LUME

A Real-Time 3D Interactive Platformer

Jonathan Rawson

Introduction

Video game design is a constantly expanding industry, grossing more than 31.6 billion dollars annually. Game design is not only the work of large entertainment companies but has also become available to small companies, freelancing, and individuals. This growing industry is one of the major contributors to new and upcoming technologies in the computer science field. Computer games and graphics are constantly evolving with the invention of better hardware and more efficient algorithms. The challenge in game design is to create something unique and up-to-date with the latest technologies. Adventure games, in particular, require a large amount of content in order to drive the objective-based system. It is this genre of video game that provides a large amount of revenue for the game industry, often spawning from storylines of books and movies. Because of the impact of this genre and the expansiveness of the game industry as a whole, our team chose to implement a 3D interactive adventure game for our senior project.

Project Lume was a two quarter long real-time graphics project. Our design team consisted of seven team members: Mike Buerli, Brent Dimapilis, Trent Ellingsen, Jeff Good, Teal Owyang, Jonathan Rawson, and Ryan Schroeder. As a team, we successfully developed a fully playable computer game, along with the necessary engine, content, and tools needed to create such a game. This project not only produced a unique and interactive 3d game, but also gave team members invaluable experience with computer gaming and graphics technologies.

Project Overview

Lume is a unique 3D adventure game created in C++ and OpenGL. The game exhibits a large array of computer graphics technologies complemented by a strong story and distinctive aesthetic appeal. Lume also uses a different style of controls by implementing both first and third person perspectives, while still keeping the controls intuitive and fluid.

Lume is centered on a creation and sandbox feel intended to give the user a fully immersive adventure experience. Players are encouraged to build upon the world in order to complete objectives, reach checkpoints, and gain more abilities. The world originally starts dark and desolate, however, when the user interacts with buildings and moves throughout the world, they give light and life back to the world. This provides the player with a strong sense of influence over their surroundings. Throughout the course of the game, the player gains more abilities through leveling up such as the ability to control of moving platforms, further extend blocks, and jump through blocks that have been built. All of these abilities must be utilized in order for the player to reach new heights and complete the various objectives for each level.

The main objective of Lume is to reach the top of a collection of buildings, which comprise a level, to harness more energy. Focusing on this simple idea is imperative in game development. Abilities and controls must be simple and engaging; a game must begin with a simple concept and build upon that concept with features that help bring more value to the game by either improving basic game play or

strengthening the story. The focus in Lume was the ability to extend blocks from any building in the world and climb up them, allowing the user to choose their own path through various levels. Throughout the development of Lume, other features were added to help strengthen the story or improve the game play.

Look and feel is another vital component to game development. A game must have a very cohesive theme and style in order to keep the user engaged. Lume's theme focuses on futuristic and abstract lighting with heavy contrasts between light and dark. Objects are constantly pulsating with a darker shade of light, which contrasts the bright glow of the edges surrounding buildings. The color pallet of the world includes a variety of neon colors, which help convey a futuristic style. The look and feel of a game must also help convey the story. Lume focuses on tying a relationship between the dark and bland qualities of the world to objects that KOG is controlling. As the player progresses through the game they harness energy from KOG and create a brighter and more colorful world, with the newly harnessed energy providing a stark contrast between light and dark. Players can add color to the world by creating blocks on any building. Organic, colorful patterns extend from each block a player creates in the world. Lastly, to further convey the differences between KOG and the player, objects associated with KOG are much more linear and rigid, while objects associated with the player are more organic and abstract.

The last element and platform for which a game is built upon is the storyline. The KOG story was developed by teammates throughout the two quarters plotting out much more detail than is visible in the game. The story is of a future civilization,

which is under the complete control of KOG, whose only hope of survival is Lume (the player). The story takes on the classic battle of humanity versus technology as well as nature versus industry. The game begins in the year 4200, where KOG, a privatized company, now controls all civilization. All of Earth's resources have been depleted and the world is now solely based on energy. Project Lume is an experiment created by KOG to provide a new and more efficient way of storing energy. Lume (the character) is the first prototype of this experiment, which KOG quickly realizes is a failure due to Lume's unexpected ability to manipulate the energy he collects.

A key element of the story is the background of Insight. Insight is a computer AI designed in 3119 at the creation of KOG as a fail-safe to disable KOG should it ever gain too much control. Presently, one thousand years after KOG's rise to power, Insight finally sees its chance to take KOG down using Project Lume. The first objective of the game is to download Insight (initially the player does not know what they are downloading). Once Insight is downloaded it talks to the player via the Heads Up Display (HUD), giving the player feedback and an introduction to the game's plot. Insight guides you through a series of levels, in which you harness an increasing amount of KOG's energy. Once all of the energy is collected you can reach the upper city, where the player is given the opportunity to finally defeat KOG. By shutting KOG down, all energy is relinquished back to the earth, allowing humanity and nature to start once again.

Related Work

The general look and feel for Lume was inspired by Disney's movie, Tron: Legacy (Figure 1). Lume utilizes the dark and moody electronic sounds of the Tron universe as inspiration to create a unique ambiance for the player as they visit the different sections of the KOG empire. Visually, the user is presented with an initially dark and bland world that lights up in vibrant neon oranges, blues, and greens as the player harnesses more energy and interacts with different areas of a level. The building architecture mimics the rigid futuristic building style used in Tron but adds a unique organic texture style to the sides of buildings that the player interacts with.

Figure 1: Tron Legacy

Throughout the development of Lume, the concentration was adding new innovations to the 3D platformer genre. Lume's gameplay was inspired by several other 3D platformers including: Assassin's Creed by Ubisoft, Super Mario 64 by Nintendo, and Mirror's Edge by Electronic Arts. Traversing through the world by rooftop was a mechanic used in Mirror's Edge (Figure 2) and Assassin's Creed

(Figure 3). The inspiration behind completing objectives in the different KOG districts stems from the star collection mechanic of Super Mario 64 (Figure 4). The key difference between these games and Lume is the innovative way in which the player traverses the world. As developers our intent was to create an intuitive path through the levels, however, with the ability to expand on buildings in the KOG world, the player is free to create their own path through a level. Minecraft was the main inspiration behind allowing the player to modify the world by building blocks everywhere. Allowing the player to change the world grants them the ability to form their own path and visually alter a level differently each time they play.

Figure 2: Mirror's Edge

Figure 3: Assassin's Creed

Figure 4: Super Mario 64

Technologies

Lume implements a variety of graphics technologies that allow the game to have an aesthetically pleasing look, while running in an efficient manner. Below is a list of the various technologies that was implemented, with the team member(s) that worked on it:

- 3D Interactive Environment (All)
- Collision Detection (Mike Buerli, Ryan Schroeder)
- Spacial Data Structure (Mike Buerli)
- Frame Buffer Objects (Mike Buerli)
- 3D Modeling and Animation (Teal Owyang, Brent Dimapilis)
- Model importer (Teal Owyang)
- "Spring-Loaded" Camera (Jeffrey Good)
- Freeform Camera (Ryan Schroeder)
- Camera Pathing (Ryan Schroeder)
- Robot AI/Pathing (Jeffrey Good)
- Bloom Shader (Jeffrey Good)
- Mapper/Importer (Jeffrey Good, Jonathan Rawson)
- Level Design (Jonathan Rawson, Jeffrey Good, Trent Ellingsen, Ryan Schroeder)
- Objectives (Ryan Schroeder)
- Growing Objects (Mike Buerli)
- Moving Objects (Jonathan Rawson)
- Heads Up Display/Main Menu (Jonathan Rawson)
- Picking (Mike Buerli)
- Player Logic (Ryan Schroeder)
- View Frustum Culling (Ryan Schroeder)
- Particle Explosions (Trent Ellingsen)
- Dynamic Abstract Lighting (Mike Buerli)
- Animated Textures (Trent Ellingsen)
- Textures & Logo design (Trent Ellingsen)
- Simple Level of Detail (Trent Ellingsen)
- Orbs Particle System (Brent Dimapilis)
- 2D Billboard Texturing (Brent Dimapilis)
- Robot/Plant Texturing (Brent Dimapilis)

Overview – Look and Feel

Within the game environment there are three 3D models that represent characters. These models were created using Blender and include the main character 'Lume' as well as two of the enemy robots controlled by the KOG corporation. The levels in which the game's world resides were created through a standalone map creating program. The mapper exports a custom file type, developed by the team, that the game is able to interpret and construct each of the worlds with. One of the technologies implemented in Lume was billboarding to simulate a 3D look using 2D

objects. To create the look and feel of futuristic posters and logos, 2D textures were placed in 3D space and were alpha blended create the effect of glowing advertisements. There are two particle systems in place, the first occurs when the character gathers of energy and there are orbs that follow the character in a flocking simulation, the second occurs when robots are sprinted through and killed. Using both bloom and blur effects, the outer glow of the building was manufactured. Animated textures were used to create the title, intro, and loading screens.

Overview – Optimization

Lume implements several technologies that allow for optimized gameplay. To help the bottleneck of the graphics pipeline, Lume does not send all the geometry down the pipeline every frame. Using view frustum culling, objects that are not currently being viewed by the camera are culled out and not rasterized by the GPU. Skyline, a level in Lume, has a group of 70 robots. In order to continue ensuring smooth gameplay during Skyline, level of detail was implemented to draw objects with less geometry when the player is further away and objects with full geometry when closer. Each object drawn to the screen has an update function that is called during every frame. The objects are rendered this way in order to alter color or move blocks. The object update function is turned off when objects are too far away from the player. To further improve performance, a uniform spacial data structure was implemented that breaks up the world into a 3D grid. The 3D grid allows for testing collisions based upon the character's position. The hit test function is only used to check the collisions of objects occupying the surrounding bucket spaces. To

optimize the collision detection, Lume uses axis aligned bounding boxes to test the potential hits.

Mapper/Importer

Early on, it was clear that Lume would benefit from a level editor tailored specifically to the game. Initially, Call of Duty's Gradient Map Editor was looked into as a possible solution for this problem. Through research and testing it was discovered that the vertices output by the Gradient Editor did not adequately translate to the vertices used by OpenGL. Instead of outputting the vertices for drawing, when an object was created using Gradient, three vertices were specified to indicate a plane, and the object's position was determined by the intersection of six planes.

In place of using vector algebra to determine the side the object lay on with respect to each plane, a "mapper" program was developed specifically for the needs of levels in Lume. The mapper uses glutCreateSubWindow and glutSetWindow to divide the main window into three sub sections: the top element allows for editing of the map and viewing of all objects in any 2D plane configuration (x-y, y-z, x-z), the bottom left element is a real time rendering of the map in 3D, and the bottom right element allows for texture mapping and specification of types for each object in the map. Each object that is created by the user is stored as a Brush with data for drawing, type, and texture number.

The advantage of making a mapper specifically for Lume comes from having complete control of the way in which it is exported. Originally the exporter wrote

only the smallest and largest x, y, and z value for each object to a file. After spending time making maps, the team quickly realized that it would also be beneficial to include a value for the texture used on each object. This satisfied the needs of the team until different types of objects were being drawn in the game. Once models were included, the mapper needed yet another value exported that represented what type each object was. This was stored as a string that, when parsed by the importer, created the corresponding object type in the game's data structure. Code Figure 1 is an excerpt from the Tutorial map file, created by the mapper.

```
-200 22 -106 -189 24 -96 textures/concrete.jpg Spawn
225 426 -64 283 428 165 textures/concrete.jpg Cube
172 426 27 226 428 81 textures/concrete.jpg Cube
131 425 66 185 427 121 textures/concrete.jpg Cube
82 424 109 139 426 168 textures/concrete.jpg Cube
131 425 -16 185 427 40 textures/concrete.jpg Cube
83 424 -64 140 426 -7 textures/concrete.jpg Cube
-180 425 120 -30 427 171 textures/concrete.jpg Cube
-220 424 -26 -163 426 132 textures/concrete.jpg Cube
-180 425 -68 -36 427 -18 textures/concrete.jpg Cube
-178 33 -111 -159 37 -98 textures/sky/up.jpg Trigger Downloads
```

Code Figure 1 Mapper Export File

A challenge that arose from developing in the mapper and importing to the game was the way in which objects were oriented in each. Often times, an object would be drawn facing one way in the mapper, and would have a different orientation in the game. This happened because the mapper stored only the x, y, z values, and not the orientation in 3D space (mostly applicable for models and ramp objects). This was corrected by modifying the "type" string exported by the mapper. It was a procedure of trial and error, but if it was determined that an object did not have correct orientation in the game, the map was modified to export a 180 degree version of the

type. The most concrete example of this was for correcting a ramp so that the slope was drawn in a way that the player could walk up it, instead of facing the other way.

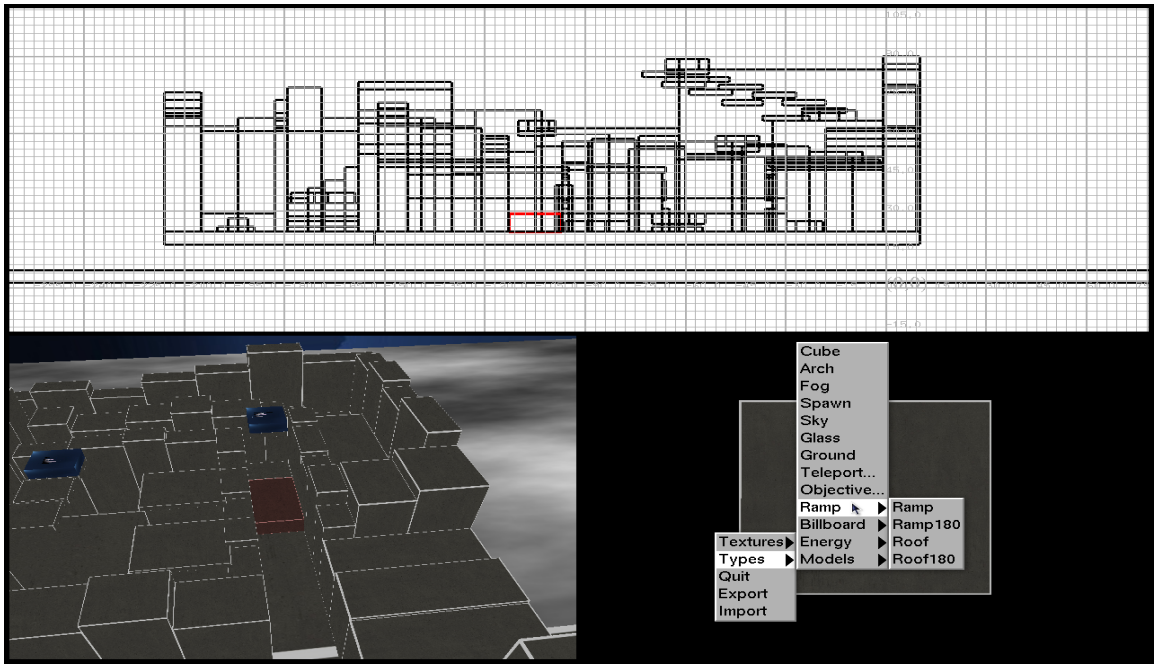

Figure 5 - Ramp in Mapper

Figure 6 - In Game Ramp

As Lume developed, it was decided that the main way for the story to be told would be through objectives, including the collection of objects related to the destruction

of KOG. Since these objectives were related to a character's position, there needed to be a way to include objectives in the mapper. The type "Trigger" was added to the mapper, with an extra parameter for the objective name that the trigger was applicable to. In the importer, anytime a "Trigger" was noted, it was added to the currentLevel's objectives vector, with a name for collectable item. This allowed for a dynamic number of objectives and collectable items per objective, while specifying the location of each item in the mapper.

```
if(!strcmp(token, "Trigger")){
   token = strtok(NULL, "\0");
   strcpy(type, token);
   trigger* trig = new trigger(x, y, z, dx, dy, dz, type);
   if(!levelExist){
      for(unsigned int i = 0; i < currentLevel->objectives.size(); i++){
         if(strcmp(type, currentLevel->objectives[i]->collectableName) == 0){
            currentLevel->objectives[i]->triggers.push_back(trig);
            if(i == 0){
               objective_start = pnt3d(trig->x, trig->y+0.5, trig->z);
}  }  }  }  }
```

Code Figure 2 - Triggers in ImportMap

Heads Up Display/Main Menu

One of the key components of a video game is the Heads Up Display and any Menus that the player navigates through. Each of these is accomplished by using orthogonal projections. This allows for a 2D layer to be rendered on top of the 3D scene that already exists. OpenGL and glut provide functions that make this possible.

In order to create a 2D layer, a new matrix must be pushed onto the projection matrix stack, and then modified with a call to gluOrtho2D. With appropriate translates and scaling (accounting for the y-values in glut ranging from top to bottom, instead of bottom to top), any 2D shape that is drawn when switching back

to the modelview matrix will be rendered on top of the 3D scene. After all drawing for the 2D layer is complete, the matrix on the top of the projection matrix stack must be popped off so that any additional drawing is back in the 3D scene.

Lume's HUD and Main Menu were both implemented using this orthogonal projection technique. When switching between the Main Menu and the game, the game state needed to be paused so that any extraneous key presses did not affect the player while in the menu. This was achieved by updating a boolean that represented whether or not the game was paused. If it was, the game state would not be affected, and the display function would only draw the Main Menu. Code Figure 3 shows how the two results of hitting the ESC key. In addition, values in the menu are highlighted when the player scrolls over them with the mouse. This was achieved by passing information about the mouse position to the HUD and updating the menu accordingly for an interactive user experience.

```
case 27: //brings up the game menu
         if(bPause && character.gameStarted){
            startLoop = (long) glutGet(GLUT_ELAPSED_TIME);
            glutWarpPointer(GW / 2, GH / 2);
            glutTimerFunc(TIMERINT, gameLoop, TIMERINT);
            glutSetCursor(GLUT_CURSOR_NONE);
            bPause = false;
            if (currentLevel) sound.playLoop(currentLevel->music());
         }
         else if (character.gameStarted) {
            fbo->update(1000);
            glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
            bPause = true;
            sound.playLoop("Lume.ogg");
         }
         break;
```

Code Figure 3 - Main Menu Pause

Since the Main Menu included animated textures, the HUD also had to store Sprites that held the data for the textures to cycle through. The Sprites used for the HUD

included the orange experience bar, blue energy bar, the Main Menu screen, and the loading screen. In the update function of the HUD, each Sprite is updated as well so the correct frame is displayed in the HUD or Menu.

After completing the first level, the player has downloaded Insight, which communicates through the HUD. This was made possible by the creation of a new class that stores a vector of char* along with a duration for which they should be displayed. For this to be displayed in the HUD, an instance of Insight was made in the HUD class and incorporated into the existing update method.

Level Design

In order for Lume to have a meaningful impact on a player, the level design must be carefully considered with respect to elements of story, look and feel, and overall theme of the game. The Tutorial level is the player's first experience in the world created by Lume. Therefore, it was designed with elements that taught the player how to play as he progressed through the level (Figure 7 is taken from the beginning of the level). This included textures attached to walls that specified player controls and the purpose of certain elements in the game (such as the orb checkpoints).

Figure 7 - WASD Key in Tutorial

The Tutorial also had to be designed in a way that was straightforward to follow. For inexperienced players who are learning the game mechanics, a path is clear without being explicitly displayed with arrows. Creating varying sizes of buildings and placing orb checkpoints in strategic locations gave the level a sense of direction without forcing the player to follow one path. Orbs were placed in open spaces in the level so that the player would be guided towards a particular location without feeling like it was forced.

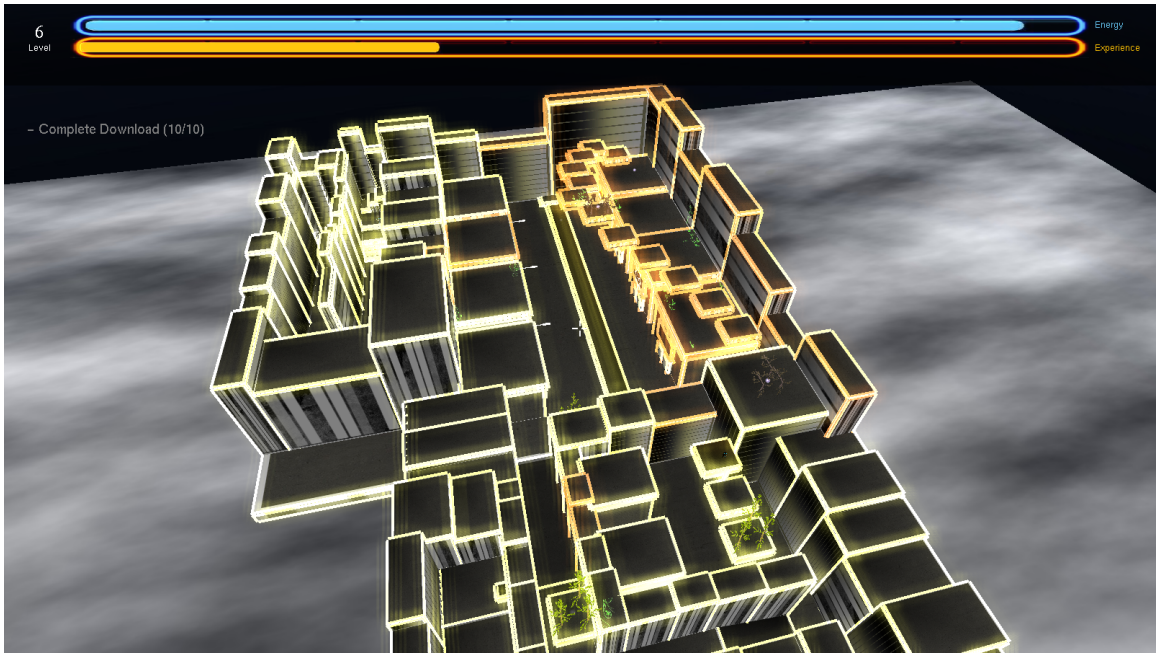

Figure 8 - Overhead of Tutorial

The first level also aimed to convey the feel of the city and a world run by KOG. This was made possible by textures placed on buildings to represent storefronts. Each texture was given the name of a KOG business, such as KOG Burger, which creates the feeling that this is an inhabitable world controlled by KOG. This was expanded upon by the dead trees that reside at each checkpoint and objective. The purpose of the dead trees was to give the player the feeling that he has the ability to bring life back to the world, which is essential to the story.

Moving Objects

Another important element of Lume is the puzzle layout of levels. One of the puzzles featured in the game is navigation of the player along the paths of moving objects. Since all of the objects represented in the mapper were static, a new way to create moving platforms was developed. With control over the importer, a new filetype was created to store Path nodes for Moving platforms and Robots.

The path format can be seen in the Code Figure 4. It was necessary to specify a varying amount of positions so that a Moving Object would not be constricted to moving between two nodes. Nodes were stored in a vector and cycled through as the object reached each position in the vector. So that Moving Platforms could move with varying speeds, a speed factor was specified in the Path file. This was used as multiplier in calculating the velocity vector for each object.

```
Moving 9 X:314 Y:227 Z:-191 DX:5 DY:1 DZ:5 Control:0 Speed:.07 textures/door.jpg
Moving 9 X:314 Y:197 Z:-191 Speed:.07 textures/door.jpg
Moving 9 X:324 Y:197 Z:-191 Speed:.07 textures/door.jpg
Moving 9 X:324 Y:227 Z:-191 Speed:.07 textures/door.jpg
Moving 10 X:325 Y:233 Z:-210 DX:5 DY:1 DZ:85 Control:0 Speed:-.03 textures/door.jpg
Moving 10 X:325 Y:233 Z:-210 Speed:-.03 textures/door.jpg
Moving 11 X:325 Y:227 Z:-197 DX:5 DY:1 DZ:5 Control:1 Speed:.07 textures/door.jpg
Moving 12 X:330 Y:230 Z:-253 DX:5 DY:1 DZ:5 Control:0 Speed:.07 textures/door.jpg
Moving 12 X:330 Y:250 Z:-253 Speed:.07 textures/door.jpg
Moving 12 X:330 Y:250 Z:-259 Speed:.07 textures/door.jpg
Moving 12 X:330 Y:230 Z:-259 Speed:.07 textures/door.jpg
Robot 0 70 5.00 -94.00
Robot 0 70 5.00 -116.00
Robot 1 67 5.00 -94.00
Robot 1 67 5.00 -116.00
```

Code Figure 4 - Path File for Moving Objects

As a feature unique to Lume, moving platforms were also controllable by the player if he had enough energy to do so. This required a reference to the character in the Moving Object. The Moving Object stored a boolean for whether it was controllable

and if it was being controlled by the player. If this was the case, the player draw method referenced the object that was being controlled. This allowed for the camera to be drawn from the perspective of the moving object, and give the player the ability to move the object anywhere in the x-z plane.

Results

At the beginning of the project, the only things we had were a few Youtube videos to give us inspiration for the look and feel of the game, and some tools developed during the previous quarter. After two quarters, we were able to turn it into a playable game with a story and unique look and feel.

Our team was particularly proud of how we were able to take our limited graphics knowledge and use certain technology to create an appealing look. After completing a level, the world lights up (Figure 9). Robot enemies are destroyed when sprinting through them, creating a particle effect that surrounds the robot. (Figure 10) 3D models of trees are animated to be brought back to life (Figure 11). A bloom shader is used on top of the trees to give them a pretty glow (Figure 11). An excessive amount of blocks built on a single wall shows the creation and life that is brought to the game (Figure 12). The look of the blocks on a wall is very appealing and demonstrates the freedom the player has to change the world he or she is in. All of these technologies were things the team learned over two quarters, and are proud to have in the game.

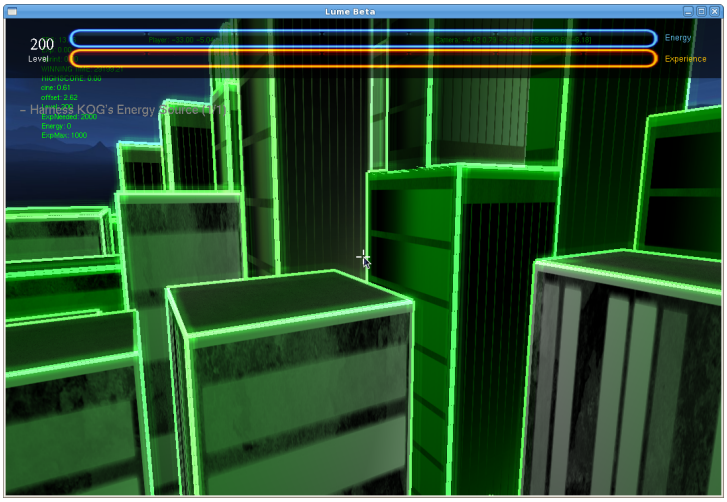

Figure 9 – Neon Colors

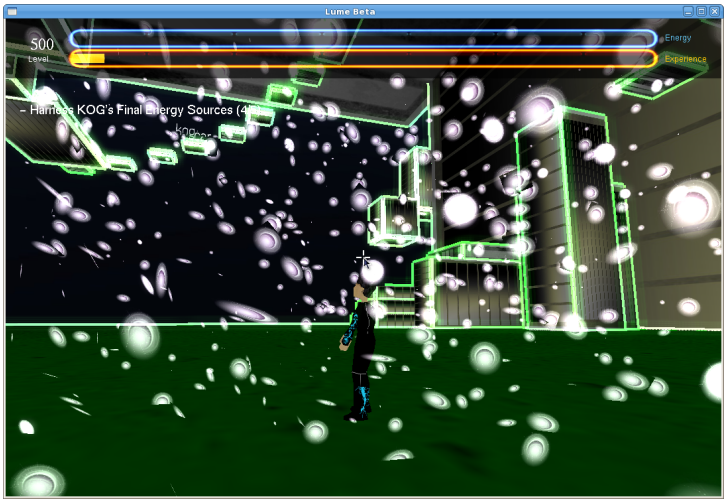

Figure 10 – Particle Effect

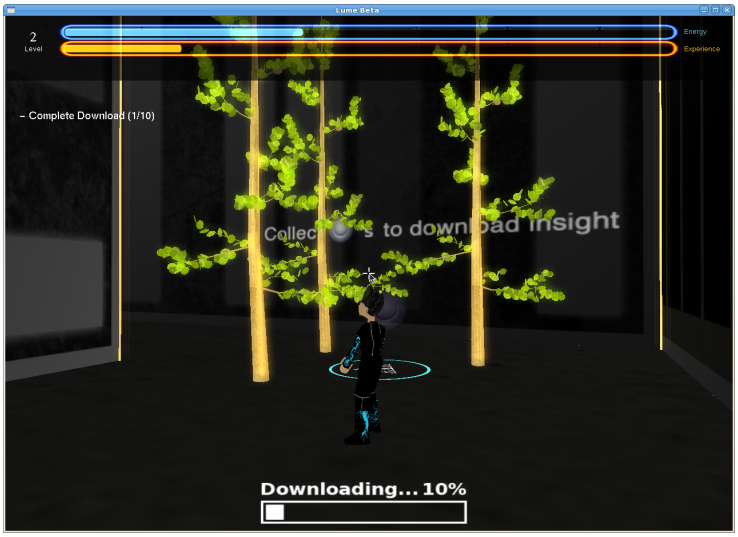

Figure 11 – Nature and Bloom

Figure 12 – Excess of Blocks

We were also happy that we were able to create a game with a simple game mechanic that gives the player the freedom to explore the world while still having set objectives that keeps the player interested. Below is a picture of the block that a player has built (Figure 13). The player can jump on the block to maneuver around the world. In the top left of the figure is the current objective the player must complete, with a progress bar towards completing the objective below.

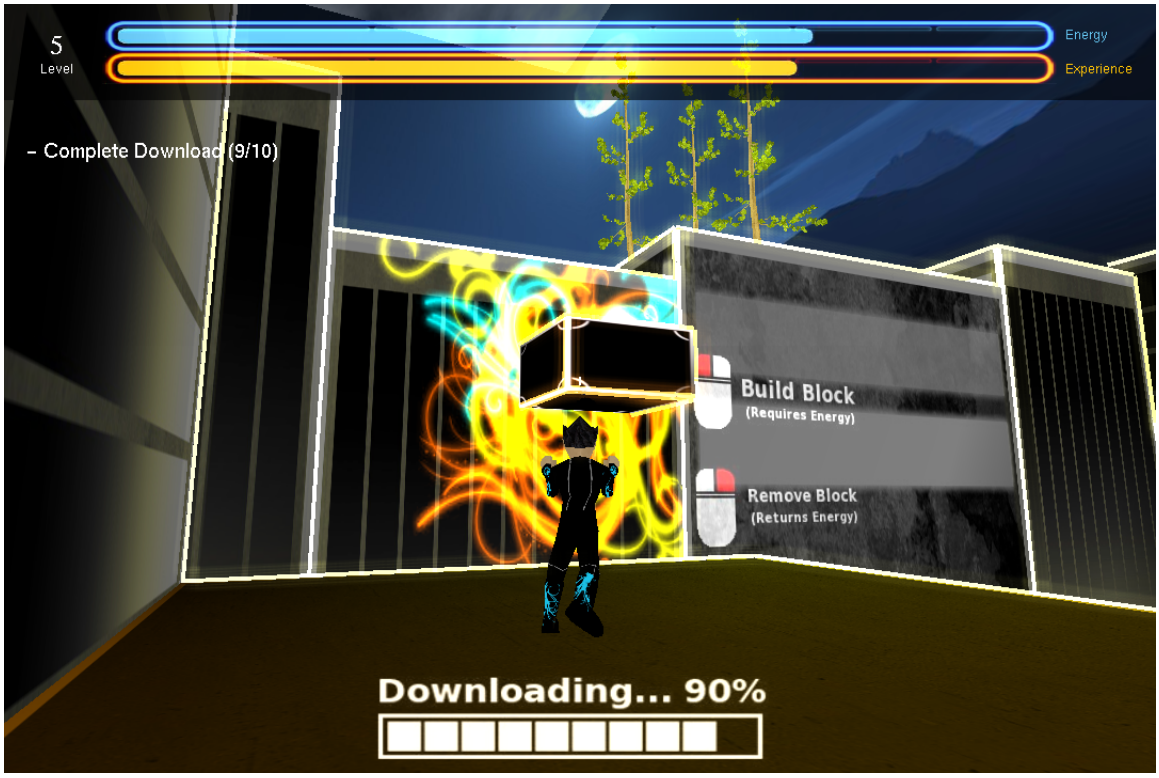

Figure 13 – Block Building

An important aspect of the development was the integration of play testing and the strong feedback that people gave as they played Lume. Included below are a couple player feedback forms that demonstrate the type of questions that were asked and what kind of responses play testers gave.

Play Tester	Comments
Joanne Mark (Week 15/20)	**Fun Level (1-10)** **1- Not fun / 10 - Totally awesome** 8 **Game Crash?** No **Laggy?** No **Bugs seen?** Can see through roof w/ camera Camera goes into the wall Ground causing death should be more clear

	Likable features • Finding / Gathering insight • The trees • The energy trails • The evolution from the growing objects • Outline glow of the buildings **Dislikes** • Without building outlines it is hard to tell distance • Didn't like the ramp to ramp jumps • Don't like the last part of suburbs with up & over needed to go up the levels, better if it was just going up **Suggestions** • Have a girl character too! • Have high score list (maybe have something similar to Zelda load screen where it shows stats) • Multiple saves for different players • Freebies - find hidden star to get extra energy / cannon shot / turn on moving walkways **Miscellaneous** Stopped at the back of the energy source building in the suburbs because of frustration
Brian Sukkar (Week 17/20)	**Fun Level (1-10)** **1- Not fun / 10 - Totally awesome** 6/7 **Game Crash?** No **Laggy?** at one point -> see bug section below (otherwise no lag) **Bugs seen?** • Died for no reason (probably lag + enemy shove) • Camera went all whack and couldn't see • Feet off edge but still on block • De synced moving platforms **Likable features** • Pretty

- Challenge is fun
- Different routes are good
- Difficult to control direction at first but felt good after a while
- Everything moving

Dislikes

- Jumping through blocks - it doesn't seem different enough once that feature is introduced

Additional Suggestions

- Fit trees more (but cool as is as well)
- Like the give life but maybe make purple? like Avatar - the movie

Miscellaneous
Maybe black hole or something to bring to beginning / teleport at top of level

Based upon earlier feedback from classmates and demonstration viewers, certain aspects of Lume were changed or expanded upon. One of the design aspects was to create a better way to show that the setting is a city. To accomplish this, billboards and logos were designed and integrated into the futuristic city (Figure 14).

Figure 14 – Logos in the City

Another aspect that was commented on was the lack of interaction with enemies or other characters. Users wanted another way to act against the robots of KOG. In addition to the "sprint through to destroy" interaction, the ability to "freeze" large robots was added, to give the character the ability to feel more powerful. This freezing ability is shown in Figure 15.

Figure 15 - Freezing Ability

This experience gave the development team invaluable insight into what it is like to make a game on a team. Each technology that was developed had a unique way to make the game better, and we used each team member's strengths to make a positive contribution to the game. The game would not be the same without the effort of each team member. The tasks that each person completed were motivated by decisions made about game mechanics, aesthetic appeal, story, technology, and play tester opinions.

The game testing started in the later stages of our game development, so the game mechanics and most of the look feel had been determined by this time. The major contribution that game testers gave were comments that clarified the story. Some players were confused at what the story was, and we later created cut scenes at the beginning and end of the game to clear up the story line. Game testers also said they felt lost at what to do in the game because it of their ability to roam freely. To address this, we created clear objectives in the game that gave the player an idea of what they are supposed to do in each level. We were also constantly modifying individual levels in the game to have a linear amount of difficulty as a player progresses through the game.

Dividing the work was done based on each member's strengths or what they were particularly interested in. Allowing everyone to choose what they were interested in resulted in a more rapid development because each member was eager to learn about the technology involved. Small teams were assigned to different tasks in order to prevent anyone from working alone. These teams made up of two or more members allowed for more monitoring and motivation for each member.

Working alone as oppose to working in small teams was shown to be inefficient during the two quarter process. Deciding on design decisions without the input of at least one other member was likely to result in individual error. Furthermore, code was harder to interpret and errors were harder to debug when other members looking at it were not directly involved. Overall, our development process can be described as allowing small teams to rapidly develop technologies that they were

most interested in, while in a manner that utilized each team member's strengths. This was proven to be successful as is indicated by the progress of game over these last two quarters.

Conclusion

Creating a video game was a very extensive process that taught me many things in addition to the specifics mentioned in the Algorithms section. When I started the project, I had a very limited knowledge of the graphics pipeline, and did not understand the purpose of learning more efficient techniques for drawing objects on the screen. This stemmed from my naïve belief that complex graphical optimizations were not necessary since I did not see any problems with the programs I was writing at the time. During the development of Lume, it became apparent that optimizing the pipeline was vital to the game running at 30-60 frames per second. Techniques such as view frustum culling and dividing the game's world into a spatial data structure were things I learned how to implement. After including them in the game, I witnessed the effect they had and learned the importance of efficiency in real time rendering.

Another learning experience I had was in regards to working on a large-scale software project with a team. Although Software Engineering courses taught me about this, learning about it through the development of a video game was an entirely different situation. I learned how important it was for the team to be open-minded about the details of the game. It was crucial that we were not "married" to any specific ideas so that the game could be developed with an iterative approach.

This included weekly updates to the story, slight tweaks in game mechanics, and changes to visuals. What I found to be beneficial was having a clear thematic vision for our game, which was Tron. No matter how far we deviated from original storylines or game mechanics, having a clear theme in mind allowed for the team to make the game feel cohesive and complete.

The biggest thing I've taken away from this project is the ability to see the impact of the work that the team has done on anyone who watches or plays the game. Being able to have a tangible product has allowed for instant feedback and gives people reason to pay attention to advances in computer science. The game ties together many of the concepts that I have learned about over the past three years, and allows for me to feel a sense of reward for the hours put into the project. It has compelled me to learn more about graphics in the future, and has given me something to be proud of in the present.

Future Work

After two quarters of work, Lume has become a complete interactive 3D game. The next stage, should the team choose to pursue it, would be preparing Lume for release on Steam. The following aspects of Lume would need to be completed in order to confidently release Lume on Steam for users around the world to play.

Story

Currently, Lume has a complete storyline which is not conveyed clearly enough through playing the game. To improve upon this, the team would need to

further develop the storyline by including more artistic cut scenes and more objectives that add purpose to the player's choices. Interactive communication with Insight, so that the player can ask questions, would also expand upon game mechanics and story elements.

Memory Management

None of the classes in Lume have taken full advantage of destructors in C++. In addition to running a profiler to determine which methods can be optimized for better performance, the implementation of destructors for all of our classes would greatly improve the performance of Lume on older machines.

Portability

The current system specifications for Lume are 32-bit Linux operating systems. In order for our game to have the impact we wish for it to have, it must be modified to be playable on Windows and Mac systems. This requires some redesign of the code, most notably in the included libraries in each header file and the Makefile. Some function calls are specific to the operating system (namely Windows) and would need to be changed. This would allow us to distribute our game on Steam for any operating system

References

Call of Duty Gradiant Map Editor Tutorials –
http://callofduty.filefront.com/info/Tut_Map_1_Surgeon

C – File Writing –
http://www.phanderson.com/files/file_write.html

C Plus Plus Classes –
http://www.cplusplus.com/

Lighthouse Tutorials –
http://www.lighthouse3d.com/tutorials/

Swiftless Tutorial – Heads Up Display -
http://www.swiftless.com/tutorials/opengl/orthogonal.html