LUME

Senior Project Write-Up

By Teal Owyang
and project team members, Mike Buerli, Brent Dimapilis,
Trent Ellingsen, Jeff Good, Jonathan Rawson and Ryan Schroeder

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 alike 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 story-lines 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 around a creation and sandbox feel intended to give the user a fully immersing adventure experience. Players are encouraged to build upon the world in order to reach new objectives, 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 providing the player a strong sense of influence over their surroundings. Throughout the course of the game, the player gains more abilities through leveling up including: 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 possesses 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 simply by building 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. To be brief, the story is of a future civilization, which is under the complete control of KOG, who's only hope of survival is Lume (the player). The story takes on the classic battle of humanity versus technology (not evident until the end of the game) as well as nature versus industry. The story starts out in the year 4200, where KOG, a privatized company now controls all civilization. All of earths resources have been depleted and the world is now solely based on energy. The intro scene for Lume starts with an image of a door labeled Project Lume. Project Lume is an experiment conducted by KOG to create 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 energy. A computer terminal is seen activating a program called Insight. Insight is a computer AI designed back in 3119 at the creation of KOG, designed 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 intro shows Lume falling from the sky (down from the upper KOG city) and then awakening in the tutorial level where the game begins. 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 the energy is collected you can reach the upper city, where the player is given the opportunity to finally defeat KOG. After completing the game, it is revealed that KOG stands for Komputer Organized Government and that KOG is a Komputer (next gen computer). By shutting KOG down, all energy is relinquished back to the earth, allowing humanity and nature to start once again.

Related Works

The general look and feel for Lume was inspired by the movie Tron: Legacy by Disney displayed in Figure 1 below. 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 focused on adding new innovations to the 3D platformer genre. Lume's game-play 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 displayed in 2 below and Assassin's Creed displayed in Figure 3. The inspiration behind completing various objectives in the different KOG districts stems from the star collection mechanic of Super Mario 64 displayed in 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 build on nearly every building 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 anywhere in the world. Allowing the player the change the world freely grants them the ability to form their own path and visually alter a level differently on each play through.

Figure 2: Mirror's Edge

Figure 3: Assassin's Creed

Figure 4: Super Mario 64

Technologies

Lume was implemented with a variety of graphics technologies that allowed the game to have an aesthetically pleasing look, while running in an efficient manner.

Below is a list of the various technologies that were implemented and team member(s) that worked on them:

- 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)

Look & 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 stand alone 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.

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. 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.

Getting Animated 3D Models into an OpenGL World

In our game, we required 3D models to represent important aspects of our game including various world objects, robot enemies, and the protagonist of our game, Lume. Some of these 3D models had to support animation. These 3D models made our world look more believable and gave the world more life through animation and other various modeling techniques. There was a small pipeline dedicated to getting these 3D models in our game. First, the 3D model must be either imported or modeled in a 3D modeling application, such as Maya or Blender. Next, the model had to be exported into an MD5 mesh and anim file format. This format is then

read to be displayed in the game. Finally, the logic had be set to run the animations correctly. When all this is done, a 3D model with animation is in the game. I will describe the full details of each of these steps down below.

Modeling Process

The models were generated in two different 3D modeling applications called Maya 2010 and Blender 2.49b. Both applications have the capability to create 3D models without animation and with animation. All the models had to be constructed with triangulated polygons; this means no NURBs or volumetric shapes. Another member that worked on this project, Jeff Good, was mostly in charge of getting 3D models in our game without animations. I created some models without animation, but I mostly worked on getting animated models in the game. Below, I will describe how the animated models were created for our game.

Modeling

The models that are animated in our game are the protagonist and one of the robots. Although I created the first robot initially in our game, the animated robot that made it into the game was modeled by Brent Dimapilis. Brent and I worked together to get the robot in the game with this same pipeline. I will be focusing mostly on the main character when describing how to get 3D animated models in our game.

The main character model was given thanks to Bill Hess and the Journey In The Dark team from a previous CSC 476 class. You can see the character below in Figure 1. The model was in an MD5 format with a few animation files as well. I used a Python script found online to import the character model and skeletal mesh into Blender [3].

The model had to be finalized in Blender first. In one of our demonstrations during game development, some people were familiar with the model given by Bill Hess, and desired a unique model for our game. In response, I modified the character mesh in Blender to give a new look more fit for the game. This final character mesh is shown in Figure 2. This was done by changing the character mesh and UV mapping the character.

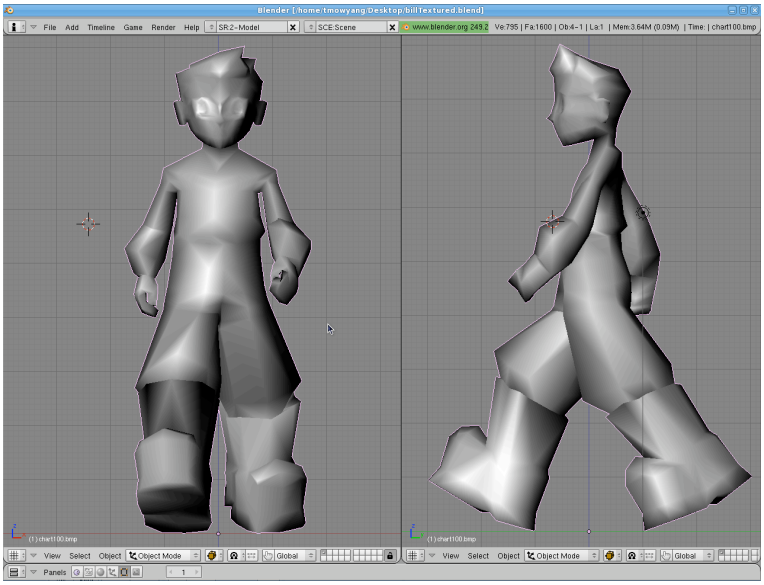

***Figure 1**: Original Character Model*

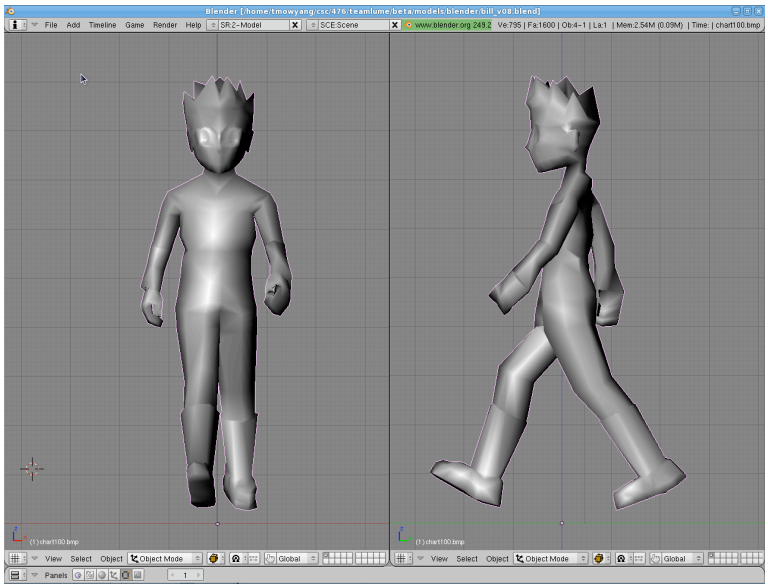

***Figure 2**: New Character Model*

UV Mapping

UV mapping is the process of making a 2D image representation of a 3D model. Our model was UV mapped in Blender. By creating what is called seams in your 3D model, you can unwrap your 3D model into a 2D image. Blender exports a .tga image of the unwrapped character, and the image is then used in Photoshop to create a separate image that will go back on to of the 3D model. Team member, Trent Ellingsen, was responsible for generating the character texture. You can see the 2D images below, and in Figure 5, you can see the new texture wrapped around the 3D model.

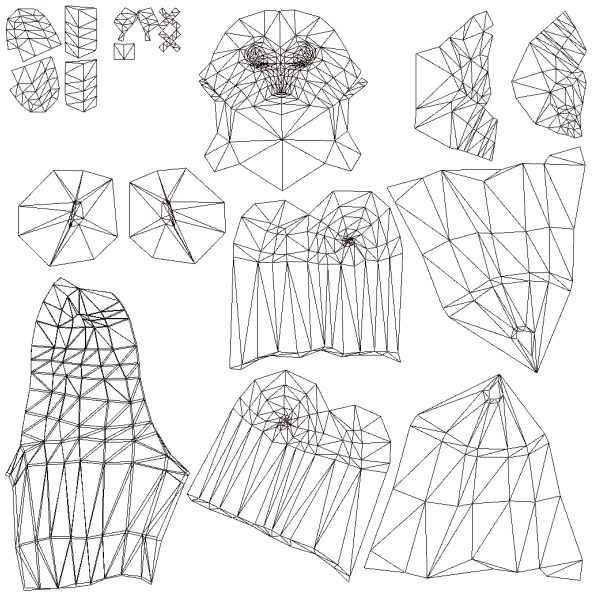

***Figure 3**: TGA Unwrapped Character*

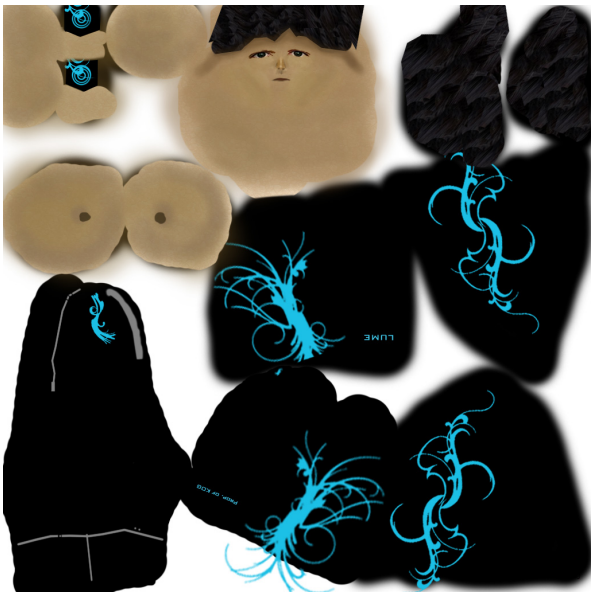

***Figure 4**: New Texture Based on TGA*

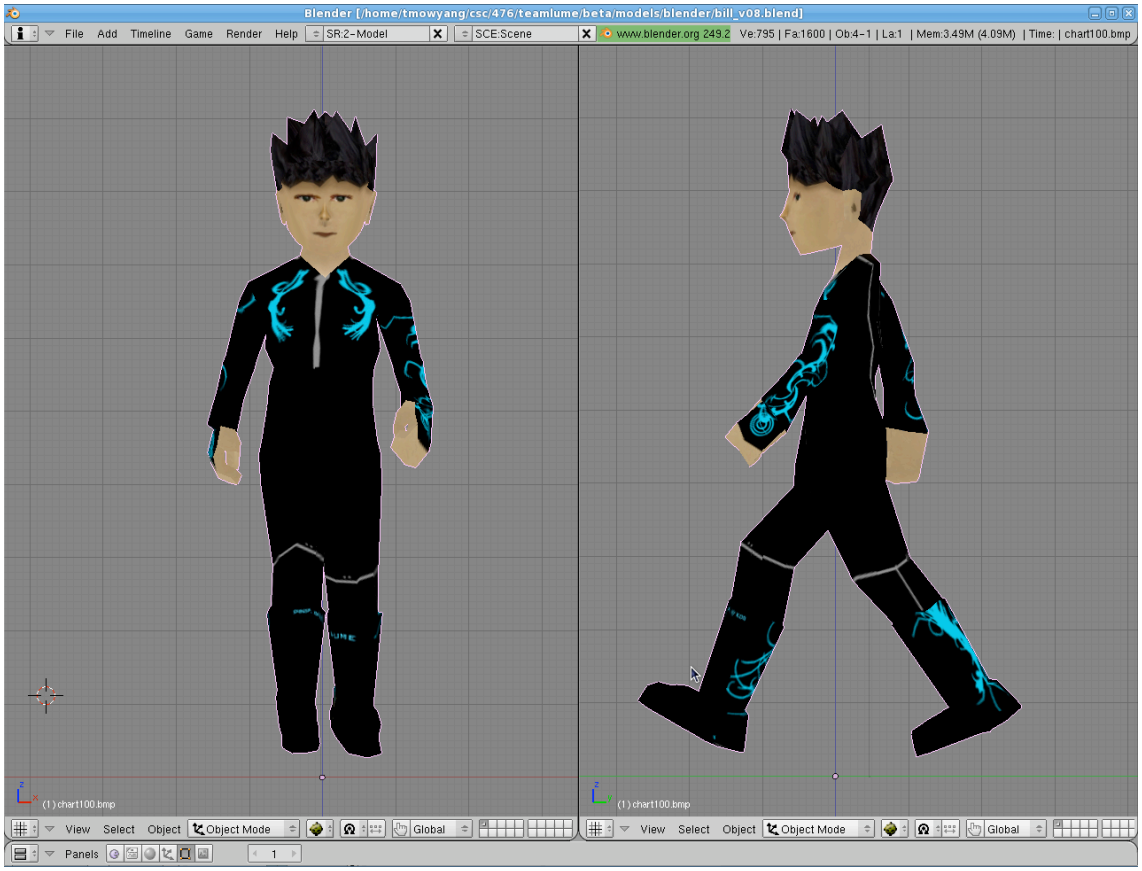

Figure 5: *Textured Character*

Importing 3D Model

After completing the model, the model is exported with a Blender to MD5 exporter [3]. Once the MD5 files are accessible, the final step is importing the model into our game. My main resource for importing the model into the game was David Henry's tutorial [1]. MD5 contains two different types of files; MD5 mesh and MD5 anim files. The MD5 mesh contains the model's bind-pose skeleton and potentially multiple meshes. In this game, each MD5 mesh file contains only one mesh for simplicity. The mesh data in the file includes:

- Vertices
- Triangles
- Vertex weights
- Shader name

When reading the MD5 file, the 3D vertices are all read into an array and then the index of the array is used when reading in the triangle faces. Each triangle has three index values that create a triangle face. The vertex weights are used to compute the vertex final position based around the joint it is bound to. Finally the shader name, which is not used in our game. Normally, it would hold a file that tells what textures to apply to the mesh and how to combine them. Since the game use UV mapping, this was unnecessary.

The MD5 anim file contains all things related to animating the model. It stores information on:

- Skeletal hierarchy with flags for each joint
- A base skeleton from which the animated skeleton is computed
- A list of frames that contain data to compute the skeleton position based off the base skeleton

When reading the MD5 anim file, the baseframe data is read like the text below.

```
baseframe {
    ( pos.x pos.y pos.z ) ( orient.x orient.y orient.z )
    ...
}
```

It contains the position and orientation of each joint that will create the base skeleton. This data will later be used for moving the skeleton and thus animating the model. This will be discussed in more detail in the section below on the skeletal mesh.

Next is the frame data that contain parameters used to compute the position of the skeleton for each frame. In the MD5 file, this information is stored like the text below.

```
frame frameIndex {
    <float> <float> <float> ...
}
```

After collecting all the data for each frame, the skeleton position can be computed.

Skeletal Mesh

The character had to have a skeleton that was bound with the mesh. Skeletal meshes are necessary for character animation. When the skeleton attached to the model moves, the model moves with the bones and joints. If there weren't a skeleton mesh, the only way to animate a character is by drawing a new model in different positions for every frame. This effectively what 2D animation does. In a real-time game, the resources required to do something like that is impractical. But with a skeletal mesh, it can be practical to have an animated model in real-time.

This is thanks to what is called quaternions. Quaternions are a way of representing rotations without matrices. Quaternions contain rotational data and not position. This is the same thing as the MD5's orientation data for each joint. With quaternions and a skeletal mesh, animation can be determined without actually calculating the position change of every vertex!

To recap what happened when importing the model, each frame read in has it's own skeleton with position and orientation data for each joint in that skeleton. Moving the skeleton from one frame to the next requires what is called skeletal interpolation. There are two types of interpolations that use the position and orientation data read from the MD5 anim file. There is *linear interpolation* for the skeleton position and *spherical linear interpolation* for skeleton orientation.

Linear interpolation finds the position between two points. It is done for the model by the code block below:

```
finalJoint->pos.x =
            jointA->pos.x + interp * (jointB->pos.x - jointA->pos.x);

finalJoint->pos.y =
            jointA->pos.y + interp * (jointB->pos.y - jointA->pos.y);

finalJoint->pos.z =
            jointA->pos.z + interp * (jointB->pos.z - jointA->pos.z);
```

`interp` is a value between 0 and 1 that represents how far between `jointA` and `jointB` the `finalJoint` should be at. If `interp` is equal to 0, the `finalJoint` position would be at the position of `jointA`, and if `interp` is equal to 1, the `finalJoint` position would be at the position of `jointB`.

Spherical linear interpolation is another form of linear interpolation, where instead of interpolating along a straight line, it rotationally interpolates from one point on a sphere to another. In Figure 6 below, p0 would be the first frame of the joint orientation and p1 would be the next frame.

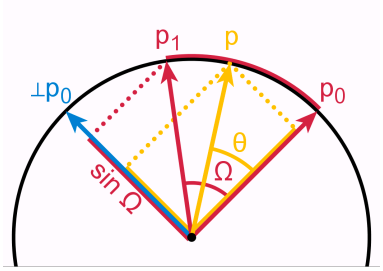

Figure 6: Spherical Linear Interpolation

Below is the geometric formula for spherical linear interpolation. p0 and p1 are the same as in Figure 6. t represents the same 0 to 1 value as `interp` did in regular linear interpolation.

Compute Ω as the angle subtended by the arc, so that cos Ω = $p_0 \cdot p_1$

$$\mathrm{Slerp}(p_0, p_1; t) = \frac{\sin\left[(1-t)\Omega\right]}{\sin\Omega} p_0 + \frac{\sin[t\Omega]}{\sin\Omega} p_1.$$

This formula will output a 3D point between p0 and p1.

Animating the Model

To animate the model, the skeletal position for the current and next frame is computed, and then skeletal interpolation smoothly moves the skeleton from one frame to the next. This is all finally drawn into the scene and there is finally an animated character in the game.

Results

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

Our team was particularly proud of how we were able to take our limited graphics knowledge, and creatively use technology to create an appealing look. After completing a level, the world lights up (Figure 5). Robot enemies are destroyed when sprinting through them, creating a particle effect that surrounds the robot. (Figure 6) 3D models of trees are animated to be brought back to life (Figure 7). A bloom shader is used on top of the trees to give them a pretty glow (Figure 7). An excessive amount of blocks built on a single wall shows the creation and life

that is brought to the game (Figure 8). 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.

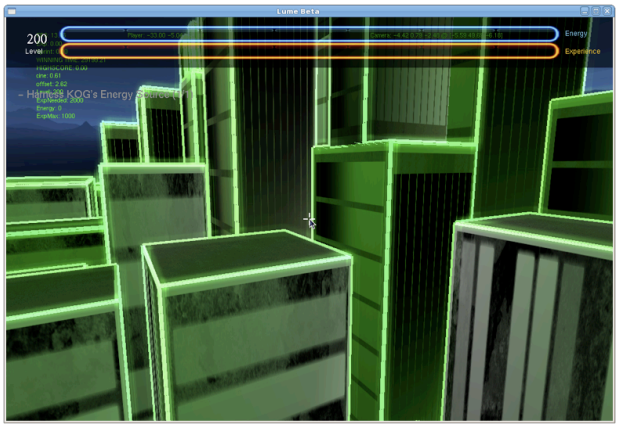

Figure 5

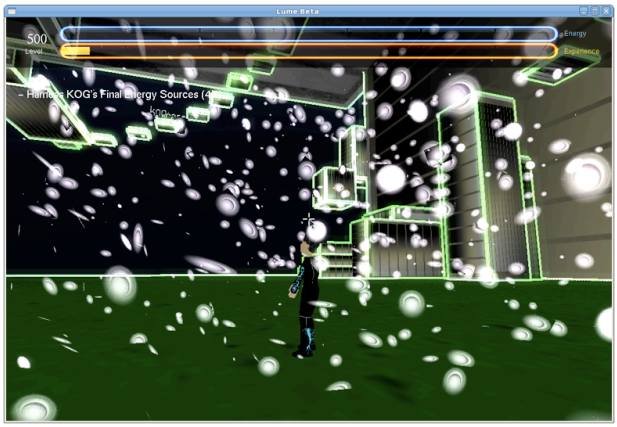

Figure 6

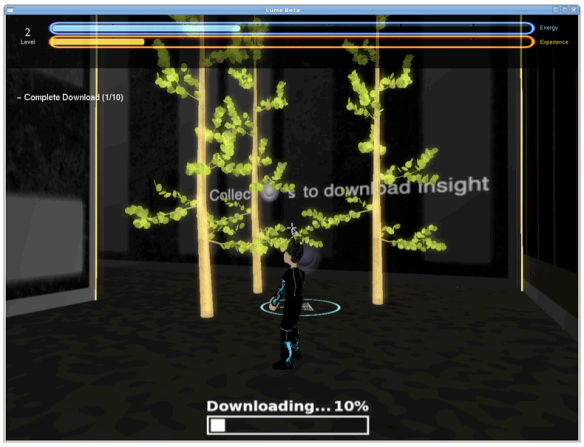

Figure 7

Figure 8

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 9). 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.

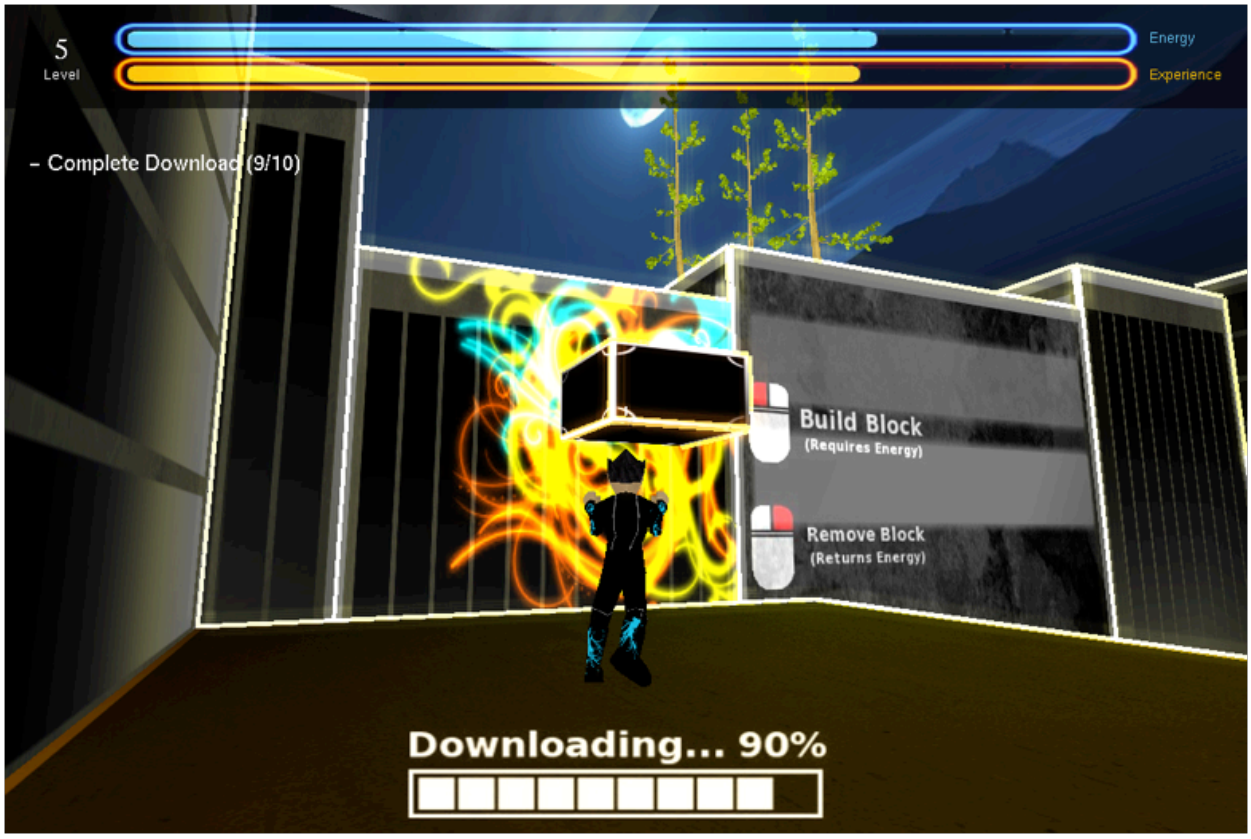

Figure 9

Play Tester	Comments
Joanne Mark Week 15 / 20	**Fun Level (1-10)** **1- Not fun / 10 - Totally awesome** 8 **Bugs seen?** Yes : 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 gobj • Outline glow of the buildings • The music **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 • Need more check points **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

Play Tester	Comments
Brian Sukkar Week 17 / 20	**Fun Level (1-10)** **1- Not fun / 10 - Totally awesome** 6/7 **Bugs seen?** • Died for no reason (probably lag + enemy shove) • Camera went all wack 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** • Make controls more clear (maybe cut scene?) • When first enemy appears tell about sprint to kill them • Change level 1 block that is floating • Full animation • Suburb if go wrong way on first part then hard to go back. • 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

Play Tester	Comments
Ken Li Week 17 / 20	**Fun Level (1-10)** **1- not fun / 10 - totally awesome** 5 **Bugs seen?** • Leveled up then able to take all blocks back • Bad picking sometimes (character against wall trying to remove block placed to the side) **Likable features** • Graphics • Colors of making blocks • Likes building lighting up when close • First level well made **Dislikes** • Frustrated • Check point so far back • Save feature needs explanation **Additional Suggestions** need 'e' explained suburbs not well built likes if there is narration Would like some sort of gun that shoots the blocks or some indication that he has the power **Miscellaneous** Stopped out of frustration on bottom of suburbs tower. Took portal into first, then left to unlock the other worlds.

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 10).

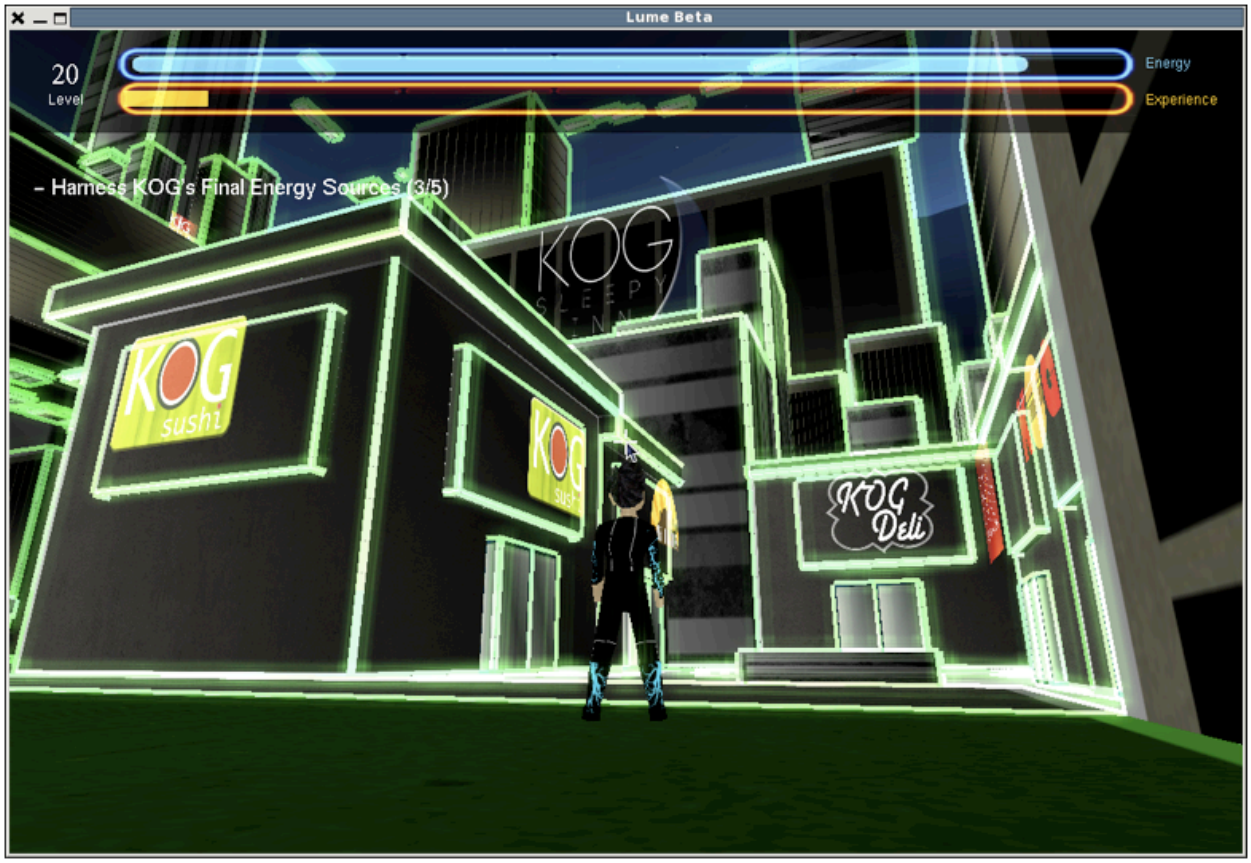

Figure 10: Logos in City

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 11.

Figure 11: 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. Our game was played by students from Cal Poly, including computer science masters students concentrating in computer graphics. Having fresh eyes playing our game was a great help. The play testers were able to convey what didn't feel right in the game, explain what they liked about the game, and describe what was not clear in our game. Fixing these problems or expanding upon things that people enjoyed allowed for the team to make a game focused on the player's desires.

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.

Future Work

After two quarters of work, Lume has become a fully fleshed out 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.

More In-Depth Storyline

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.

Conclusion

This project was an extremely rewarding experience. This has been the largest and longest team project I have taken on. It not only gave me great insight on the technical process of making a game, but it taught me valuable lessons on how to work on a team. This project put our team into situations where we had to decide on what we could reasonable finish by our milestones, and prioritize what was more important to complete. There were points in the earlier stages of game development where it was a real challenge to let ourselves stop for a moment from thinking about the next technology, and really think about look and feel and game mechanics.

In this process, I was able to learn about my teammates' and my own strengths and weaknesses. I was able learn how each person works most effectively, and give out tasks for teammates to accomplish based on how they work. Each person was different, and it was a great experience to learn how to get work done on such a large team.

References

[1] Henry, David. "MD5Mesh and MD5Anim Files Formats Specifications (Doom 3's Models)." *Tfc.duke.free.fr*. 21 Aug. 2005. Web. <http://tfc.duke.free.fr/coding/md5-specs-en.html>. Main source for writing the MD5 Importer

[2] Hess, Bill, Josh, Wesley, Brian, Adam, and Alex. "Journey In The Dark." *Journey In The Dark*. Web. 18 Apr. 2011. <http://jitd.blogspot.com/>. Previous CPE 476 game where model was taken from.

[3] "MD5, MD3, ASE Import Export Scripts, 3D Modelling Tools : KatsBits UTILITIES." *Blender Tutorials and Training with KatsBits.com*. Web. <http://www.katsbits.com/tools/>. MD5 to Blender Importer and Blender to MD5 Exporter Python scripts

[4] "Getting Started Maya 2011." Autodesk Inc., 2010. Web. <http://images.autodesk.com/adsk/files/gettingstartedmaya2011.pdf>. Used for modeling in Maya

[5] "YouTube - Texturing Part1." *YouTube - Broadcast Yourself.* Web. <http://www.youtube.com/watch?v=Bhp2ihqnJIk>. Tutorial used for UV mapping Blender models

[6] "YouTube - Blender Tutorial Series - Part 13 - UV Mapping." *YouTube - Broadcast Yourself.* Web. <http://www.youtube.com/watch?v=e5_2seDBQrw>. Tutorial used for UV mapping Blender models