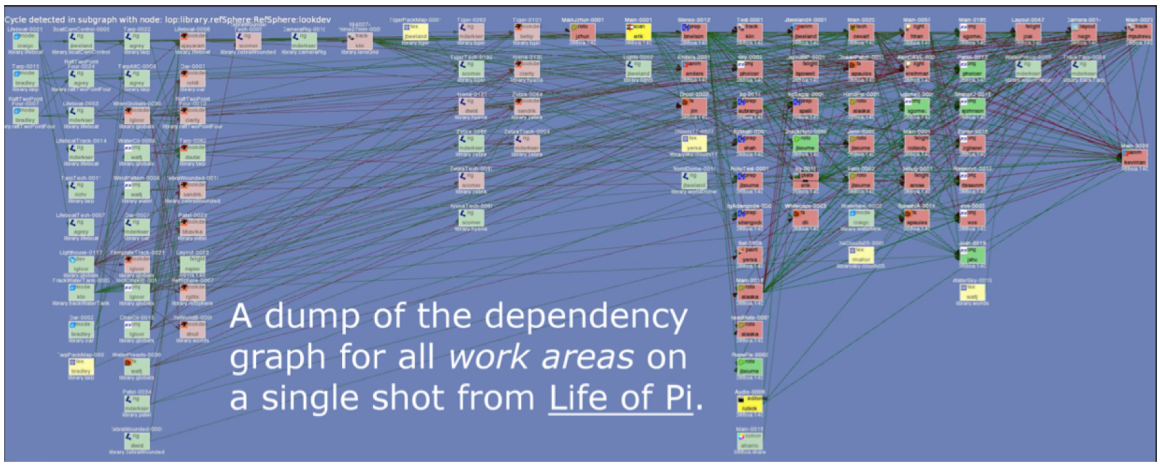


Figure 1.7: Asset dependency graph for a single shot of *Life of Pi*. Courtesy of Dr. Tessendorf[7]

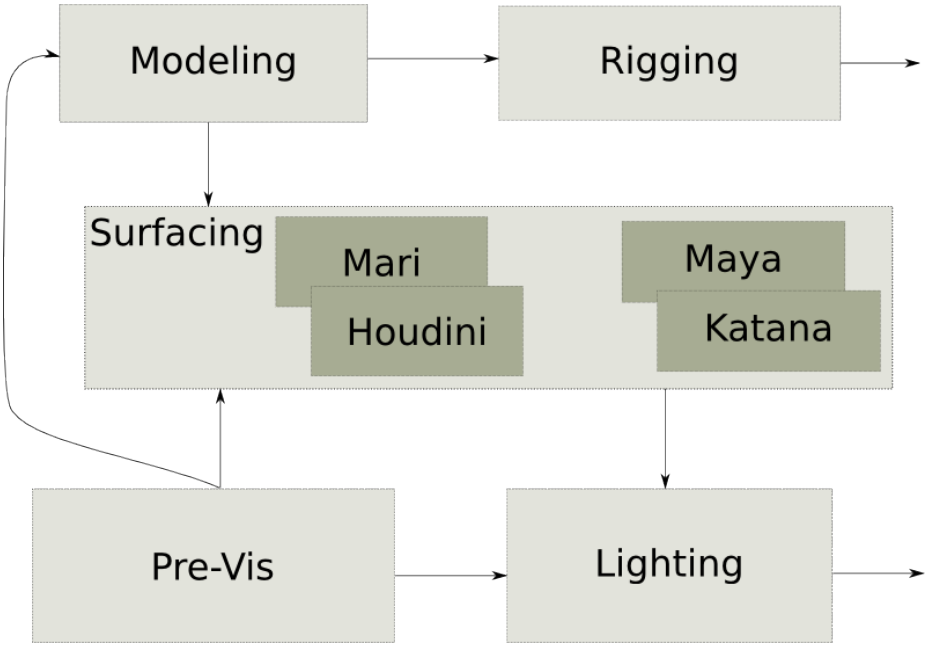


Figure 1.8: An example of the surfacing workflow as it fits in the production pipeline

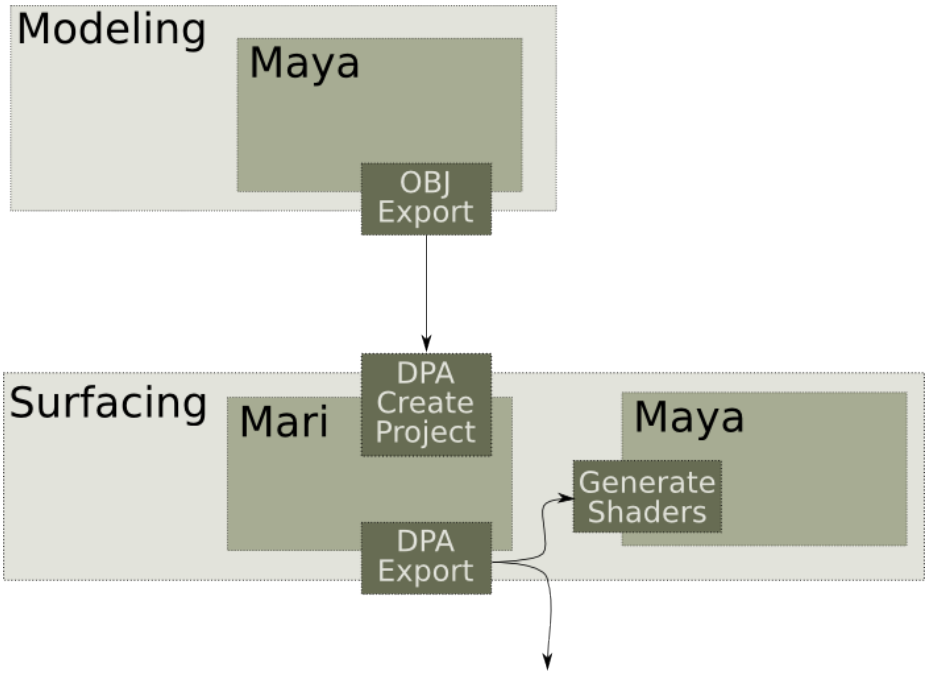


Figure 1.9: Implementation of the DPA surfacing pipeline, in which the data between workflows is connected using the tools discussed in this thesis.

This thesis explains how the surfacing pipeline within the Digital Production Arts studio was designed with these artistic concerns in mind. Chapter 2 discusses the software and tools at our disposal at the beginning of the design process. Chapter 3 explains the design choices and tools shown in 1.9. Chapter 4 discusses the artistic successes of these tools and recommendations for future work.

Chapter 2

Background

When designing a new set of tools for use within an existing pipeline, it is important to first consider the available tools and workflow, and how they address the problems from an artist's perspective. From there the weaknesses and strengths of each tool and task with regard for artist comfort can be assessed. Weaknesses are points in production where preventable mistakes are made and significant time is spent by the artists not creating reviewable material. Once problematic areas are identified, work can begin on patching the problems. This chapter discusses the tools already available in the DPA studio. These tools were created in a way that artists found intuitive and effective.

DPA has a suite of tools to handle file management, publishing, subscribing, rendering, and more, including APIs and command line tools to create assets and organize files on disk[4]. Collectively these tools are referred to as "the pipeline" within the studio. The production pipeline is not a static environment. It constantly shifts and evolves to meet the ever changing demands of production. As our films become more complex, the pipeline has to adapt, and any major new tools being added must also be able to adapt.

There are two main iterations of the DPA pipeline that are relevant here. The current pipeline was developed by Dr. Jerry Tessendorf and DPA students in 2012. The first phase of the surfacing pipeline was developed for it. The second major iteration of the DPA pipeline is under development by Josh Tomlinson and DPA students. The second phase of the surfacing pipeline is built for this new version of the DPA pipeline.

Within the DPA studio we use several existing software packages. The software available

includes Autodesk Maya, the Foundry's Mari, and Pixar's Renderman. Mari is used for painting texture maps, RenderMan is used for its shaders and rendering, and Maya is used for modeling, shader creation, lighting, and its rendering package. For the purposes of this thesis, Maya and Mari are the two programs we will discuss as we access Renderman's shaders through the Maya interface.

2.1 Autodesk Maya

Autodesk Maya is the 3D software package currently used by the Digital Production Arts studio for modeling, rigging, creating shader nodes, layout, animation, and lighting. For the purposes of this thesis, we focus on its use for creating shaders for surfacing 3D objects.

Maya uses a node based structure for handling its shaders in which a Shading Group is the root node. The Shading Group is connected to several different types of nodes, most notably surface nodes or displacement nodes. Surface nodes control all of the common aspects of material interaction with light, including color, refractions, reflections, transparency, bump, and more depending on the shader type. Common surface nodes include Blinn, Phong, Phong-E, Mental Ray materials, and Render Man shaders. Because of how Maya connects displacement attributes, the displacement is handled by a separate node from the surface.

Maps are attached to the shader nodes using file nodes that contain several properties including the file path, filtering options, and color space options. The file nodes can be connected to nearly any attribute in the shader node. Also several nodes exist to manipulate the maps that can be inserted between the file and shader nodes.

Once a shader network is assembled, whether it is a single Blinn node or a complex custom network, it is then attached to the 3D mesh, either to the object as a whole, or to individual faces. The entire node network can be manipulated through Maya's Hypershade window, as shown in figure 2.1, which provides a GUI to view a variety of node networks from surfacing to lighting. Additionally, Maya provides tools for UV unwrapping in order to generate appropriate flat surfaces for an artist to paint using the UV Editor window.

Maya provides the user with a variety of scripting tools to facilitate plug-in development. It uses a combination of Python based API's and the MEL scripting language that ties into nearly every aspect of Maya's functionality. It also provides a bridge between the Python API and the MEL language using a package called PyMEL, and the ability to run MEL commands through the

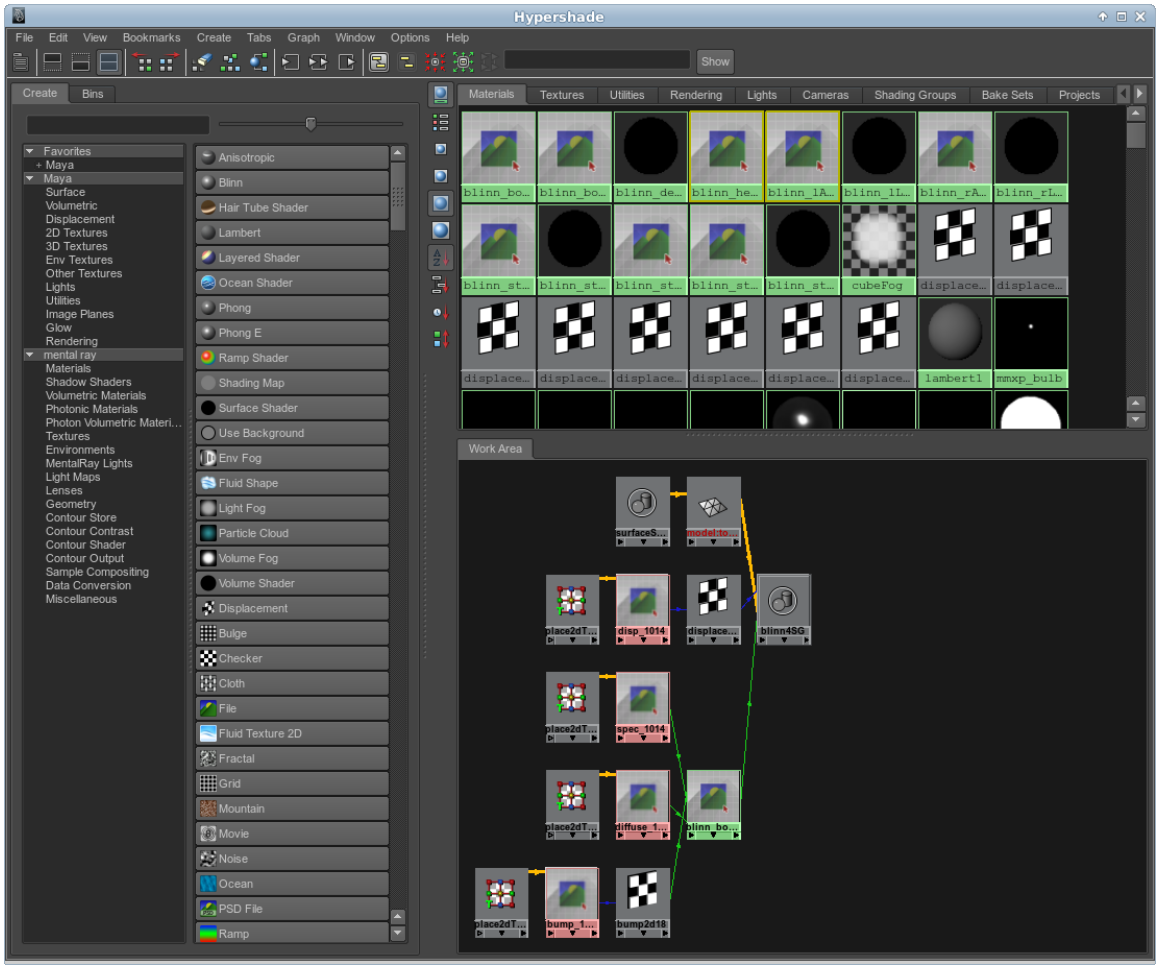


Figure 2.1: Maya's Hypershade Window

Python API.

Through the Python based API Maya provides access to customized elements of PyQt to build GUIs. This allows for functional and consistent plug-in designs.

2.2 Foundry's Mari

Beginning in the summer of 2013, DPA began using the Foundry's Mari 1.6 to handle the bulk of its surfacing. This package is designed to allow an artist to paint on a 3D object with all the functionality and ease of Adobe Photoshop, but with the convenience of being able to paint directly on a 3D mesh. It provides access to several 3D views including a perspective camera, orthographic camera, and traditional UV patches.

Mari uses a projection system to allow users to paint on the 3D model. It stores the artist's painting as a 2D map in a buffer, projecting it onto the 3D object from the current camera. There is a versatile masking tool to prevent unwanted stretched projection and control the application of the painting. Mari allows the user to work on several different maps on the same object, each with their own layers that behave intuitively to users familiar with other digital painting packages. Each map can be exported in a set of 2D images such as tifs or pngs. One image is exported for each UV patch. As shown in figure 2.2, Mari provides a convenient way to paint on a 3D object while seeing both the UV patches and the 3D image. For an artist this makes Mari an extremely comfortable way of visualizing the painting of a 2D map for a 3D mesh.

The GUI in Mari is built up out of a series of palettes. Each palette is a GUI element that can be docked in the main window, or pulled off into a stand-alone window. This allows users to easily manipulate their painting tools to construct a work environment that is comfortable for them. Examples of palettes include a brush palette, color picker, layer manager, channel manager, shader manager, and python console.

Mari uses a Python based API to allow users to easily access their GUI information and package functionality to facilitate plug-in development. PyQt is used to build the GUIs, and provide a built in Python console to allow for convenient development.

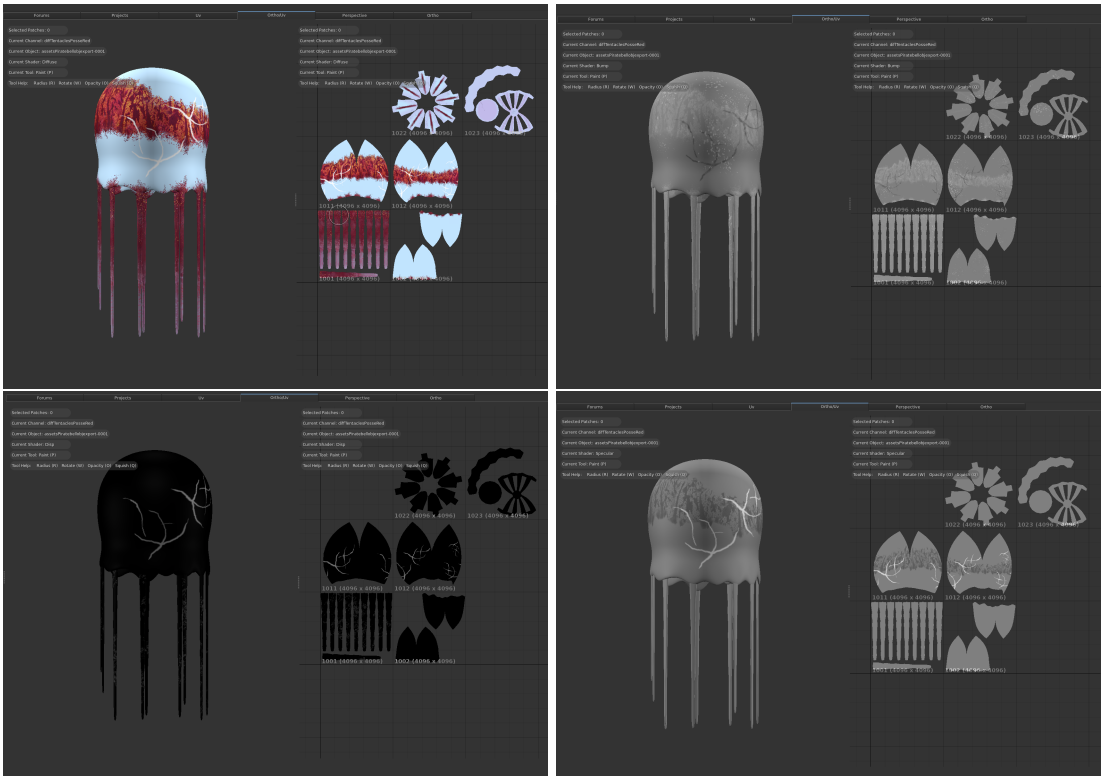


Figure 2.2: Mari GUI

2.3 Initial Workflow

An important component to designing an effective workflow is to be aware of existing structures. This can help identify the most problematic areas of the pipeline and any choke points that are slowing production down. As of summer 2013 the DPA production workflow for painting texture maps in Mari contained several steps that invited user error and sloppy workarounds from stressed artists:

1. Creating a Wavefront OBJ file required exiting the 3D graphics package and manually creating an asset within the pipeline through the command terminal. It also required manually changing file permissions, and applying and reverting a modifier on the mesh.

2. Creating a new project in Mari manually, including setting up channel names, channel settings, and OBJ settings.

3. Exporting from Mari required a significant amount of time outside of the Mari package to create pipeline assets on the command line, being aware of the current version of the assets, and then exporting each map into the correct location, then finally changing permissions of all the exported assets. The export process was two steps: flattening the channel layers, then exporting. For a single map on a single object it is a manageable process. In the instance of the 2013 production of QA-ARM-A, when attempting to texture 7 unique arm models each with 3 maps attached, that process ended up taking as much time to step through as actually painting the textures.

4. Shader creations in Maya required the artist to set up consistent and well named file nodes, shader nodes, and utility nodes. Then for each file node navigating to the correct file path. Once this was set up, it also required the artist to maintain the most recent file path to a texture manually.

In the event of an object with 5 UV patches, and 4 maps (for example: Diffuse, Specular, Bump, and Displacement) that means the artist would need to create 5 shader nodes, and 20 file nodes. Then every time artists updated a texture file they would need to create a new version of the texture asset, change the permissions appropriately, and manually update each file node.

Needless to say this system invites user error. Misnamed files, incorrect permissions, and incorrectly updated file nodes can (and have) caused rendering issues. Overwriting old files to avoid

having to create an asset and update the file nodes in Maya can (and have) caused problems with losing data and being unable to return to an older version. Incorrect export settings can (and have) caused issues with image resolution and color space when rendering.

The benefits of having a system of tools to manage these problems can be seen by looking at how much work it would take to create a complex shader such as the sunken canon in *Peanut Butter Jelly*[1].

Map Name	Versions	Files
Barnacle	19	54
Fungus	23	64
Cellular	17	46
Algae	22	61
Bubble	22	61
Barnacles	8	19
Growth	36	100
Transparency	29	85
Disp	58	172
Diffuse	30	88
Bump	32	94
Specular	28	82
Total Versions	324	
Total Files	926	

Table 2.1: Surfacing assets for Peanut Butter Jelly's cannon

Table 2.1 shows the volume of maps and iterations that an artist would have to manage. To create this using the same system we had in place during the summer 2013 productions artists would have needed to manually create each of the 12 map assets on the command line. They then would have needed to manually create an asset for each version - that is 324 total commands typed into the terminal, at a minimum. At this point an artist looking to take shortcuts might consider just overwriting old data, which creates massive problems if they need to return to a previous. The

cannon only uses two UV patches, but that means for every map the artist needs to create two file nodes. Then they need to manually update the file path on both those nodes every time they export a new map. That means 24 file nodes which all must have the most recent file path to a texture associated with it.

Now consider a more complicated example, such as the hero characters in *Peanut Butter Jelly*[1][3].

The two stars of *Peanut Butter Jelly* are a flyboy jellyfish and a pirate jellyfish. Each hero has a gang of companions that share the hero's basic look. The jellyfish model for the pirate has 8 UV patches, and the flyboy had 3 UV patches. While the gangs shared the model and core design with the captains, they needed unique changes to the texture maps to ensure that each character had a unique look[3]. This is a massive organizational challenge. The artists needed a system flexible enough to allow them to manage these different looks so they could focus on the artistic direction of the characters. Tables 2.2, 2.3, and 2.4 are a sample of the sheer volume of maps and characters that were required. Updating these manually would have been impossible.

Map Name	Versions	Files
Blur	3	10
Disp	6	13
Eyemap	8	20
5	21	39
4	18	33
3	7	19
2	12	26
1	8	19
Total Versions	83	
Total Files	179	

Table 2.2: Surfacing assets for Peanut Butter Jelly's Flyboy gang members

Map Name	Versions	Files
Transparency	3	9
Specular	5	17
Disp	18	66
Bump	5	16
Diffuse	36	137
Eyemap	59	228
Blur	24	88
Total Versions	150	
Total Files	561	

Table 2.3: Surfacing assets for Peanut Butter Jelly's Hero Flyboy

Map Name	Versions	Files
Transparency	4	24
Specular	4	24
Disp	5	30
Bump	8	48
Diffuse	33	198
Total Versions	54	
Total Files	324	

Table 2.4: Surfacing assets for Peanut Butter Jelly's Pirate Captain

Chapter 3

Design and Implementation

Based on the problematic components of the preexisting pipeline described in the previous chapter, breaking the surfacing pipeline tools into smaller solvable problems is clearer. The goal in the design of these tools is to get everyone on the same page, minimize file system navigation, and automatically enforce naming conventions. Four major points of concern became clear: OBJ exporting from Maya, starting up a new Mari project, Mari texture file export, and shader generation in Maya.

Implementation and deployment was done in two major stages - a first and a second stage - with many minor patches and hot-fixes in between. The reason for this was to account for the organic nature of the pipeline. The first stage was to make sure the basic principles were sound. The second stage was to identify the changes in artists needs as the complexity of projects ramped up.

3.1 OBJ Export

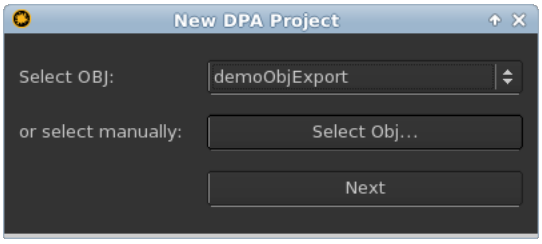
Due to the fact that transferring the mesh from Maya to Mari requires consistent mesh settings and asset creation, it is logical to automate the process of generating OBJ files, particularly because there are enough steps to be suspect to major errors. Maya's smooth mesh preview requires a permanent alteration to the mesh in order to correctly export smoothed faces in an OBJ. An artist could either fail to smooth the mesh before export and create problems with UVs not lining up between Mari and Maya, or fail to undo the application of the preview and cause problems with

the density of the model down the pipeline.

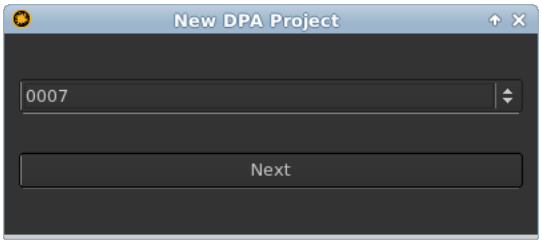
The final implementation is a one-click tool that generates the asset, exports a smoothed OBJ, and doesn't damage the model. This saves artists significant time and effort, allowing more time for texturing.

3.2 Create Mari Project

In order to alleviate the burden of having to locate the OBJ file and initialize a project with the correct settings, the New DPA Project tool in Mari provides a layer between the artist and the file system. The tool populates a list of all published OBJ files in the pipeline for an artist to easily navigate, and provides a back-up option to open a file browser to locate an OBJ that may be outside of the pipeline, as shown in Figure 3.1. Then it programatically creates a new Mari project and generates several common maps that the artist will likely use. These maps are created with standard naming conventions and settings to ensure that the artist can start painting as quickly as possible.



(a) Selecting the OBJ



(b) Selecting the OBJ version

Figure 3.1: Create New Mari Project Windows

3.3 Mari Export

Mari's patch structure generates many files, which creates a substantial file management issue. The surfacing pipeline focuses on reducing the user's interactions with the file system by creating the asset directories, managing the current version of the texture files, and handling the naming conventions for the maps. In order to ensure that the user does not have to go through the asset directories to manually find the most recent files, a master symbolic link points to the most recent texture files.

The GUI for this process needed to fit into Mari's palette structure unobtrusively and allow for the export of either the selected or all the painted channels. To accommodate RenderMan, it allows for tex conversion as well. Tex conversion can either be done serially on the local machine or in parallel through the DPA queuing system. Additionally, artists need the ability to have hi and low quality texture images exported. The only practical difference between them is the path of the master link. This allows artists to switch the quality of the image by only switching the words "hi" and "lo" in the master link path. This combination of features reduces the decisions an artist and distractions an artist faces while painting.

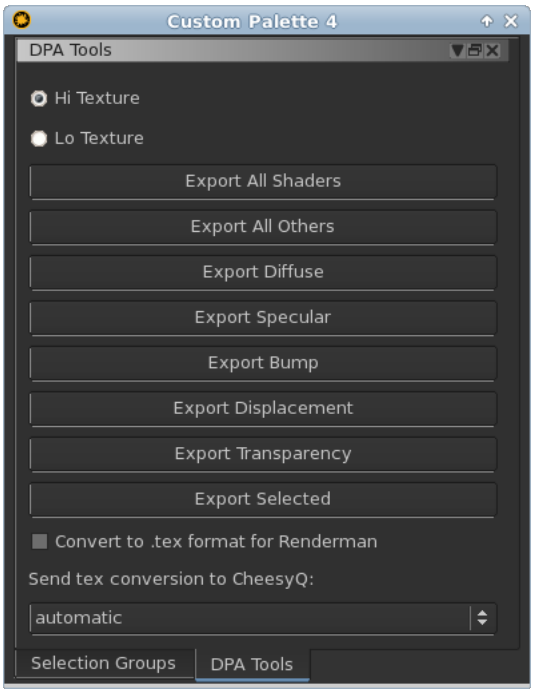


Figure 3.2: First Generation Mari Export Panel

3.4 Shader Creation

Creating and maintaining shader nodes in Maya is exceptionally time consuming, motivating artists to take shortcuts and limit the sophistication of shader node networks. Using the surfacing pipeline and the Mari export tools, several pieces of information are available to reduce this burden:

Texture files are programatically locatable, as they are stored correctly within the pipeline. Given the naming convention applied during Mari Export, texture files can be connected to shader nodes automatically. For example, the diffuse map should connect to the color attribute, and the specular map should connect to either the specular color or specular intensity attributes. More complex schemes, such as connecting a displacement map to an adjustment node and then to the shader group, are also possible as automated steps.

Applying Maya's scripting abilities and node based shading networks, shaders can be generated for artists. If an object has 5 uv patches and requires 3 maps (eg. diffuse, bump, and specular) with textures exported from Mari, that would be 15 file nodes and paths. A total of 5 shaders, and 15 connections would need to be created and named. Manually creating them is a a significant time burden for artists, particularly if they must also update each node when new texture files are generated.

This portion of the surfacing pipeline has two parts. The first part is a shader generation tool that works in the DPA pipeline. The second part takes the data gathered from artists and the new tools being created for the new DPA pipeline to create a more robust surfacing pipeline.

The first part of the shader creation was built exclusively for Maya and went into use September 2013 for the production *Alien Oasis*. As of February 2015 it is still in use while the new DPA pipeline is being phased in.

The GUI was designed to mimic the simplicity of the Mari export palette, as shown in 3.2 and 3.3. The basic choices an artist needs to make are: shader node type and maps to attach.

When generating the node networks the three most important components were functionality, consistent naming, and uniform file settings. The ability to manage the color space settings of a production from the moment the files are set up enables the team to prevent color issues further down the pipeline in lighting and rendering.

In the first stage of the pipeline the naming conventions of each node type was not as consistent as it should have been, but was an improvement from the automatically generated 'file',

Figure 3.3: Maya Shader Generation GUI

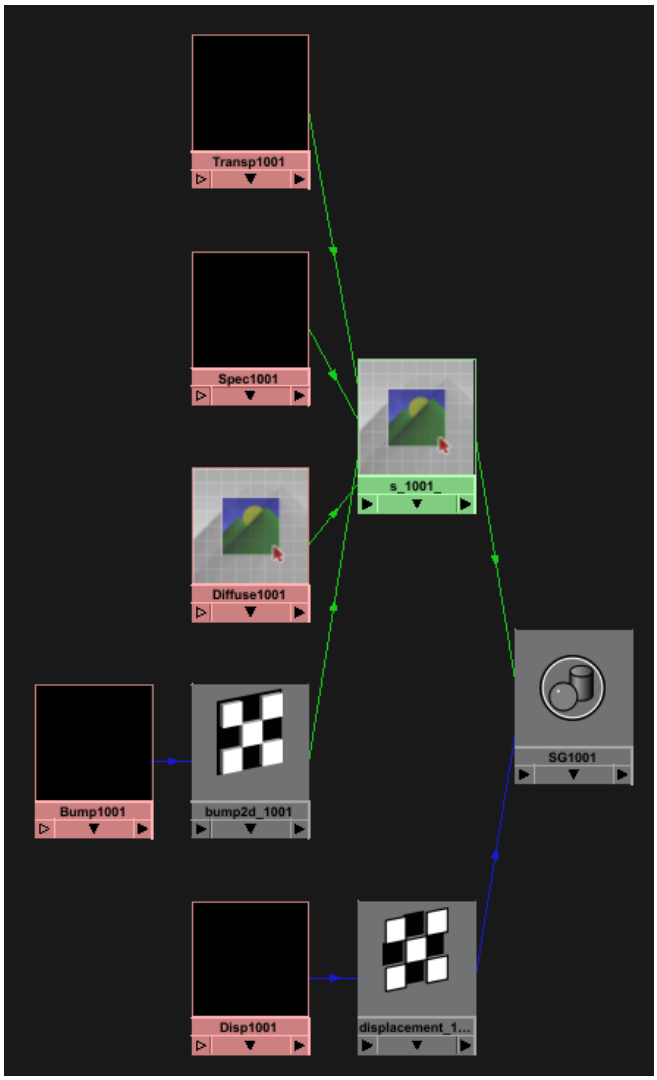


Figure 3.4: An example of the generated node network

'file1', 'file18' node names that are easy for users to leave when crunched for time.

Node network naming conventions, as shown in figure 3.4, in the first stage tool are:

- Shader node = s_10[uv]_ to immediately tell artists that they are dealing with the shader from the given uv patch, where [uv] is the patch identifier from Mari. The additional _ is to prevent Maya from treating the uv patch like a number to increment. This way if a second node network is generated it's automatically named "s_10[uv]_1" instead of "s_10[uv+1]'. It also gives artists a place to add additional descriptive information such as 's_10[uv]_metal' or 's_10[uv]_burned' if they have decided to duplicate the node network and have alternate shader attributes.

- File nodes = [mapName]10[uv] to immediately let the artist know what map is linked and for what uv patch, where [mapName] is the channel name from Mari (such as Diffuse or Bump), and [uv] is the patch identifier.

- Utility nodes = [utilType]_10[uv] so that the user knows what the function of the utility is, where [utilType] is the Maya specific shader type and [uv] is the patch identifier. This is particularly helpful when Maya re-uses icons, or has similar icons for different utility nodes such as the bump and displacement utilities.

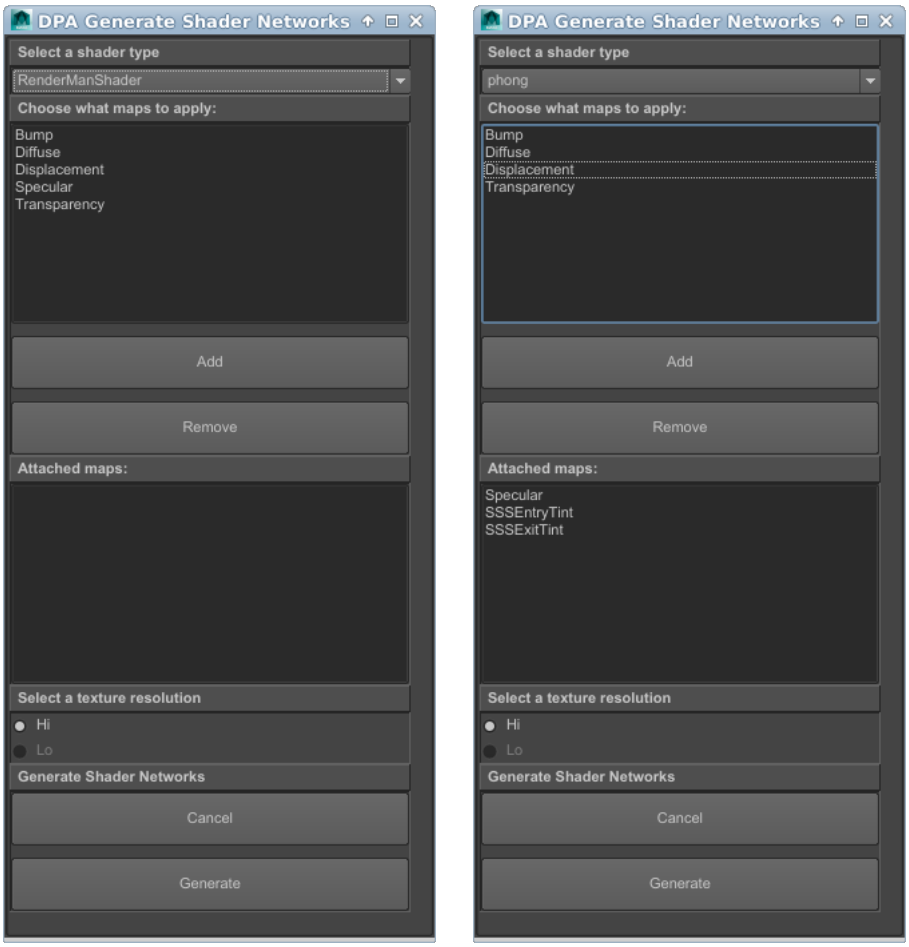
- Shading group = SG10[uv] is the root node for any shader in Maya, where [uv] is the patch identifier.

After a year in production, the first stage of shader generation in the DPA pipeline was successful. It has worked through two productions and several hot-fixes that increased artist efficiency, as shown table 4.1. There is a dramatic increase in the quantity and variety of texture assets from QA-ARM-A and Robo+Repair to Alien Oasis and Peanut Butter Jelly.

However, with a major overhaul of the core pipeline in progress and new production needs, it was clear that the first stage of the shader generation would not be robust enough for long term use. A more flexible and dynamic option was needed. The most critical components of the first stage of code to be addressed were: handle custom maps in node network generation, custom maps in Mari project startup, and integrating the new products, publishing, and subscription systems.

Such fundamental updates were too far-reaching to justify patching the older code. Starting over and using lessons learned from the previous tools made more sense for a cleaner and more usable

tool. Because of the increasing need to handle custom maps and shaders[1][6], the second stage of the surfacing pipeline was designed to theoretically work for any map.



(a) Available maps for RenderMan's General Purpose Shader, none selected

(b) Available maps for a Phong shader, with three maps selected

Figure 3.5: Maya Shader Generation GUI, showing how users selects maps to be used in their generated shader network

In order to achieve that level of flexibility the shader generation code needs to approach texture maps and their connections to shader networks more abstractly instead of hard-coding that information into the software. Additionally, storing the map information and shader network connections externally to the main shader generation code then dynamically loading that data when the tool is called up makes adding new map and connection information possible without editing the code.

The new DPA pipeline contains an API for configuration files that is a natural method for storing this connection information. These configuration files are loaded when the tool is launched, allowing the artists to define maps and connections without needing to understand the software generation code. All they should need to know is the nodes and network connections required by Maya's hypershade.

From an artist's perspective, figure 3.5 illustrates how configuration files are presented with options to load when creating or modifying shader networks.

The given configuration structure is essentially a way of organizing key/value pairs. Each attribute in the configuration file is a string key, and its value can be anything including floats, strings, lists, and another configuration file.

Each map has its own configuration file that is roughly defined as having three main parameters: software package, name, and color. The name is the name of map, the color is considered the default color for the map if it hasn't been painted yet, and then each software package is an embedded configuration file for that product. Within each tool the configuration file defines all of the information that tool will require to construct a shader. The following describes the core structure of a map configuration file and the value types expected:

```
toolName: [configuration file]
name: [string]
defaultColor: [r,g,b,a]
```

A shortened example of a configuration file for a bump map:

```
Maya: [config file]
name: Bump
defaultColor: [0.5,0.5,0.5,1.0]
```

For Maya, the tool configuration defines only one type of key: shader type. Users can define as many shader types as they need, and each one will map to the configuration file for that shader. The key is also passed to Maya on shader creation, so it must match the string that Maya uses to identify that shader type.

Within each shader another set of keys is defined. At a minimum the shader will need two keys: nodes and connections. This will inform the shader generation code what nodes to create and

how to connect them. Additional keys can include file extensions, shader file locations for custom RenderMan shaders, and custom RenderMan attributes that need to be attached to nodes. These are designed to be as flexible as possible so that if a new shader type needs custom information it is relatively easy to add it to the shader generation.

The following is an example of the Maya shader configurations for a bump map:

```
blinn:
    nodes: [file, bump2d]
    connections: [[file, outAlpha, bump2d, bumpValue],
                  [bump2d, outNormal, shader, normalCamera]]
    ext: tif

phong:
    nodes: [file, bump2d]
    connections: [[file, outAlpha, bump2d, bumpValue],
                  [bump2d, outNormal, shader, normalCamera]]

RenderManShader:
    nodes: []
    connections: [[shaderName, sBumpMap, path]]
    shaderLoc:
        /DPA/moosefs/PeanutButterJelly/prod/share/assets/
        rmLex/0001/hi/sl/lexShader.slo
    ext: tex

mia_material_x_passes:
    nodes: [file,bump2d]
    connections: [[file,outAlpha,bump2d,bumpValue],
                  [bump2d,outNormal,shader,overall_bump]]
```

Chapter 4

Results

The quality of the surfacing work being done in the DPA studio has dramatically improved since the implementation of these tools. During the production of *QA-ARM-A* there was a huge struggle with managing the flow of data through the pipeline, and our solution was to radically simplify our shader networks and scene so that we could complete the film[5]. As shown in Figure 4.1, as our surfacing pipeline has grown more sophisticated, so has the quality of our work. *The Water is Always Bluer...* from 2012 was created with no pipeline in place, *QA-ARM-A* was created in 2013 with a basic production pipeline in place, but there was no surfacing pipeline. *Peanut Butter Jelly* was completed in 2015 with the benefit of the tools described in this thesis.

We can see the improvement in the generated data in addition to the visual results. As shown in Table 4.1, the surfacing pipeline enabled a dramatic increase in iterations and adherence to the pipeline procedures. The two films *QA-ARM-A* and *Robo+Repair* were created with no standard surfacing pipeline, so artists were conservative about versioning their files and the quantity of maps they were creating. *Alien Oasis* and *Peanut Butter Jelly* were both created using the surfacing pipeline described in this thesis and the output of texture assets dramatically increased. Examples of the surfacing created in *Alien Oasis* can be seen in 4.2. Notably, *Alien Oasis* and *Robo+Repair* have nearly the same number of work items, yet *Alien Oasis* produced thousands of versions compared to *Robo+Repair*'s 56 versions. The shows currently in production are on track for continuing that trend.

Less easily calculated are the ease of starting up a new Mari project, making modifications to existing textures, the reduced learning curve to get started in the surfacing pipeline, and the ease

(a) *The Water is Always Bluer...* (2012)

(b) *QA-ARM-A* (2013)

(c) *Peanut Butter Jelly* (2015)

Figure 4.1: Final Frames

Show	Workitems	versions	files
QAARMA *	26	66	387
Robo+Repair *	29	56	642
Alien Oasis (Fall 2013)	30	3336	14420
Peanut Butter Jelly (2014)	116	3938	28560
2015TARS+	38	1290	7130
Cylon+	37	872	4353
* no surfacing pipeline in place - numbers are a minimum + still in production			

Table 4.1: Surfacing assets in various productions

of passing work off to another artist. However, in conversations with surfacing artists they have expressed that they are spending their time actually surfacing rather than performing tedious and repetitive tasks. Compared to the 2013 production of QA-ARM-A, where each texturing artist had a unique workflow and managed files manually, this is a huge improvement.

The basic structure of the tools have proven themselves, and made increasingly more complex productions possible. Doing the work of PBJ and Alien Oasis would have been extraordinarily difficult without tools. Due to the number of artists, varying skill sets of the artists, and the volume and complexity of shaders, these productions would likely not have been possible without a surfacing pipeline in place.

The greatest challenge to designing and implementing the DPA surfacing workflow was finding a design that would enhance the artistic output of the texturing and improve how quickly an artist can go from a blank mesh to a first pass. This required not only a personal understanding of what tools an artist needs to create, but also communication with the other artists in the studio to ensure the workflow was as comfortable as possible. A constant flow of information and feedback between developers and artists was critical to achieving a solution to this artistic problem.

Since the implementation of the surfacing pipeline, two productions have been completed and two more are in progress. In the completed productions of *Alien Oasis* and *Peanut Butter Jelly*, there are many examples of artists benefiting from these tools - in particular the jellyfish in *Peanut Butter Jelly* [3] and managing the large number of texture artists in *Alien Oasis*.

While working on *Alien Oasis* we had twice as many students working on the film as we had for our previous six productions - the typical team size had been approximately 5 students on earlier productions, and *Alien Oasis* had 10 students. This made ensuring that all the artists were

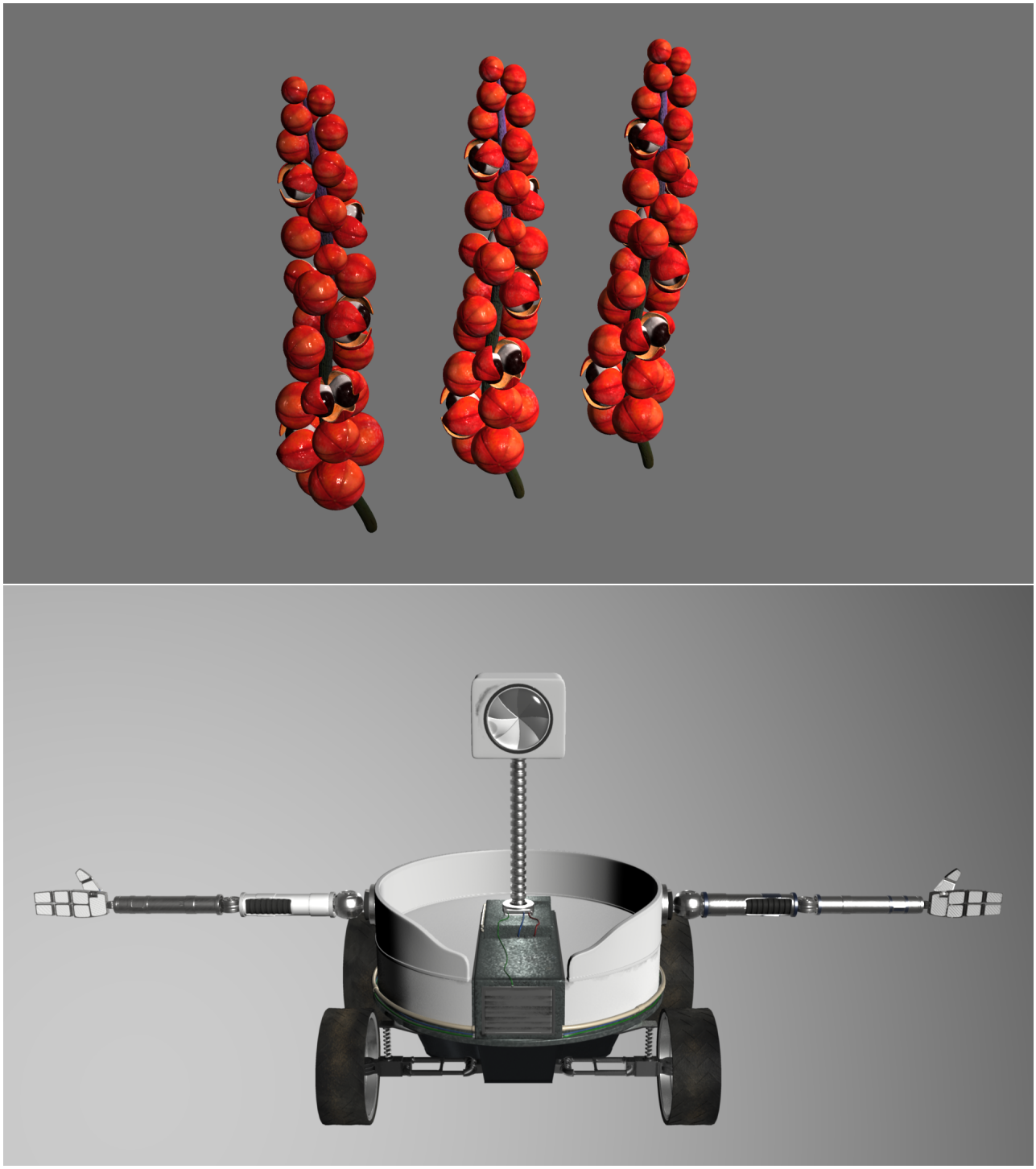


Figure 4.2: Alien Oasis Surfacing

on the same page significantly more of a challenge as we had a large number of people with varying degrees of experience and comfort with surfacing. Having the pipeline in place ensured that every artist started with a consistent environment, and when we needed to make production-wide changes, gave us a way to enforce those changes across all surfacing work items. This happened several times during the production. When we decided to change the organization of generated assets so that each output map was a unique asset, all that was required of the artists was a one time update to their shader file paths. Also, when we realized that the file settings on the output texture maps was incorrect, a one time re-export was all that was required to push the change through all workitems. The final example of the successes of the surfacing pipeline was the ease of switching work between artists. The effort to make texture resolutions more efficient required only one artist to go back through the surfacing work of many to resize and re-export, instead of calling back every surfacing artist.

Most recently, the surfacing pipeline was heavily used in *Peanut Butter Jelly*, which contained a dramatically more complex shading work than any of our previous productions. From lichen and coral covered rocks, to sunken ships, to jellyfish there was a variety of organic and layered surfaces to manage. In order to handle the rendering demands of the production, it was decided to use Pixar's RenderMan renderer and shaders to create the film. This required the surfacing pipeline to make several changes to accommodate RenderMan's tex format, allowing the artist to convert to tex on export from Mari rather than manually. Additionally, the export tool gives the option for artists to convert the textures in parallel through a queuing system rather than locking their computer up while converting textures serially.

One of the biggest and most important challenges was surfacing the jellyfish. As the stars of the film, they needed to have a strong artistic vision, be easily recognized as jellyfish and unique characters, and be customizable to create variations for crowds[1]. To accomplish this the artist creating the jellyfish relied heavily on the ability to create custom maps to control everything from diffuse color, to masking refractions, to control the final look[3]. Having the surfacing pipeline in place alleviated the burden of transfering data between Maya and Mari when she changed the geometry or UVs on the object, and when rapidly updating iterations of the texture map. This allowed her to focus on the look and feel of the jellyfish artistically and in a workflow that was intuitive to her as an artist, rather than fighting the file system.

4.1 Recommendations for future work

While the current status of the surfacing pipeline is stable and functional, the demands of production are constantly changing. Looking towards the future use of these tools there are several features that could be implemented to enable artists to create realistic textures more easily:

1. With the current setup of using configuration files, the natural first step to making the system more usable will be to expand the number of available maps. The larger variety of maps that exist, the less set up work artists will have to do in order to get from their first pass at painting to their first render.

2. Currently the generation of configuration files is done manually. Having a user interface to help create the files, or the ability to take a shader network and convert it into a configuration file would make the system substantially more usable for custom node networks.

3. The configuration files are designed so that we can expand the shader generation beyond Maya. Currently all the shader generation is being done in Maya, but it could just as readily benefit artists using Houdini to control effects maps, or artists using Katana if future productions use that application for texturing and lighting.

4. With UV mapping being such a time-consuming and frustrating point in our productions, integrating PTex into the surfacing pipeline would be a significant improvement in efficiency, freeing both the modeling and surfacing artists from the tedious and time consuming task.

5. Newer versions of RenderMan include built in support for textures exported by Mari that would be worth looking into, as would integrating RenderMan's slim into the shader generation.

Bibliography

[1] Alex Beaty. Peanut butter jelly: An animated short film. Master's thesis, Clemson University, May 2015.

[2] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, October 1976.

[3] Brianne Campbell. Surfacing jellyfish for peanut butter jelly. Master's thesis, Clemson University, May 2015.

[4] Timothy Curtis. Efficient control of assets in a modern production pipeline. Master's thesis, Clemson University, May 2014.

[5] Timothy Curtis. Lighting and compositing for qa-arm-a. Master's thesis, Clemson University, May 2014.

[6] Karen Stritzinger. The rusterizer: An art-directable and semi-procedural tool for generating rust surfaces. Master's thesis, Clemson University, May 2014.

[7] Dr. Jerry Tessendorf. private communication, 2015.