Design Document 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

February 17, 2017

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 This simulation is using outdated libraries which result in outdated graphics. Design definitions outlined in this document will be used to accomplish our customer's request. The request being to update the simulation's graphics with warm cool shaders, shadows and silhouettes.

Keywords: OpenInventor, OpenGL, OpenRave, shaders, warm cool shaders, silhouettes, shadows, robotic simulation, geometry, visualization, render, vertex lines

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to elaborate on the design and logic of how we will be implementing our requirements.

1.2 Scope

The scope of this document is solely about how new features will be implemented and how those implementations will be tested.

1.3 Context

Currently, our client is using visualizations, of a robot hand grasping objects, to collect data online. These visualizations are created from a simulation program (OpenRave); the data collected is used to create a model of the human grasp. However, the current graphics in the simulation are outdated. It is hard to see and understand the shapes and contact points represented in the scene.
The context of this document is focused primarily on the visuals of the project.

1.4 Summary

In clauses 3 to 5, Matthew Huang outlines the design of Gooch shading, silhouettes, and run-time analysis with CodeXL respectively. In clauses 6 to 8, Justin Bibler outlines the design of shadow mapping, performance benchmarks using FRAPS [1], and methods of code maintainability. In clauses 9 and 10, Daniel Goh outlines the methods and guidelines to create the online survey, and methods to analyze and visualize the collected data.

REFERENCES

[1] Fraps, “Fraps.” [Online]. Available: http://www.fraps.com/
[2] mathsteacher.com, “Frequency and frequency tables.” [Online]. Available: http://www.mathsteacher.com.au/year8/ch17_stat/03_freq/freq.htm
[3] I. Dykhta, “Hatching and gooch shading in glsl.” [Online]. Available: http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=27
[4] S. . B. Cat, “Non-photorealitic rendering(cel shading).” [Online]. Available: http://sunandblackcat.com/tipFullView.php?l=eng&topicid=15
[5] OGLdev, “Stencil shadow volume.” [Online]. Available: http://ogldev.atspace.co.uk/www/tutorial40/tutorial40.html
[6] F. D. M. Pizka, “How to effectively define and measure maintainability.” [Online]. Available: http://www.itestra.de/fileadmin/Redaktion/Documents/07testradefineandmeasuremaintainability.pdf
[7] “Qualtrics.” [Online]. Available: https://www.qualtrics.com/
[8] S. Monkey, “Writing good survey questions.” [Online]. Available: https://www.surveymonkey.com/mp/writing-survey-questions/
[9] ——, “Making sense of the numbers: When to embrace relativity, and when to ignore it.” [Online]. Available: https://www.surveymonkey.com/blog/2012/06/28/making-sense-numbers-when-embrace-relativity-when-ignore-it/

2 GLOSSARY

FRAPS - FRAPS is a benchmarking, screen capture and screen recording utility for Windows.

FPS - Frames per second (FPS) is a unit that measures display device performance.

Frequency (statistics) - Frequency of a particular data value is the number of times the data value occurs. [2]

3 Requirement: Gooch Shading

By Matthew Huang

3.1 Design Concerns

The rendering of the scene is done by an API call from the qtcoinrave plugin (which OpenRave uses as a viewer). This means that we have no direct access to the actual rendering main loop. As a result, we'll have to use the qtcoin3D API to implement our changes. The OpenGL Shading Language (GLSL) will be used to implement Gooch Shading. Gooch Shading requires use of both the vertex buffer and fragment buffer. In its implementation, Gooch Shading makes use of a warm and cool color to highlight its objects. Gooch shaded objects should not have any ”blacked out” areas on them.

3.2 Design Stakeholders

Cindy Grimm, Justin Bibler, Matthew Huang, and Daniel Goh

3.3 Context Viewpoint

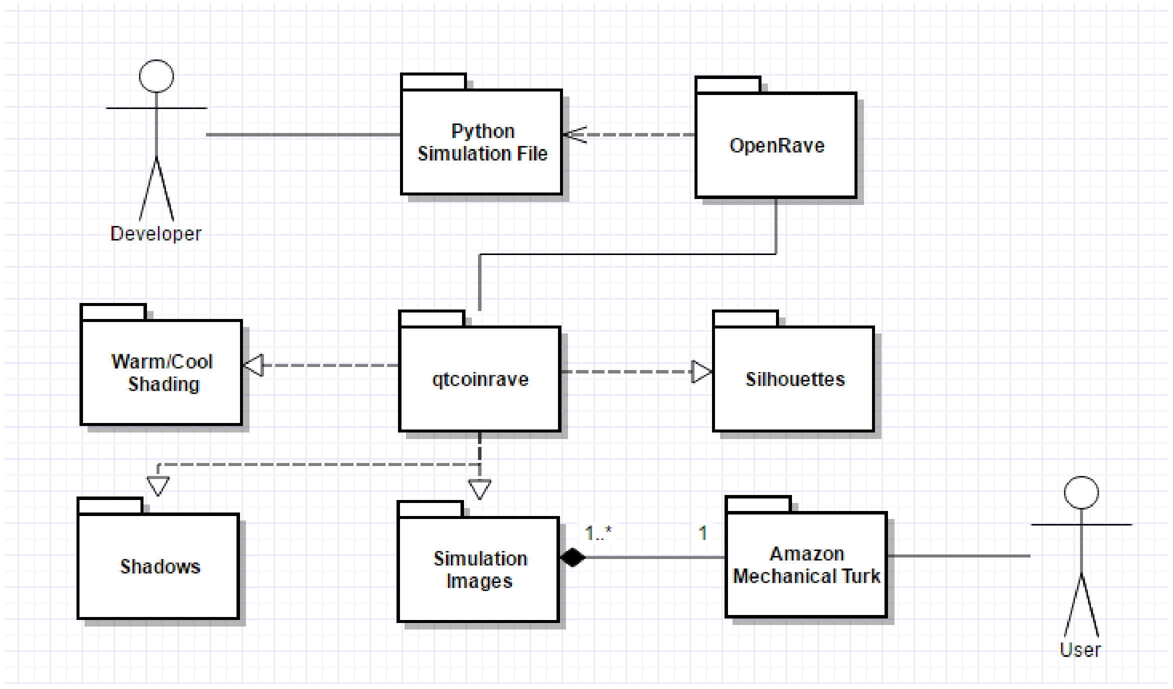


Fig. 1.
Context Viewpoint Diagram that shows relation of Gooch Shading to OpenRave

3.3.1 Design View

Gooch Shading will be implemented by passing vertex and fragment shaders to qtcoinviewer. A successful Gooch Shading implementation will use warm and cool coloring to help improve object geometry (which will be displayed by Qt). Improved object geometry means improved simulation images for users of the Amazon Mechanical Turk. The updated images will aid user confidence in how they answer the mechanical turk questions which will improve the data collected.

3.4 Compositional Viewpoint

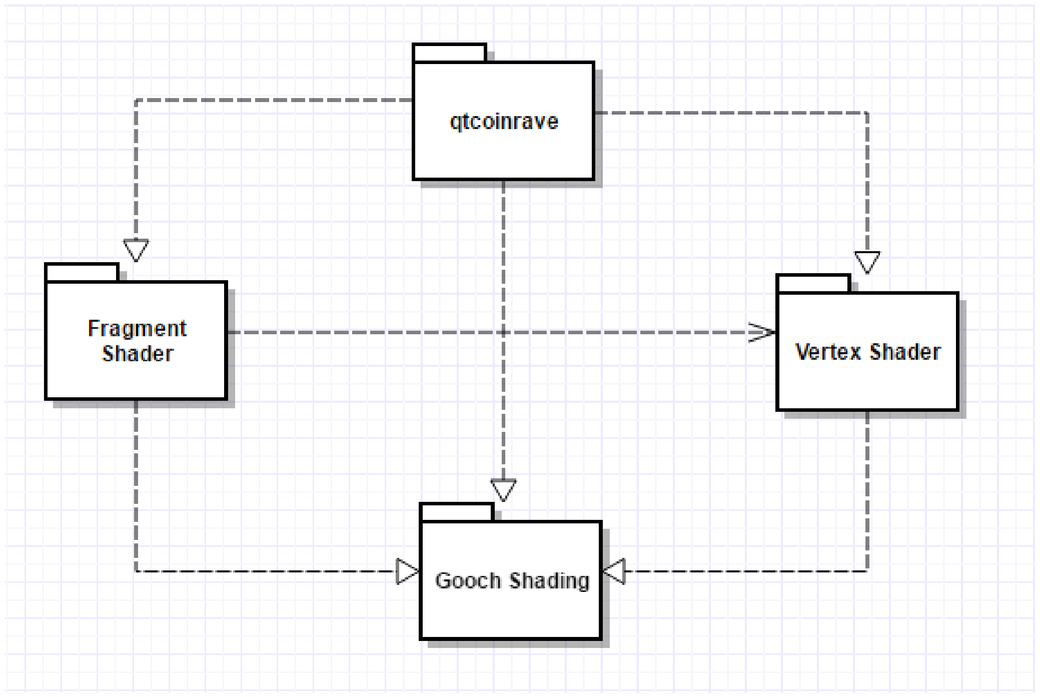


Fig. 2.
Compositional Viewpoint Diagram that shows the flow of Gooch Shading implementation to OpenRave

3.4.1 Design View

This implementation of Gooch Shading makes use of the following entities: the vertex shader, the fragment shader, and qtcoinviewer. The shaders will be given to qtcoinviewer using the SoShaderProgram widget provided in the qtcoin3D API. Both shaders will have all relevant variables (camera position, model coordinates, light position, colors, etc.) by default. The vertex shader calculates the light position and passes its variables to the fragment shader. The fragment shader is the entity that shades the 3D objects. In it, it calculates the diffuse lighting along with the augmented warm and cool colors that will shade the object. The augmented warm and cool colors are determined by the object's original color and the Gooch Shading weight. After that, linear interpolation is done on the two augmented colors to determine what the final color will be for that fragment. In GLSL, linear interpolation can be done using the mix() function. This shading will highlight the entire object leaving no completely ”blacked out” areas.

3.5 Logical Viewpoint

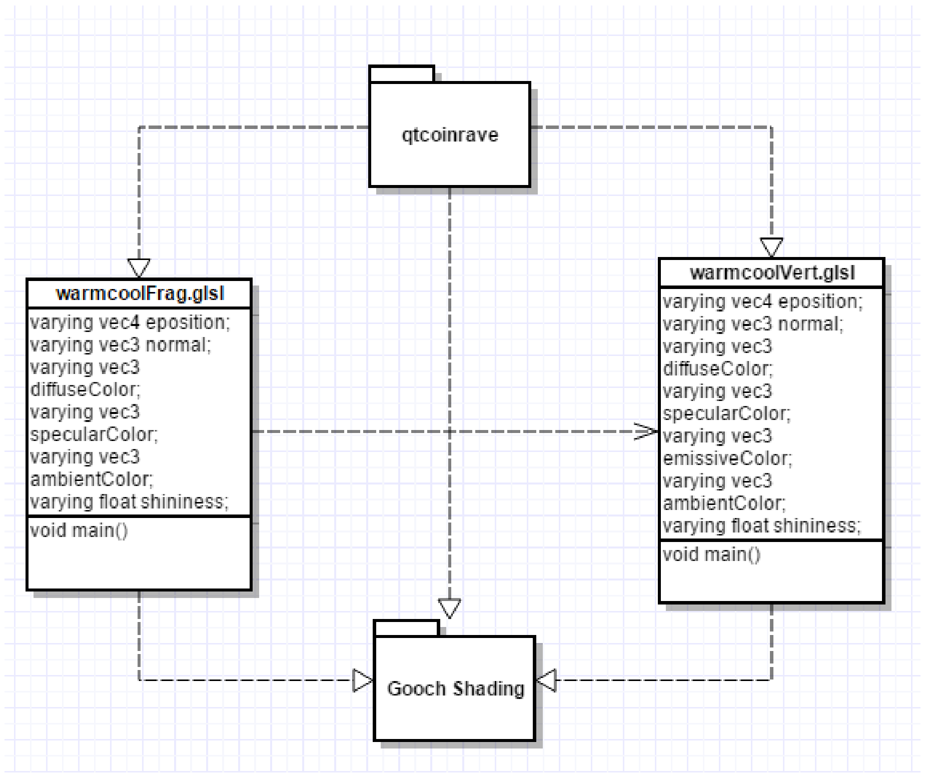


Fig. 3.
Logical Viewpoint Diagram that shows the implementations of Gooch Shading

3.5.1 Design View

In the code, the Gooch Shading will be implemented by passing vertex and fragment shaders to qtcoinviewer. For the initial Gooch Shading implementation, we will use Julio Medina's vertex and fragment shaders. From there, we will modify it to assure that the object's original color is not completely lost. Julio makes use of the following variables in his Vertex Shader:

- Eye position (eposition.
- Object normal vector.
- Specular Color (from object).
- Shininess (from object).
- diffuse color (from object).
- Ambient color (from object).

The warm and cool colors are determined by the developer and the object color is supplied by the python file passed into OpenRave. Both shaders will be using GLSL version 1.2. The vertex shader is used to retrieve the object's color, calculate the eye position, calculate the normal vector, and set up clipping. The vertex shader will retrieve the object's color using the gl_FrontMaterial call. The eye position will be computed by multiplying the model view matrix with gl_Vertex. The normal vector will be computed by multiplying the model view matrix with gl_Normal. World clipping will be computed by multiplying the model view projection matrix with gl_Vertex. After this, the vertex shader outputs the colors, the eye position, and the object's normal vector to the fragment shader. The fragment shader must then compute the ambient, diffuse, and specular lighting along with the augmented warm and cool colors. The ambient lighting for the object is determined by multiplying the ambient color with the ambient lighting. The diffuse lighting for the object is determined using the equation diffuseColor*NT*LightColor where NT is the dot product of the normal and light vectors. The specular lighting for the object is determined by multiplying the specular color with the light color along with a shininess value. Following that, the augmented warm and cool colors are determined using the formula: color + object_color*gooch_shading_weight, and then linearly interpolated using the GLSL mix() function [3]. This interpolation determines the shade coloring for that specific fragment of the object.

4 Requirement: Silhouettes

By Matthew Huang

4.1 Design Concerns

Silhouettes need to give the 3D shapes in the simulation solid borders so that the objects in the scene can be differentiated from each other. The silhouettes need to be working in situations where objects overlap with each other and under multiple viewpoints. The algorithm for how the silhouettes are drawn needs to be clear.

4.2 Design Stakeholders

Cindy Grimm, Justin Bibler, Matthew Huang, and Daniel Goh

4.3 Context Viewpoint

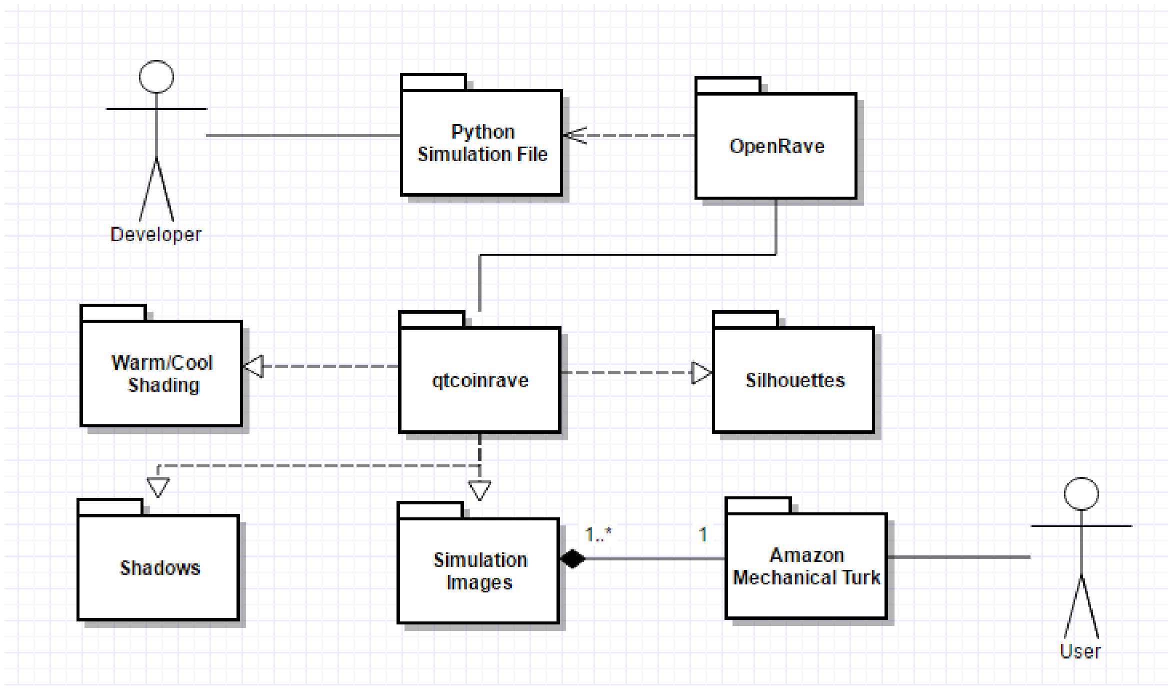


Fig. 4.
Context Viewpoint Diagram that that shows relation of silhouettes to qtcoin and OpenRave

4.3.1 Design View

The context viewpoint for silhouettes is essentially the same as the one for shaders. That is to say, its end goal is to improve the end user experience by enhancing the images sent to the Amazon Mechanical Turk. Additionally, it will also be implemented in the qtcoinrave plugin. Where it differs, however, is in how it affects the simulation images. Instead of highlighting 3D object geometry like the Gooch Shading will do, silhouettes will emphasize an object's borders (I.E. where an object begins and ends). These silhouettes are meant to help users differentiate between objects in the simulation images.

4.4 Compositional Viewpoint

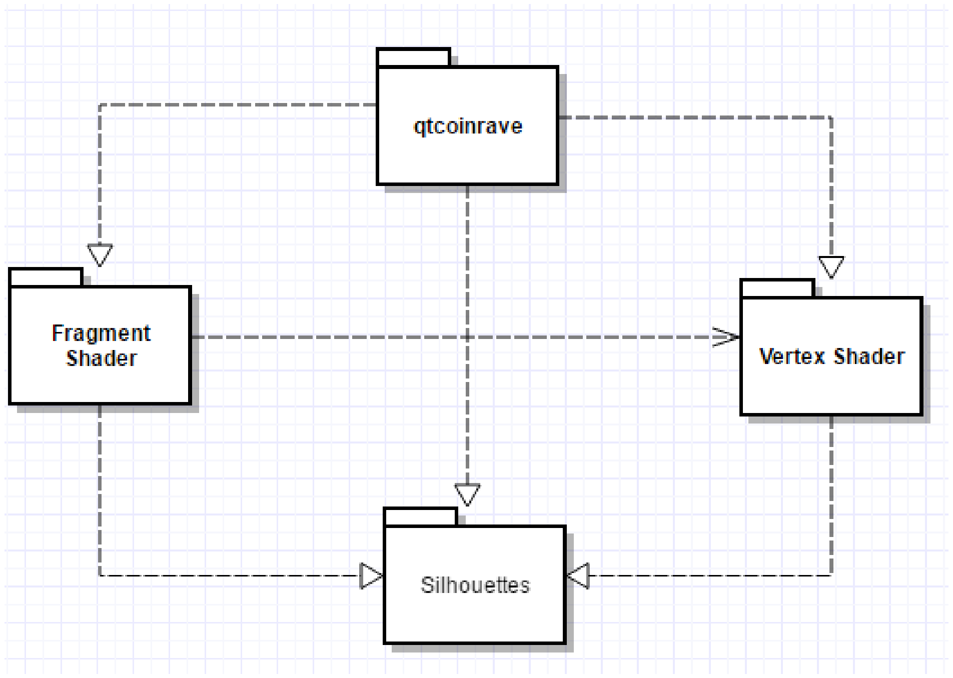


Fig. 5.
Compositional Viewpoint Diagram that show relationship of Silhouettes

4.4.1 Design View

Due to the fact that we don't have direct access to the main rendering loop, we will be using a simpler method for implementing silhouettes called cel shading silhouettes. These silhouettes make use of the vertex and fragment shader similar to Gooch shading. In the vertex shader, the object (robot grasping arm and the shapes it picks up) are enlarged in the direction of their normals. The normals can be calculated within the vertex shader using the equation gl_Position = gl_ModelViewProjectionMatrix*vec4(gl_vertex.xyz*scaler, 1.). The amount that the scene is scaled is based on the scaler variable. The fragment shader will color the silhouette by assigning a vec4 to gl_FragColor. Each shader will be loaded into qtcoinviewer.cpp using the SoShaderProgram object provided in the Coin3D API.

4.5 Algorithm Viewpoint

Silhouette Algorithm pseudo-code implementation

```
SoShaderObject->FragmentShader = silhouetteFrag.glsl
SoShaderObject->VertexShader = silhouetteVert.glsl
glEnable(GL\_CULL\_FACE)          \\enable culling
glCullFace(GL\_FRONT)             \\enable culling of front faces
glDepthMask(GL\_TRUE)             \\enable z-buffer writes.
GLRender()
SoShaderObject->FragmentShader = warmcoolFrag.glsl
SoShaderObject->VertexShader = warmcoolVert.glsl
glCullFace(GL\_BACK)              \\enable culling of back faces
glEnable(GL\_DEPTH\_TEST);
GLRender()
```

4.5.1 Design View

The pseudo-code created above is partially based off of the code available found here [4]. This silhouette algorithm requires two rendering passes; the first to draw the silhouette and the second to draw the objects (which in our case will be warm cool shaded). Thus, to implement this method, we will need to be able to control when qtcoinviewer renders objects. As for the algorithm, it uses the z buffer along with the glCullFace function to render the images in a way where the object overlaps with the silhouette. The z buffer allows us to have depth in our scene meaning objects can be drawn in the same place while still retaining their individual model coordinates. The glCullFace functions culls either the front faces, or the back faces depending on how you set it. Culling the front faces means that only the back faces of the object will be rendered (the front faces will be discarded). By culling the front faces during silhouette rendering, we can ensure that the silhouettes remain in the background while the warm cool scene sits in the foreground. Culling the back faces during the warm cool scene render is done to improve efficiency.

5 Requirement: Runtime Analysis using CodeXL

By Matthew Huang

5.1 Design Concerns

The modifications made to the current OpenRave system (Gooch Shading, silhouettes, and shadows) may slow the system down. As a requirement, the modified system must run at a minimum 30 frames per second (fps). As a requirement, the modified system must render the initial simulation image within 30 seconds. To assist with these requirements, the debugger shall have the ability to track function calls.

5.2 Design Stakeholders

Cindy Grimm, Justin Bibler, Matthew Huang, and Daniel Goh

5.3 Context Viewpoint

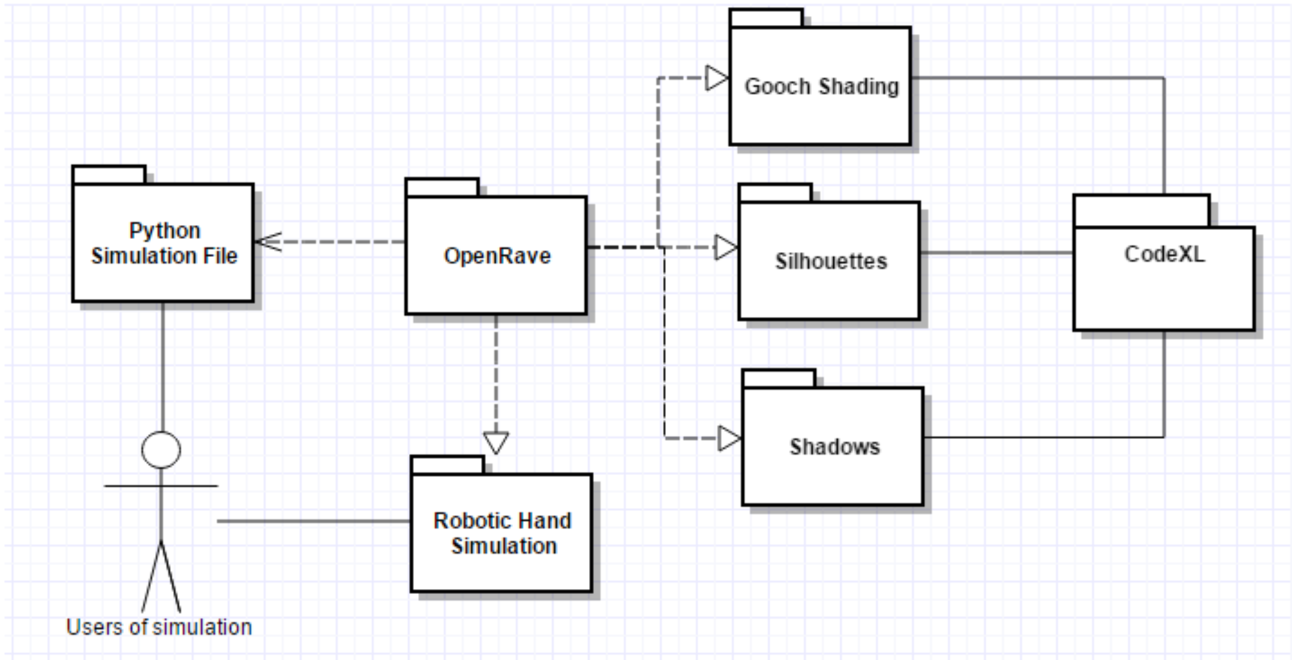


Fig. 6.
Context Viewpoint Diagram that shows relationship of CodeXL

5.3.1 Design View

CodeXL will be primarily used to track our Gooch Shading, silhouette, and shadow implementations. Its main goal is to monitor the effect that these three implementation have on the OpenRave simulation. If the implementations cause the robotic hand simulation to drop below 30 frames per second, CodeXL should be able to pinpoint where the cause originated from. The goal of this is to ensure a pleasant and smooth experience for the users of the robotic hand simulation.

5.4 Dependancy Viewpoint

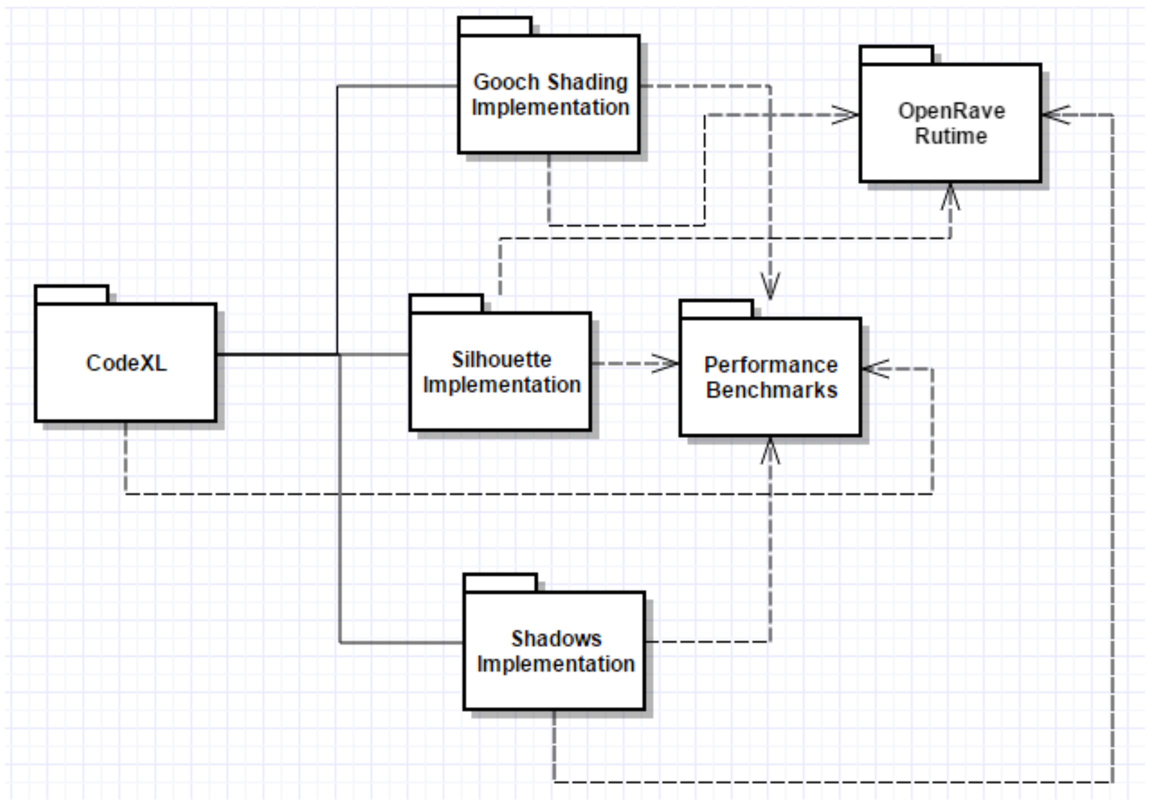


Fig. 7.
Dependancy Viewpoint Diagram that shows dependencies of CodeXL

5.4.1 Design View

How CodeXL monitors the three implementations will depend on the performance benchmarks for the whole system. If the implementations are in and the performance benchmarks are met, inefficient function calls or other issues brought up by CodeXL may be ignored. If the implementations bring the system down below performance benchmarks, CodeXL will have to display what is happening in the system that is causing it to slow down. Causes like large amounts of the same function calls, long running functions, and other bugs should be tracked. Specifically, if the frames drop below 30fps, CodeXL should be able track which implementation (or which combination of implementations) is causing the most performance loss. If the silhouettes are causing significant performance loss, they will be the first to be refactored since there are many ways to implement silhouettes. Gooch Shading is less flexible in its implementation so refactoring it will have a lower priority (shadows is somewhere between silhouettes and Gooch Shading).

If OpenRave takes more than 30 seconds to render, problems likely exist in how we initialize our scene. In this situation, CodeXL will be used to primarily track activity in the Display function since that is the function the renders the initial scene. Other object initialization functions will not be tracked since we assume that they are pulled from libraries we have no control over. Additionally, performance benchmarks will only be changed as a last resort (I.E. we cannot refactor our implementations anymore).

5.5 Interface Viewpoint

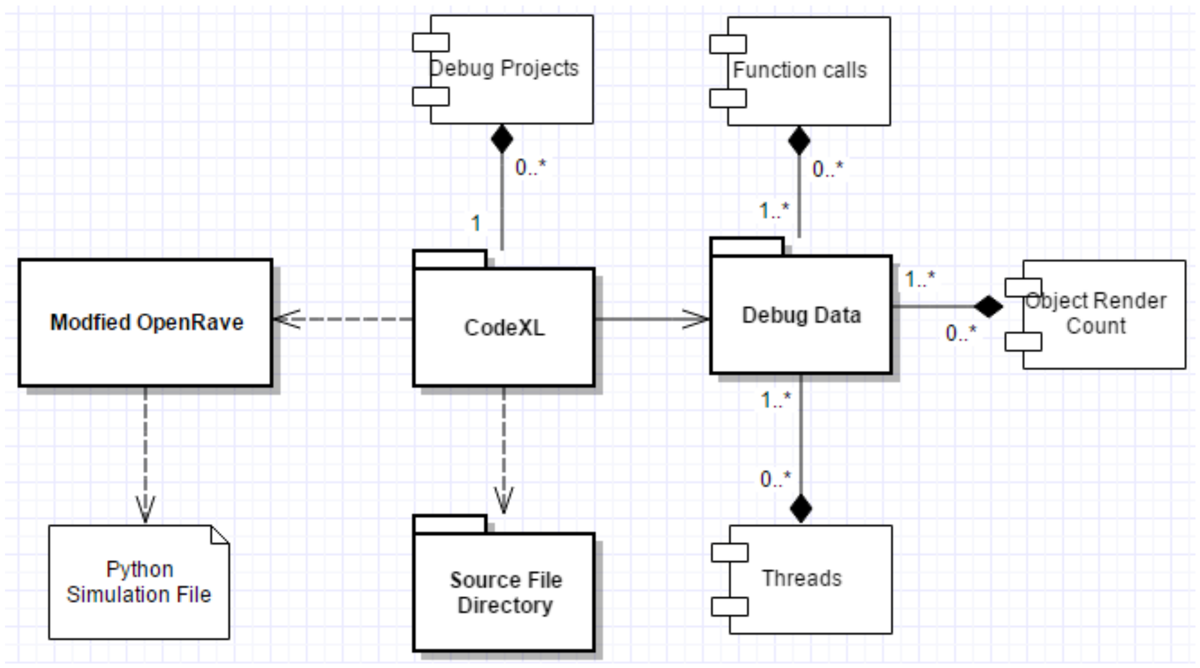


Fig. 8.
Interface Viewpoint Diagram that shows the interface relations of CodeXL

5.5.1 Design View

To monitor OpenRave, create a debug project in CodeXL and set the OpenRave executable as the debug project's executable. Additionally, provide the python simulation file as a command line argument and set the source file directory to where the python file lives (this is all done in creating the debug project). Afterwards, execute the debug project and CodeXL will display debug information such as rendered objects, created threads, and function calls (if you turn that on). Note, however, that tracking function calls slows down CodeXL significantly so it shouldn't be used for basic testing. Past that, CodeXL also provides access to the source code currently running directly on the CodeXL GUI.

6 Requirement: Shadows

By Justin Bibler

6.1 Software Design Description

Our client requires us to add shadows into the simulation's graphics. The addition of shadows will improve the overall graphics of the simulation, which will help a user better understand where an object is within a 3D space.

6.2 Design Stakeholders

Justin Bibler, Matthew Huang, and Daniel Goh

6.3 Design Concerns

All stakeholder share similar design concerns.
Shadows design concerns:

1) Will our shadow implementations have a major impact on FPS.
2) Will our shadows be too pixelated.
3) Will users be able to better understand where an object is because of our shadow implementation.

6.4 Design Views

6.4.1 Interface Design

This section is dedicated to talking about the graphical quality of our shadow implementation.

6.4.1.1 Design Concerns

Interface design concerns:

1) Will users be able to better understand where an object is because of our shadow implementations.
2) Will our shadows be too pixelated.

6.4.1.2 Design Elements

Name

Shadow Interface.

Type

Visual, or Graphic.

Purpose

Shadows with the simulation exist to better help users understand an objects location within a 3D space.

Relationship

The quality of the visuals are primarily determined by how the shadows are implemented in the shadow group class.

Interface attribute

As long as an object is within the spotlight, the object should cast a shadow onto it's surrounding objects. Whether or not the shadow is visible is dependent on if there is a surface that is blocking another surface from a light source. These shadows should have minimal pixelation. In addition, the shadows should be relatively close in shape to their original models, as to not confuse what shadow is projected out from what object. The shadows should also have a realistic coloration to them, meaning a shadow's dark spot should be clearly visible on all objects it's projecting onto. All of this is done so that the users will easily be able to understand what shadow is related to what object, and to receive a better understanding of the 3D environment in the simulation.

6.4.2 Algorithm Viewpoint

The algorithm for calculating the shadows is done in the shadow group class within coin3d. This class creates shadows by using the dynamic shadow mapping technique.

6.4.2.1 Design Concerns

Algorithm design concerns:

1) Will the algorithm be costly to the simulation's FPS.

6.4.2.2 Design Elements

Name

Algorithmic implementation of shadows.

Type

Algorithm, or method.

Relationship

Shaders, and the Shadow Interface.

Design Constraints

The major concern about using the algorithm within the shadow group class is that it uses it's own built in vertex, and fragment shaders. This introduces a problem when attempting to run multiple different shaders. Essentially, by using the shadow group class we lock our shaders down to the default ones created within the class. It is however possible to override the default shaders with custom shaders, meaning knowledge of how the algorithm works is important for when we have to create new shaders that will work with all our other implementations. Another constraint is that we wish to not edit anything within the coin3d library. We wish to treat it primarily as a black box where we construct modules and pass them in to be rendered. This introduces a problem in which we must create a shader that has an algorithm that interacts with the shadow group class shadowing algorithm. Finally, there is minimal documentation about how to use the shadow group class. Therefore, it may prove difficult to work with the shadow group class.

Processing Attribute

The bases of the algorithm depends on the use of the depth buffer to test whether or not a fragment is in between a shadow casting fragment. In the first render the depth of each fragment is calculated from the point of view of the light source. In this case it will be from the spotlight class that is placed in the scene, and is bound to the shadow group class. On the second render, the scene will be rendered as normal except with a test that will examine if our current fragment is blocked by another object. The test is done by checking to see if our current fragment is further from the light then the shadowmap that we generated. If the fragment is within that shadowmap we know that a shadow is being casted by another object onto our current location. Thus, a shadow should be draw at this fragment. [5]

7 Requirement: Measuring the Simulation's FPS

By Justin Bibler

7.1 Software Design Description

Our project requires us to measure and analyze the runtime and FPS of our altered simulation. This is done so that we can ensure that our additions fulfill our performance metrics: the simulation must maintain an average 30 FPS. If we are unable to reach this metric our implementations will be considered incorrect, and will require revisions.

7.2 Design Stakeholders

Justin Bibler, Matthew Huang, and Daniel Goh

7.3 Design Concerns

All stakeholders share similar design concerns.
Requirement design concerns:

1) Failing to meet our target FPS and runtime will result in revisions of our implementations.
2) Will our measurements be accurate for the majority of computers.
3) Software used for measuring may cost money.

7.4 Design Views

7.4.1 Information Viewpoint

The purpose of this viewpoint is describe how information will be gathered, and how it will be analyzed. Our team will be using an external tool to obtain data related to the simulation's FPS. FRAPS will be used to examine the FPS and export the information into a file to be analyzed.

7.4.1.1 Design Concern

Interface viewpoint concerns:

1) FRAPS is paid software.
2) Will our measurements be accurate for the majority of computers.

7.4.1.2 Design Elements

Name

Method of analyzing and collecting FPS data.

Type

Method.

Purpose

Data collected will be analyzed to ensure that our implementations into the simulation meet our requirements.

Data attribute

FRAPS will be used to capture data related to FPS, this will be done by using the benchmarking tools that come with the software. We will run this test for one minute ten times, on five different sample simulations with our updated graphics. This data will be exported into a file, from here we will analyze the data by finding the mean and median across all executions. If these values are within a certain standard deviation of 30 FPS, our implementation will be considered correct.

Author

FRAPS is owned by Beepa.[1]

7.5 Rational

Even though FRAPS is a paid software I've decided that it be best if we use it. It's ability to export data into a file allows our group to easily obtain and analyze data. Furthermore, our concern related to the validity of the data stems from the fact that our group members own above average computers. Thus, testing on our computers would most likely be inaccurate relative to the average computer. To circumvent this problem we plan to do most of our testing on the computers on the OSU campus.

8 Requirement: Code Maintainability

By Justin Bibler

8.1 Software Design Description

The goal behind code maintainability is to increase the cost effectiveness of changes being made to a system. [6] Our group has made this a requirement because our client wishes to be able to easily change, or add into our implementations. The primary logic behind this design is the use of good programming practices such as documentation, and commenting, but also the use of external resources which are described within the viewpoint below.

8.2 Design Stakeholders

Justin Bibler, Matthew Huang, and Daniel Goh

8.3 Design Concerns

All stakeholders share similar design concerns.
Requirement design concerns:

1) Cost of static code debugging tools.
2) Lack of documentation for preexisting code.
3) Lack of comments for preexisting code.

8.4 Design Views

8.4.1 Resource Viewpoint

External resources will be used to help us maintain and upkeep our code quality. The first resource we will be using, which within today's standard is almost a necessity, will be GitHub. GitHub is a repository that is used for code version control. Secondly, we will be using Cppcheck. Cppcheck is a static code analysis tool that examines code for bugs that a compiler may not catch.

8.4.1.1 Design Concern

Interface viewpoint concerns:

1) Using Cppcheck may bring about problems within preexisting code not written by us.

8.4.1.2 Design Elements

Entities

Both GitHub, and Cppcheck are free to use by the public

Name

Resources for Code Maintainability.

Type

External resource.

Function

GitHub will be used for version control of the code. Cppcheck will be used to assist in finding and fixing bugs within our code implementations.

Resource attribute

GitHub is important because it will allow us to have easy access to past versions of code. Using GitHub is like a safety net that will allow us to be free with how we edit the code, because in the case that we create a massive new problem; it is very easy for us to pull an earlier version of the code to work on. Along with this GitHub also has a built in wiki that will be used to record and document our new implementations and functions within the code. Allowing for easy reference for ourselves and our client.

Cppcheck will be used to examine our new implementations into the code. Using this tool will allow us to quickly find and eliminate bugs within the code. Few amount of bugs within code will allow us to make changing and creating new implementations safer and easier.

Author

GitHub is owned by GitHub, Inc. Cppcheck is open source.

8.5 Rational

The reason Cppcheck was decided upon was because it is open source and free, other static analysis tools cost a significant amount of money. Which makes the Cppcheck option the best for our group. Also, the design behind ”Code Maintainability” can be improved by the use of the resources I talked about. However, a large part of it will rely more on how well our group documents the improvements we make through resources such as GitHub. It'll be important while we go forward to document both how we implemented our improvements, and how to use our improvements. This is because, as mentioned in the Requirement concerns, we are working off preexisting code. Thus, our client will need some sort of good reference of how to navigate the code, and how to make proper adjustments.

9 Requirement: Creation of Online Survey

By Daniel Goh

9.1 Software Design Description

This requirement is established to allow participants to provide their input to determine if the team's visual enhancement to the robotic simulation has improved from its initial state. This will be fulfilled by running online surveys. Qualtrics is the selected platform to run the online survey.

9.2 Design Stakeholders

Justin Bibler, Matthew Huang, and Daniel Goh

9.3 Design Views

9.3.1 Interface Viewpoint

9.3.1.1 Design Concern

Qualtrics is the selected service that will be utilized to carry out the online survey. The design concern for the online survey will include if participants will be able to understand the questions of the online survey and select their choice of better visuals.

9.3.1.2 Design Elements

Name

Survey Interface Understandability

Type

System used to assess resulting visuals for the robot grasping simulation.

Purpose

This element exists to help guide the creation of an online survey that is easily understood and usable by the participants.

Function

Qualtrics [7] is a matured survey platform that allows unlimited number of questions and the function to attach images to the created survey. Participants will be able to select a picture or answer based on the questions presented. The questions will cover the aesthetic aspects between pictures, the presence of shadows between pictures, and identification of object relativity or position within the picture.

Interface

Methods of interaction provided by Qualtrics include:

- Allowing participants to view attached images for a question
- Allowing participants to select a single answer or image for a question
- Allowing participants to select multiple answers or images for a question (checkboxes)
- Participants responses will be stored and will be evaluated using the built-in data analysis and visualization tools.

Users should not be overwhelmed by information in the interface while taking the survey. Survey questions should ask the user for a specific opinion (e.g. questions asking about presence of shadows should only focus on asking about shadows and nothing more) at a time and ideally have no more than one sentence [8]. Survey questions should be designed to be balanced and not biased. For example, stating the left visual is the improved visual in the question while asking the user to select the better visual is a form of bias.

10 Requirement: Analysis and Visualization of Collected Data

By Daniel Goh

10.1 Software Design Description

The data collected from the online survey will require analysis to determine if the team's visual enhancement to the robotic simulation has improved from its initial state. The data will need to be organized and analyzable by the stakeholders. Qualtrics will be used to analyze and visualize the collected data.

10.2 Design Stakeholders

Justin Bibler, Matthew Huang, and Daniel Goh

10.3 Design Views

10.3.1 Information Viewpoint

10.3.1.1 Design Concerns

Qualtrics [7] is the selected platform to parse and visualize the data collected from the online survey. The design concern includes if the data collected can be analyzed and visualized meaningfully to reflect the collective participants' responses.

10.3.1.2 Design Elements

Name

Data Visualization

Type

System to analyze and visualize collected responses from the survey.

Purpose

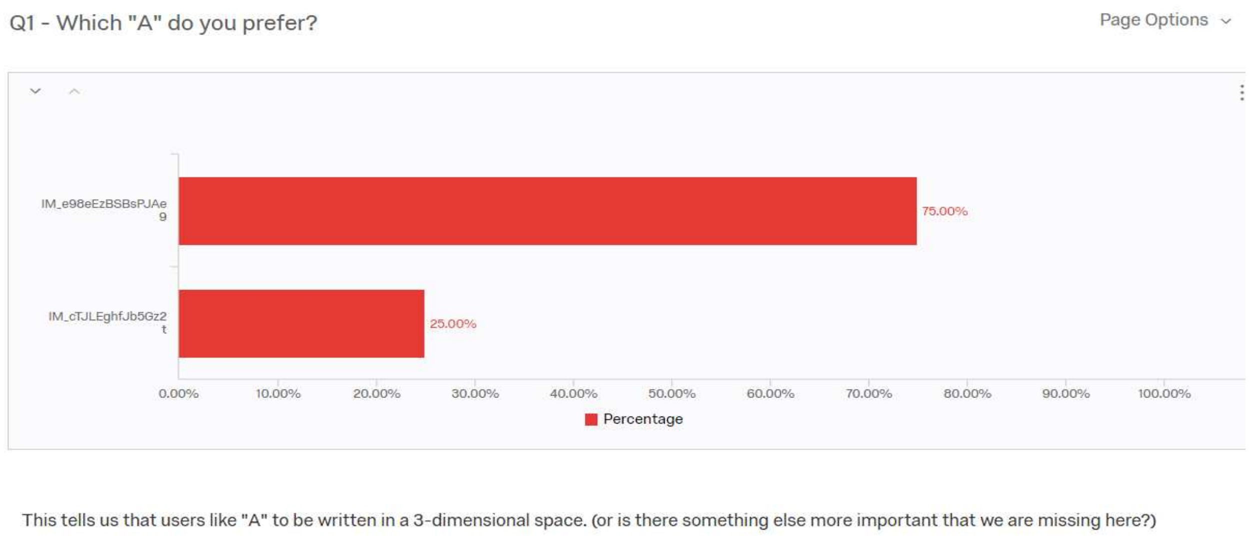
The element exists to help guide the use of Qualtrics to create data visualizations.

Data Attribute

As the survey data is collected by Qualtrics, the data analysis and visualization tools will have access to parse the survey data. The resulting data content will be generated as a report within Qualtrics.

The survey is aimed to determine if the team's graphical enhancement did improve the users understanding of the image. With that, the selected data analysis method that will be used to determine the project's success will be based on frequency among all respondents [9].

The aim of the performance metrics is to get at least 80% responses to select the image with the team's enhancements. The visualizations will be created with the percentage function within Qualtrics.



This tells us that users like "A" to be written in a 3-dimensional space. (or is there something else more important that we are missing here?)

Fig. 9.
An example survey in which the responses are shown in percentage form. This will allow the team to easily determine if the aesthetic performance metrics are met.

11 Conclusion

In conclusion, this design document will be used as a plan of action as we go forward with our implementation. This document is not set in stone; it may change as needed as we begin making changes to the code. However, it is important to note that our requirements will not change; only our approach to tackle the requirements may change.

Cindy Grimm	Date
Justin Bibler	Date
Matthew Huang	Date
Daniel Goh	Date