Technology Review for Better Graphics For A Robotics Grasping GUI

Shady Robots

Group 12

Justin Bibler

Matthew Huang

Daniel Goh

CS461: Senior Software Engineering Project

Fall 2016

November 14, 2016

Abstract: Our customer is using a simulation to create visuals that are used for online data collection. This simulation is using outdated libraries which result in outdated graphics. We, as the supplier, compare and contrast the different technology and methods that will be used during the phase of the project to ensure the project's success.

1 INTRODUCTION

The technology review document is used to determine the best approach of implementation for the project. Clause 2 lists the references made to other documents. Clause 3 lists each team member's role in the project and accomplishment goals. Clauses 4 to 6 are the technology review that is carried out by each team member. Clause 4 reviews the technology to be used to carry out runtime analysis. performance benchmarks, and best practices to implement shadows. Clause 5 reviews the best practices to implement shaders, best practices to implement silhouettes, and technology to be used to ensure maintainability of the project's source code. Clause 6 review usability inspection methods, run user interviews, and tools to analyze the respondents data.

2 ROLES

2.1 Justin Bibler

My roles within this project vary from each other, and are not one cohesive singular responsibility. I have two responsibilities that have a conclusive end, and a responsibility that is maintained throughout the entire project. The first of the two conclusive responsibilities that I have is to locate where in the code the rendering is happening. This responsibility is critically important to the entirety of the group, as without knowledge of the location of the rendering function it is difficult to implement shaders. Secondly, I will be overseeing the implementation of the shadows. Shadows help define where 3d objects are within a computer simulation, thus our client requested them to be implemented to improve the current graphics. Finally, my last responsibility is to assure code maintainability. This means that once our group has finished with the project our additions to the code will be well documented, commented, and easily understandable. Making it easy for our shader implementations to be changed and adjusted easily to fit our client's needs.

Within the "Identification of Rendering loop, Methods of code maintainability, and Shadows" section I compare and contrast three different methods and tools available to conduct and maintain my responsibilities. How the comparison of methods and tools will be conducted is: I will first explain what the responsibility is and it's relevance to the group and the project. I will then list a tool or method and how it is used or conducted. Next, I will list and analyze the benefits and negatives of a tools usage. At the end of each section I discuss my opinion on the usage of the method. Finally, after listing all of the available different method and tools I will give my verdict on what will be used.

2.2 Matthew Huang

In this team, I'm responsible for the non photorealistic shaders, the silhouettes, and the runtime performance of the modified simulation. For the shaders, I must make sure that they improve the geometric shape of each object in the simulation. For the silhouettes, I must make sure that they clearly show where an object's borders are so that user can differentiate shapes from each other. For the runtime performance, I must make sure that our implementation does not negatively affect the frames per second that the current simulation is running at. To ensure this, I have researched various debuggers that will help us see what our program is doing during runtime.

2.3 Daniel Goh

My role within Shady Robots is to measure the improvement that is implemented into the simulation. I will also be running the selected inspection method to determine if the project has reached the end goal. After running the usability inspection, I will be compiling, visualizing and analyze the data set.

Usability inspection methods will be a standard of metrics to determine if the project has reached the end goal. The goal within this document is to compare and contrast the different available usability inspection methods, data collection tools, and analytical and visualization tools. After reviewing the various tools, a final verdict of the review is presented at the end of each section.

3 Identification of Rendering loop, Methods of code maintainability, and Shadows

By Justin Bibler

3.1 Identification of Rendering loop

Our project requires our group to make new implementations into the existing OpenRave code. However, the location of the rendering functions within OpenRave are not entirely know to our client. Therefore, the responsibility of locating these functions has been given to our group. This responsibility is very important to the entirety of the project. Most of our new implementations into the code will be reliant on the use of shaders; if the location of the rendering functions are unknown to us it would be extremely difficult for us to make good shader implementations. Thus, to identify the location of the rendering functions the three most fitting methods would be: Reading Static Code, debugging, external references, and documentation.

3.1.1 Reading Static Code

This is a very basic method in which a programmer simply reads over and reviews static code. In this case the OpenRave source code would be read over to identify how the rendering function is working.

Benefits of Reading Static Code:

- The entire system can be understood at a very low level.
- Locations of specific functions within the code can be identified.
- Exact implementation of functions can be understood.

Negatives of Reading Static Code:

- This method can become very time consuming depending on the size and complexity of the code.
- Code may difficult to read depending on how it is implemented.
- Inefficient, a programmer will most likely have to flip around multiple source files to figure out how things are working together.

3.1.1.1 Discussion: Reading static code might be too simple of a method to conduct by itself. It would definitely help me get a good understanding of how the code is function. However, as mentioned in the negatives of this method it may be too time consuming. The source code for OpenRave is quite large, and I have minimal experience working on a project this large scale so it would be difficult for me to jump right in and simply read code. Perhaps if this method were to be paired with another it could be useful.

3.1.2 Debugging

Debugging is a process where a programmer will use tools, and programming methods to locate errors in code. Many different forms and tools exist to conduct debugging.[1] Thus, I will narrow the scope down to one specific form a debugging: breakpoints, and step through. Breakpoints stop an executable and breaks into the debugger. This allows a programmer to analyze the code and execute debugger commands.[1]

Benefits of Debugging:

- Lots of available tools.
- Lots of available methods.

- Powerful control over code.
- Executable within most modern IDEs.

Negatives of Debugging

- Difficult to conduct on large projects.
- Stepping through code can become time consuming.

3.1.2.1 Discussion: Debugging is a good method that would help me accomplish my goal. If I were to use a method such as breakpoints and step through code at certain points, it could become a very helpful resource to use. However, due to the size of the program I am not exactly sure how easy it will be to debug.

3.1.3 External References, and Documentation

Much like reading static code this is a very simple and obvious method that can be used to locate the rendering function. This method is where a programmer will read through the available documentation to understand how the system works. In this method a programmer will also contact other programmers who are working on the code currently, or have worked on the code in the past. With the intention of receiving guidance on how the system works.

Benefits of External References, and Documentation:

- Easy to get answers to specific questions.
- Documentation gives a good understanding of the entirety of a system.

Negatives of External References, and Documentation:

- There may be no documentation.
- External sources may not exist.
- Past programmers may be unresponsive.

3.1.3.1 Discussion: It's quite obvious that this is one of the best methods. Because I am attempting to find a specific function within the code, referring to the documentation and external sources is a good choice. Along with this, the GitHub that is being used for OpenRave has a couple of active users so it would be relatively easy to contact another programmer who is more knowledgeable to assist me.

3.1.4 Verdict

My verdict on methods for identifying the rendering function is that all of the proposed methods are useful. I will most likely begin my search in the documentation then use the knowledge I gained from the documents to traverse through the source code. This is because as helpful as the documentation will be, it's required of me to read the actual code so I will have an even better understanding of it. If I am unable to locate the rendering function through documentation and by reading the code. I will reach out to other programmers within the GitHub group for assistance. As a last result, I will be using debugging techniques to traverse through the code in attempts to locate the function.

3.2 Methods of code maintainability

The goal of creating maintainable code is to increase cost effectiveness. This means that whenever a new change needs to be made to the system, due to how the code is a designed and constructed this change will take a minimal amount of time.[2] Maintainability is important to our code, because our client wishes to have the ability to easily change and adjust our implementation of shaders into the system. Thus our implementations into the code should use methods that allow for an increase in the ability to quickly understand and modify code. These proposed methods are: Version Control, Refactoring, Documentation, and Commenting.

3.2.1 Version Control

Version control has become increasingly easy to maintain with the popularity of version control tools such as GitHub. Using version control allows for users to make copies of repositories to either use for personnel use or make altercations. This practices gives protection to the code, such as if programmer losses or makes a devastating alteration to the code. That programmer doesn't have to worry much, because they know they can pull unmodified working code from the repository.[3]

Benefits of using Version Control:

- Source code is easily accessible.
- General tests can be defined to update stable versions of code in a repository.
- Code is safe to accidental deletion, or introductions of major bugs.

Negatives of using Version Control:

- There aren't really any negatives

3.2.1.1 Discussion: Using version control is a obvious powerful tool. It essentially protects code from being lost or broken. If a programmer accidentally introduces a large bug into the system, then with version control it is very easy to step back and pull older versions of the code. In the case that a programmer accidentally deletes their code, it's easy for them to go back and simply pull it from the repository again. Using version control most assuredly would help increase code maintainability.

3.2.2 Refactoring

Refactoring is a process in which code is altered in a manner such that it does not change how the external behavior of the system works.[4] In the case of this project, refactoring would be used after our group has created a working implementation of our clients request. We would go back into our code, and adjust our code so that it might be smaller, more readable, and in most cases reusable. Such as refactoring repeated code into a singular function that can be used in multiple locations of the code.

Benefits of using Refactoring:

- Easy to do.
- Cuts down code size.
- Reusable functions can be implemented and used in different locations of code.
- Maximizes runtime.

Negatives of using Refactoring:

- After a lot of refactoring code can become less readable.

3.2.2.1 Discussion: Refactoring is a good method to maximize code, especially when refactoring repeated code into new functions. If I were to refactor our implementations into the code with the client's desires in mind. I could most likely make it simple for them to adjust, and reuse functions throughout the program if they desire to change it.

3.2.3 Documentation, and Commenting

Documentation and Commenting allows new programmers to quickly get a grasp on the code. In the case of comments, a programmer is able to quickly understand intentions behind functions and certain operations within the code to get a better understanding of how the system works. And documentation of the code allows a programmer to have references to many questions related to the code.

Benefits of Documentation, and Commenting:

- New programmers have references to the what, where, and why of code.
- Good commenting allows a programmer to get a better understanding of the code while directly reading the code.
- Programmers have references of how to use functions, either in the comments of a function or in the documentation.

Negatives of Documentation, and Commenting:

- Too much, and useless comments can make code look clustered and sloppy.
- Writing documents can become time consuming depending on the size of system.

3.2.3.1 Discussion: Early in the document I talked about how I would be using documentation to help find the rendering functions within the OpenRave code. Thus, I personally find using documentation very helpful. I personally would prefer to focus more heavily on documenting our additions outside of the code rather then simply using comments. This is because documents are simply and easier reference.

3.2.4 Verdict

Each of these methods has it's usefulness that I will utilize to keep the system maintainable. Version control allows our group to make alterations to the code without fear of losing stable versions. This also allows for our client to come in and make their own alterations later in the future after the project is finished. In regards to documentation, and commenting. I will be recording all the changes and additions we make to the source code. This allows our group, and our client to have easy references to usable functions within the system. Finally, I will be using refactoring when I see it fit to be used. Our group is not expecting to make such massive alterations into the existing code that refactoring would have a major impact. I also do not intend to refactor the currently written OpenRave code.

3.3 Shadows

Shadows are used to help understand where in an object is within a 3d simulation. However, the current graphics used in OpenRave do not contain shadows. So our client has requested that we implement shadows into the current system. My proposed methods are: Light mapping, Shadow volumes, and Shadow mapping.

3.3.1 Light mapping

A light map is simply a texture that contains information about the lighting. In most cases a light map is rendered before the program is executed to speed up the programs runtime. Essentially, this light map is blended with textures that are intended to be lit to give off an illusion of lighting, and shadows.[5]

Benefits of using Light mapping:

- Easy to implement.
- Can be pre-rendered to increase runtime

Negatives of using Light mapping:

- Lighting is not dynamic.

3.3.1.1 Discussion: Light mapping seems like a good choice for doing lighting and shading for static objects in OpenRave. However, because the simulation contains moving 3d objects it will most likely need more dynamic shadows.

3.3.2 Shadow Volumes

This form of shadowing takes advantage of the stencil buffer in OpenGL. Essentially, a shadow volume is a volume in space that contains every point in which an object could be shaded. Objects that collide with the shadow volume should generate shadowing on their surface according to the shadow volume. This is done by rendering objects shadowed pixels with different stencil values compared to their non-shadow pixels. [6]

Benefits of Shadow Volumes:

- Dynamic.
- Relatively lightweight.

Negatives of Shadow Volumes:

- Doesn't generate the best looking shadows.

3.3.2.1 Discussion: This for of creating shadows seems relatively simple for my skill level. Along with this it's dynamic, and lightweight which are both good qualities our group is looking for in our shadow implementations.

3.3.3 Shadow maps

Shadow mapping is kind of like light mapping. Shadow mapping works by rendering a scene twice. In the first render, only the depth of each fragment is computed. In the second render, the scene is rendered normally. However, the information computed in the first render is used to calculate whether or not an object has fragments in front of it relative to the light source. If a fragment has a fragment from the first render blocking it, a shadow should be placed displayed. [7]

Benefits of Shadow Mapping:

- Shadows are dynamic.
- Shadows will look good because they are fragment based.

Negatives of Shadow Mapping:

- Requires scene to be rendered multiple times.
- Worse runtime.

3.3.3.1 Discussion: Shadow mapping seems great, it also seems like it would create the best looking shadows for the simulation. However, shadow mapping would most likely greatly increase the runtime of the program and be detrimental to the FPS.

3.3.4 Verdict

I think that the shadow volume implementation would be the best to implement into the system. This is because it offer acceptable looking dynamic shadows, without sacrificing too much runtime and FPS. It would be great if we could implement the shadow mapping, but the computers that the simulation will be frequently ran on does not have the best GPU. Making shadow mapping a bad choice.

4 Non photorealistic shading, silhouettes, and optimization - run time analysis

By Matthew Huang

4.1 Non photorealistic shading

4.1.1 Gooch Shading (Cool to Warm Shading)

Gooch shading, developed by Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen, is a non photorealistic shading model for rendering objects in a technical illustration style. [8] In Gooch Shading, the following characteristics are prominent:

- Edge lines are drawn with black curves.
- 3D objects are shaded using color intensities that are far from black or white. Instead, the warmth or coolness of a color indicates its relationship to the surface normal.
- A single light source provides white light for the object.
- Shadowing is not shown.
- Metal objects are shaded as if very anisotropic

The colors used in Gooch shading have distinct tones and temperatures. The color tones are created by adding either gray to the color or by adding the color's complement to the color. The temperature of the color is defined to be warm (red, orange, yellow), cool (blue, violet, green), or temperate (red-violet, yellow-green). Shading with these colors will allow for tone scaled object color along with a warm-to-cool undertone. An example of a Gooch Shaded Object is provided in Figure 1 below. [9]

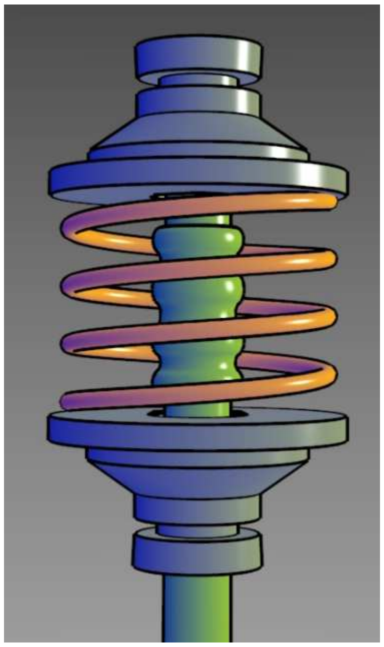

Fig. 1.
Gooch shaded model showing the implementation of black edge lines combined with curve highlights and warm to cool hue shifts.

4.1.2 Cel Shading

A common type of non photorealistic shading that is primarily used to give 3D objects in computer graphics a flat, 2D look. It accomplishes this by using fewer shading colors as opposed to the traditional shading gradient Additionally, in cel shading, color hues are distinct and do not blend. [10] This is implemented by defining constant light intensities at certain regions and is responsible for the main visual difference between realistic shading and cel shading. The lighting in cel shading is done per pixel and can be implemented in either the vertex or fragment shader. [11] Figure 2 below illustrates the differences between cel and realistic shading.

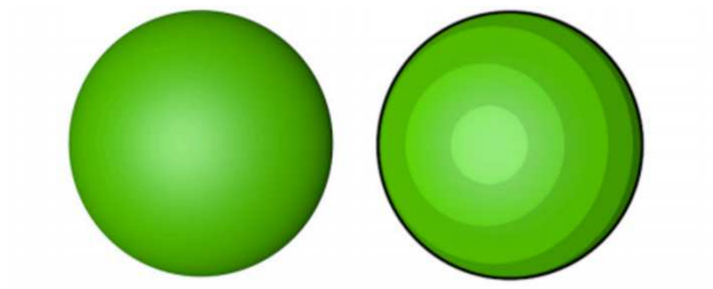

Fig. 2.
On the left is an object shaded using realistic shading. Observe that there is a wide range of shading colors and that colors blend together. On the right is an object shaded using cel shading. Notice that color hues are few in number and are distinct from each other (I.E. do not blend).

4.1.3 Hatching

A non photorealistic shading technique that does shading by drawing multiple hatches (parallel lines) to highlight darker parts of an object. The general rule of thumb is the darker the area, the more hatches are drawn there. The length, width, and distance between hatches are defined by the developer. If the hatches are not parallel and instead cross, then it is called cross-hatching. [12] Drawing hatches with different angles can help with emphasizing curves in an object. To implement hatching, textures of different levels of hatching are first created. After that, these texture can then be sent to the fragment shader and be assigned based on the lighting of the object. Figure 3 what a 3D cat would look like if it was shaded using hatching.

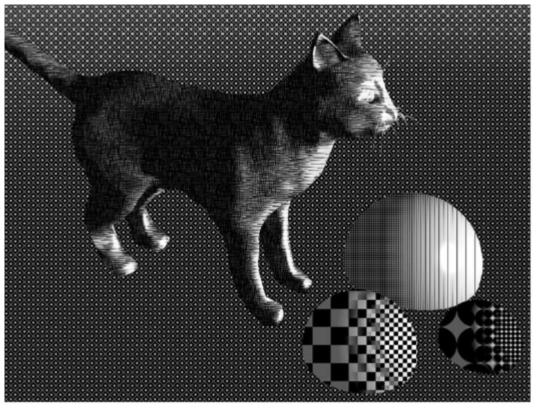

Fig. 3.
3D cat drawn using simple hatching. Notice that the number of hatches drawn is used to highlight the curves of the cat and is based on the light position.

4.1.4 Goals

Our client wants the shading to primarily highlight object shape. This entails focus on emphasizing the curves of objects, They also want all parts of any 3D rendered object to be visible if it is within view. This would mean that no piece of an object should be completely black as this would possible confuse the object's shape.

4.1.5 Criteria

These technologies will be primarily judged based on how well they highlight object shape and visibility. Additionally, how well each are documented will be considered. Speed, resource cost, and implementation flexibility will also be factored in but at a lower priority. This is because the speed and cost of shading implementations are more dependent on the specific implementation rather than the type of shading.

4.1.6 Table comparing non photorealistic shading technologies

Technology	**Highlight Shape**	**Object Visibility**	**Documentation**	**Speed & Cost**	**Flexibility**
Gooch Shading	Shape highlighted using warm to cool colors tones.	No parts of the 3D object is completely blacked out.	Well documented. Original paper detailing Gooch Shading available online.	Uses equations to calculate tones of a color. Will mostly depend on specific implementation.	Can be implemented with fragment or vertex shader.
Cel Shading	Shape highlighted using few shading colors. Color hues do not blend which creates a 2D look for the shape.	Object areas have constant light intensities so they aren't typically completely black.	Widely used and well documented. Many pieces of sample code exist online.	Fast and efficient due to fewer shading colors and constant light intensities.	Lighting can be done using vertex or fragment shader.
Hatching	Shape highlighted by using textures with different number of hatches. The darker the area, the more hatches.	Darker areas of an object have more hatches drawn there meaning parts of it could be completely black.	Not as common as cel or Gooch shading but documentation still exists.	Depends on implementation. Hatches typically drawn using textures and lighting equations.	Must use both vertex and fragment shader.

4.1.7 Discussion

All three shaders do well at highlighting object shape. For our purposes, however, Gooch shading and Hatching do a better job since they allow for the object to retain its 3D shape. Cel shading makes objects look 2D which, for us, is bad. Gooch and cel shading both do well at keeping full object visibility. Hatching has the possibility of completely blackening certain parts of objects which means visibility may be a problem. Cel shading is the most well documented since it is the most common type among the 3. Gooch is not far behind with its documentation widely available online. Hatching is the least well documented, but resources still exist online. Cel is likely the fastest and most efficient among the 3 shaders but the others aren't far behind. Gooch and cel shading can be implemented using either the vertex or fragment shader while Hatching must use both. This is not much of a problem, however, since our implementation will likely use both anyways.

4.1.8 Choice

For our purposes, Gooch shading is the best option. It does a great job at highlighting object shape while also guaranteeing full object visibility. Additionally, good documentation exists online for it so getting started won't be an issue. The other two types of shading weren't selected because they didn't highlight both object shape and visibility. Cel doesn't do well at highlighting 3D curves and Hatching does a poor job at showing object visibility.

4.2 Silhouettes

4.2.1 Stencil Buffer Technique

Technique for rendering a silhouette edge around a 3D object using the stencil buffer. This technique can draw the silhouette of an object before or after the object is drawn. In this algorithm, the object is drawn 4 times; each time displaced by 1 pixel in the x or y direction. [13] This offset is done in windows coordinates and can be implemented by changing the viewport coordinates each time. The algorithm, in detail, is provided below [6]:

1) If you want to see the object itself, render it in the usual way.
2) Clear the stencil buffer to zero.
3) Disable writing to the color and depth buffers
4) Set the stencil function to always pass, set the stencil operation to increment
5) Translate the object by +1 pixel in y, using glViewport()
6) Render the object
7) Translate the object by -2 pixels in y, using glViewport()
8) Render the object
9) Translate by +1 pixel x and +1 pixel in y
10) Render
11) Translate by -2 pixel in x
12) Render
13) Translate by +1 pixel in x. You should be back to the original position.
14) Turn on the color and depth buffer
15) Set the stencil function to pass if the stencil value is 2 or 3. Since the possible values range from 0 to 4, the stencil function can pass if stencil bit 1 is set (counting from 0).
16) Rendering any primitive that covers the object will draw only the pixels of the silhouette. For a solid color silhouette, render a polygon of the color desired over the object [6].

4.2.2 Silhouettes using Polygonal offsets

A common technique to draw silhouettes often used in cel shading (note cel shading itself not necessary to use this technique). To implement these silhouettes, first render an enlarged version of the object with a constant color (black). [14] After that, render the object again but with its normal size and a different color over the top of the enlarged version. This process is shown visually in Figure 4 below:

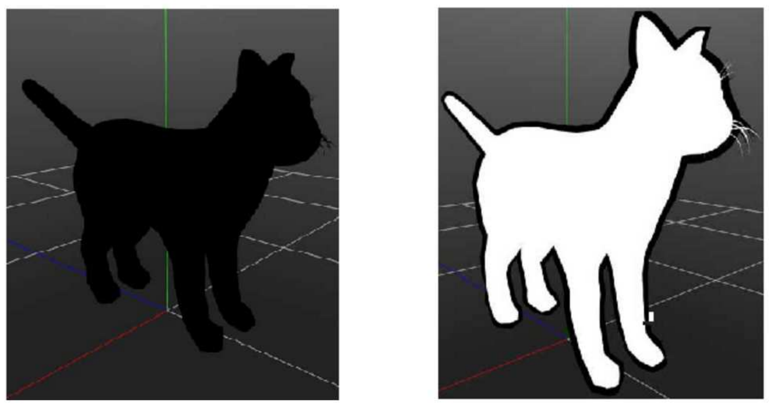

Fig. 4.
On the left is the scene after the first rendering. The scene displays an enlarged version of the object drawn only in black. On the right is the scene after the second rendering. The second rendering draws the object (in white) with its normal size over the enlarged version creating the silhouette.

4.2.3 Edge Detection Silhouettes

The basic idea of this technique is find all the front facing edges of the object and to place them in a list. There are multiple algorithms out there to do this such as the Frei-Chen edge detection algorithm. [15] After the edge list is created, draw along the list in black (or any other desired color) and the silhouette will be drawn. [16] The main part of this technique is the edge detection. Although multiple algorithms exist, some are exceedingly slow and others are difficult to implement.

4.2.4 Goals

Silhouettes should define an object's borders. In other words, they define where an object begins and ends. Their primary purpose is to help differentiate objects in the scene from each other. This will assist in defining an object's position within the scene.

4.2.5 Criteria

These silhouettes will be judged on smoothness, ease of implementation, efficiency, flexibility, and how well documented it is. Smoothness is self explanatory; silhouettes should not appear jagged when zoomed in on. Ease of implementation varies greatly between different silhouette techniques; implementation time should only be increased if the benefit is significant. Efficiency will be determined by how the silhouette is created. Some techniques create it by drawing the object twice, others create it by traversing the object's vertices. Flexibility is measured by how the silhouette behaves under different situations. For example, does it work when objects overlap?

4.2.6 Table comparing silhouettes

Technology	Smoothness	Implementation Difficulty	Efficiency	Flexibility	Documentation
Stencil Buffer Technique	Based on how the original object was drawn since its vertices to create the silhouette.	Involves messing with the Stencil Buffer. Algorithm to implement these silhouettes provided online.	Draws the object 4 times.	Handles overlapping objects well. It can display just silhouette edges or edges and borders.	Well documented on OpenGL websites and other sources.
Polygonal offset	Based on how the original object was drawn since its vertices to create the silhouette.	Easy. Simply draw the object twice but with an offset on one of them.	Draws the at least twice.	Struggles when objects overlap since the silhouette is a background object.	Well document since it is easy to implement and commonly used in cel shading.
Edge Detection	Based on how the original object was drawn since its vertices to create the silhouette.	Varies depending on the edge detection algorithm used.	Traverse through every vertex of the object. Efficiency also dependent on edge detection algorithm.	Handles overlapping objects well since separate objects have their own edge lists.	Not well documented. Resources mostly come from forum posts and stackoverflow questions.

4.2.7 Discussion

All three silhouette drawing methods have the ability to create smooth silhouettes since they are all based on how the object was drawn. The polygonal offset method is by far the easiest to implement since it is just drawing the object twice but with an offset. The stencil buffer technique has a detailed and clear algorithm for its implementation so it's simple to implement too. The edge detection method is less clear due to lack of solid documentation. The polygonal offset and edge detection methods are similar in efficiency while the stencil buffer technique is the most inefficient (since it has to draw the object four times). The only method that struggles with overlapping objects is the polygonal offset method. This is significant since overlapping objects are common in 3D scenes. All three methods have resources online, but the edge detection method is the least well documented.

4.2.8 Choice

The stencil buffer technique is the best method for our purposes. It is well documented so implementing it will be straightforward. Additionally, it is flexible in what it displays (silhouette edges, borders, etc.) and when it is applied (after object is drawn or before). The only draw back is its efficiency since it has to draw the object four times. There are, however, modifications to the algorithm that can reduce the number of times it has to draw to two. [17] The edge detection case was not chosen because it is not well documented. The polygonal offset choice, while easy to implement and well documented, was not chosen because it lacked flexibility.

4.3 Optimization - Run time analysis

4.3.1 CodeXL

A powerful OpenGL and OpenGL ES debugger that was once called gDEBugger before it was bought out by AMD. It traces application activity on the OpenGL API which will assist with finding bugs and boosting performance. CodeXL can be applied to a variety of applications and works on Windows and Linux platforms. Using it is as simple as just running the application in the debugger. Additionally, CodeXL has a multitude of features that will assist in the debugging process such as static shaders analysis, GPU profiling, and power profiling. [18] CodeXL is available on github and is free to use.

4.3.2 GLSL-Debugger

An open source fork of the glslDevil project. GLSL-Debugger is a tool for debugging OpenGL programs. Key features include tracing OpenGL programs, visual GLSL step-by-step debugging, support for Linux and Windows, and record and playback OpenGL traces [11]. The source code is available online for free but requires the user to compile it. Instructions for how to do this is shown on their website.

4.3.3 GLIntercept

GLIntercept is a OpenGL function call interceptor available for the windows platform. [19] To use it, user must copy certain install files (opengl32.dll and a gliConfig.ini) to the executable folder and then edit the gliConfig.ini file to their liking. Additionally, GLIntercept has a variety of feature including function call logging, saving and tracking of shaders, saving and tracking of display lists, function stats, and more. GLIntercept is available on github for free. Importantly note that this debugger may not work on advanced features in OpenGL 3.0+ due to it being developed using the OpenGL 1.0-2.1 debugger [12].

4.3.4 Goals

The goal of this runtime analyzer is to improve application performance by tracing where resources are being used in the program. In general, our modifications should not slow down the current implementation as that would hurt the smoothness of the simulation. This analyzer will help track the changes our modifications are making to the program.

4.3.5 Criteria

The debuggers will be judged based on relevant features, usability, and documentation. Relevant features include function call tracing, data structure tracking, available platforms, etc. Usability is measured by how easy it is to install and use. Documentation is measured by how many resources on the debugger are available online. The first two factors will be of higher priority than the third because not having certain features is much more important than amount of documentation.

4.3.6 Table comparing run time analysis technologies

Technology	Program tracing and tracking	Other Relevant Features	Usability	Available Platforms	Documentation
CodeXL	Included	Static Shader Analysis. GPU and Power profiling.	Easy to use and download. Quick start guide available for download. Uses a GUI to display resource consumption.	Windows and Linux.	Well documented with a quick start guide available.
GLSLDebugger	Included	None	Source code available online. Must build and compile yourself. Uses a GUI to display program tracing.	Windows and Linux.	Not very well documented. Their website has enough documentation to get the debugger built.
GLIntercept	Included	Resource leak tracing, tracking of error states, function timer logs.	Source code available online. Must copy files over and edit certain files to debug. No GUI.	Windows.	Well documented. Github page has detailed list of features and short tutorial on how to use it.

4.3.7 Discussion

As far as features go, all of the debugger have the minimum necessary features to warrant use (I.E. program tracing). CodeXL has the most available features, but they may not all be useful to us. GLSL-Debugger has the fewest features while GLIntercept is somewhere in the middle. In the usability department, CodeXL ends up on top. It's not only easy to install, but it also comes with a useful GUI for debugging. GLSL-Debugger also comes with a GUI, but it is trickier to get working. GLIntercept has no GUI which hurts its overall usability. For available platforms, the only debugger that doesn't work on Linux is GLIntercept. This is significant as it factors into other parts of our project such as maintenance. When it comes to documentation, CodeXL once again comes out on top. Its provided quick start guide shows useful examples on how to install and use the application. GLIntercept has good documentation, but no examples while GLSL-Debugger only provides screenshot for how to use it.

4.3.8 Choice

CodeXL is the best debugger for our purposes. It provides many useful features for us to use while also being user friendly. Additionally, good documentation for it is available online; the aforementioned quick start guide is especially useful. The GLIntercept debugger was not chosen because it was limited on platform. The project must work on Linux so having a debugger that doesn't work on Linux would be a hassle. GLSL-Debugger was not chosen because it was too simple (in terms of features) and not very well documented.

5 Usability inspection methods, how to carry out the selected inspection method, and the tools to analyze the gathered data

By Daniel Goh

5.1 Usability Inspection Methods

Usability inspection will be carried out to measure the improvement between the starting and the end phase of the project. Usability.gov lists multiple usability inspection methods that can be used to determine the usability of an interface. The three main inspection methods that suits this project scope include: Individual Interviews, Online Surveys, and System Usability Scale (SUS). [20] This section will be used to review various inspection methods and select one that will best suit this project.

5.1.1 Individual Interviews [20]

An individual interview is an inspection method in which a interviewer talks to the participant for 30 minutes to an hour. During this interview, the interviewer is tasked to gain a deeper understanding of the participants. This includes taking note of the participant's attitudes, beliefs, desires, and experiences.

Guidelines to conduct individual interviews as defined by usability.gov are as follows:

- Define the aim of the study and select relevant participants.
- Prepare interview protocol which includes questions and probes to use during the interview.
- Create a comfortable interview situation by asking questions in a neutral manner and be attentive for probe queues.
- Get permission to tape interview session, and have one or more note takers during the interview.

Benefits of Individual Interviews:

- Researchers will be able to gain a better understanding of individual participants.
- Researchers will be able to observe the participant's body language and facial emotions during the interview.
- Researchers will be able to receive additional insights that might not have occurred to the interviewer.

5.1.2 Online Surveys [20]

An online survey is a form of feedback that is done over the internet. This inspection method is structured as a questionnaire, and is published online to gather feedback from a broad audience. The data is then stored within a database, and a survey tool is used to analyze the data.

Guidelines to conduct individual interviews as defined by usability.gov are as follows:

- Identify purpose of the survey.
- Determine and select target audience that will best suit the project.
- Identify methods to collect data and limitations to data collection.
- Create brief surveys.
- Provide estimated time to completion up front, and show progress of survey to participants.
- Mix in open-ended questions with closed questions.
- Allow participants to choose to answer in-depth questions through a follow-up session.

Benefits of Online Surveys:

- Researchers will be able to reach to a broader audience.
- Researchers will be able to learn who the users are.
- Users will be more willing to participate as it would not take up too much time.
- A wide variety of survey tools are available online.
- Cost efficient for researchers.

5.1.3 The System Usability Scale [20]

The System Usability Scale (SUS) is a tool created by John Brooke in 1986. The SUS method is structured as a ten item questionnaire that requires users to rate individual items with five response options; from Strongly agree to Strongly disagree.

Guidelines to conduct System Usability Scale as defined by usability.gov and UsabilityNet.org are as follows:

- Participants should not spend time thinking about items for a long time, instead they should record their immediate response. [21]
- The questionnaire are defined and covers the need for support, training, and complexity of system usability.
- Data gathered through this method needs to be ”normalized” to produce a percentile ranking.

Benefits of The System Usability Scale:

- Easy to scale to administer to participants.
- Study can yield reliable results, even on small sample sizes
- The System Usability Scale is an industry standard, and is credible in differentiating usable systems from unusable ones.

5.1.4 Verdict

After reviewing the available usability inspection methods, the **online survey method** would be a better fit for our project. By utilizing online surveys, the project would be able to collect more insight from a larger and broader audience. Individual interviews would not be appropriate as our project does not require an emotional understanding of the participant’s thoughts and body language. The System Usability Scale employs questions that dives too deep into the system, and would introduce scope creep in comparison to the requirements provided by our customer. Thus, the (BOLD)online survey method will be utilized as a usability inspection method to measure the success of the project.

5.2 How to carry out selected inspection method: Online Surveys

There are many survey tools available online. This section reviews the various survey tools that are free to use. The reviews are done after using the survey tools first-handedly.

5.2.1 Google Forms

This section covers the pros and cons of Google's survey tool, Google Forms.

Pros of Google Forms for our project purpose:

- Allows unlimited questions in a form
- Collected data are visualized as pie charts
- Collected data can be exported into Google Spreadsheets (Google's equivalent of Microsoft Excel)
- Collected data can be exported as a .CSV file to be use with other data analysis tools
- Allows images and videos to be added into each survey questions without the need to host them externally
- Allows unlimited respondents to take the survey

Cons of Google Forms for our project purpose:

- As the collected data are stored on Google's remote servers, we are giving Google (and their affiliated partners) "a worldwide license to use, host. store, reproduce, create derivative works, communicate, publish, publicly perform, publicly display and distribute" the content [22]

5.2.2 Survey Monkey

This section covers the pros and cons of Survey Monkey.

Pros of Survey Monkey for our project purpose:

- Well known as an online survey tool
- Contains a variety of different questions types
- Has a dedicated "SurveyMonkey and IRB Guidelines" under the policies section
- Provides assistance with IRB approvals by providing "evidence permission to use the SurveyMonkey platform to conduct research" [23]

Cons of Survey Monkey for our project purpose:

- Only allows 10 questions
- Only allows 100 respondents
- No data export capability
- No data reporting capability

5.2.3 Typeform

This section covers the pros and cons of Typeform.

Pros of Typeform for our project purpose:

- Allows unlimited questions
- Allows unlimited respondents
- Supports data export

- Respondent's input can be piped into other questions (User inputs their name for the first question, and the second question can access their name and call them by name in upcoming questions)
- Interface is modern and beautifully designed

Cons of Typeform for our project purpose:

- Uses keyboard as main input for survey questions

5.2.4 Verdict

After reviewing various survey tools, **Google Forms** comes in as the best choice that fits the selected usability inspection method. Our survey will not collect sensitive information (e.g. social security numbers, credit card numbers etc.), thus it is safe if Google make the data accessible to the public. The purpose of the survey (to compare the simulation visuals prior to enhancement and after enhancement) is unrelated to the original research, and does not need to be kept confidential.

5.3 Data analysis tools to analyze and visualize collected data

Data analysis and visualization tools are used in enterprise situations to allow massive data to be processed into meaningful data. With the inspection method and survey tools selected, a suitable data analysis and visualization tool will be needed to process the raw data. As the selected survey tool allows CSV (comma separated value files) to be exported, the review will only include tools that can process CSV files and that are free of charge. The reviews for data analysis and visualization tools are done after using the applications first-handedly.

5.3.1 Google Forms, Google Sheets

Google has a wide range of applications in their portfolio, and the survey tool selected, Google Forms come with an integrated pie chart visualizer. However, users will be able to transfer the survey data into Google Sheets for a more complete data analysis environment.

Pros of using Google's visualization for our project purpose:

- Easily transferable across Google platforms
- Easily shared across multiple collaborators
- Simple, light weight and easy to use
- Work can be done in a web browser
- Keeps records of previous spreadsheet versions for reverting purposes

Cons of using Google's visualization for our project purpose:

- Limited chart types
- Google (and their affiliated partners) is given "a worldwide license to use, host. store, reproduce, create derivative works, communicate, publish, publicly perform, publicly display and distribute" the content [22]

5.3.2 Microsoft Excel

Microsoft Excel is an application that requires a subscription fee (as a part of Office 365) to access. However, as an Oregon State University student, we have access to use the software with a student license. Microsoft Excel is the leader of all spreadsheet applications.

Pros of using Microsoft Excel for our project purpose:

- Easy to create charts from CSV files
- Contains many chart customization options
- Allows the user to manipulate and show the data in many different ways through PivotCharts

Cons of using Microsoft Excel for our project purpose:

- History keeping of excel files are not reliable
- PivotCharts requires user overhead to correctly manipulate

5.3.3 d3.js

D3.js is a JavaScript library that is used to manipulate documents based on the data that is fed into it. D3 emphasizes it's output on web standards, and only requires an internet browser to show the visualization of data.

Pros of using D3.js for our project purpose:

- Contains many beautifully designed templates that can be used
- Code base is open sourced
- Can display dynamic visualization on web pages
- Visualization updates on the fly as data gets updated

Cons of using D3.js for our project purpose:

- Not a thorough data analysis tool, but mainly a data visualization tool
- Has a steep learning curve to get things running

5.3.4 Verdict

After reviewing the different data analysis and visualization tools, **Google Sheets** will be used for data analysis and visualization purposes. As many of the project documents reside in a collaborative Google Drive, migrating the data analysis portion out from a well constructed ecosystem is not necessary. With Google Sheets, the spreadsheet can be worked on collaboratively and can be shared with the customer for better transparency on the ongoing data analysis.

5.4 Conclusion

In conclusion, debugging techniques will be used to identify the render loop. Refactoring code will assist code maintainability. The shadow volume implementation will be used to implement shadows. Gooch shading will be implemented for non photorealistic shading. The stencil buffer technique will be used to implement silhouettes. CodeXL will be used for run time analysis. Online surveys will be used to carry out usability inspection. Google Forms will be used to carry out the online survey. And Google Sheets will be used to do the data analysis and visualization.

REFERENCES

[1] Microsoft, “Standard debugging techniques.” [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/hardware/hh439390(v=vs.85).a

[2] F. D. Markus Pizka, “How to effectively define and measure maintainability.” [Online]. Available: http://www.itestra.de/fileadmin/Redaktion/Documents/07_itestra_define_and_measure_maintainability.pdf

[3] git, “Getting started - about version control.” [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[4] c2, “What is refactoring.” [Online]. Available: http://wiki.c2.com/?WhatIsRefactoring

[5] A. Baylis, “Lightmapping tutorial.” [Online]. Available: http://www.alsprogrammingresource.com/lightmapping_tutorial.html

[6] J. Beam, “Tutorial - stenciled shadow voumes in opengl.” [Online]. Available: http://joshbeam.com/articles/stenciled_shadow_volumes_in_opengl/

[7] opengl tutorial, “Tutorial 16 : Shadow mapping.” [Online]. Available: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

[8] E. Wiki, “Gooch shading.” [Online]. Available: https://lva.cg.tuwien.ac.at/ecg/wiki/doku.php?id=students:gooch

[9] B. G. Amy Gooch, “A non-photorealistic lighting model for automatic technical illustration.” [Online]. Available: http://artis.imag.fr/~Cyril.Soler/DEA/NonPhotoRealisticRendering/Papers/p447-gooch.pdf

[10] R. R. Luque, “The cel shading technique.” [Online]. Available: https://raulreyesfinalproject.files.wordpress.com/2012/12/dissertation_cell-shading-raul_reyes_luque.pdf

[11] M. ten Bosch, “Non-photorealistic shading.” [Online]. Available: http://marctenbosch.com/npr_shading/

[12] I. Dykhta, “Hatching and gooch shading in glsl.” [Online]. Available: http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=27

[13] OpenGL.org, “Silhouette edges.” [Online]. Available: https://www.opengl.org/archives/resources/code/samples/advanced/advanced97/notes/node2

[14] I. Dykhta, “Non-photorealistic rendering (cel shading).” [Online]. Available: http://sunandblackcat.com/tipFullView.php?l=eng&topicid=15

[15] DanielRakos, “Frei-chen edge detector.” [Online]. Available: http://rastergrid.com/blog/2011/01/frei-chen-edge-detector/

[16] S. Craitoiu, “Toon shading effect and simple contour detection.” [Online]. Available: http://rastergrid.com/blog/2011/01/frei-chen-edge-detector/

[17] OpenGL.org, “Silhouette edges.” [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/user-research/index.html

[18] gpuopen.com, “Codexl.” [Online]. Available: http://gpuopen.com/compute-product/codexl/

[19] GLSLDebugger, “Glsldebugger.” [Online]. Available: http://glsl-debugger.github.io/

[20] usability.gov, “User research methods.” [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/user-research/index.html

[21] J. Brooke, “Sus - a quick and dirty usability scale.” [Online]. Available: http://www.usabilitynet.org/trump/documents/Suschapt.doc

[22] Google, “Google terms of service.” [Online]. Available: https://www.google.com/policies/terms/

[23] SurveyMonkey, “Surveymonkey and irb guidelines.” [Online]. Available: http://help.surveymonkey.com/articles/en_US/kb/How-does-SurveyMonkey-adhere-to-IRB-guidelines