LUME

Senior Project General Write-Up By Mike Buerli, Brent Dimapilis, Trent Ellingsen, Jeff Good, Teal Owyang, Jonathan Rawson, Ryan Schroeder

Senior Project Specific Sections Write-Up By Trent Ellingsen

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.

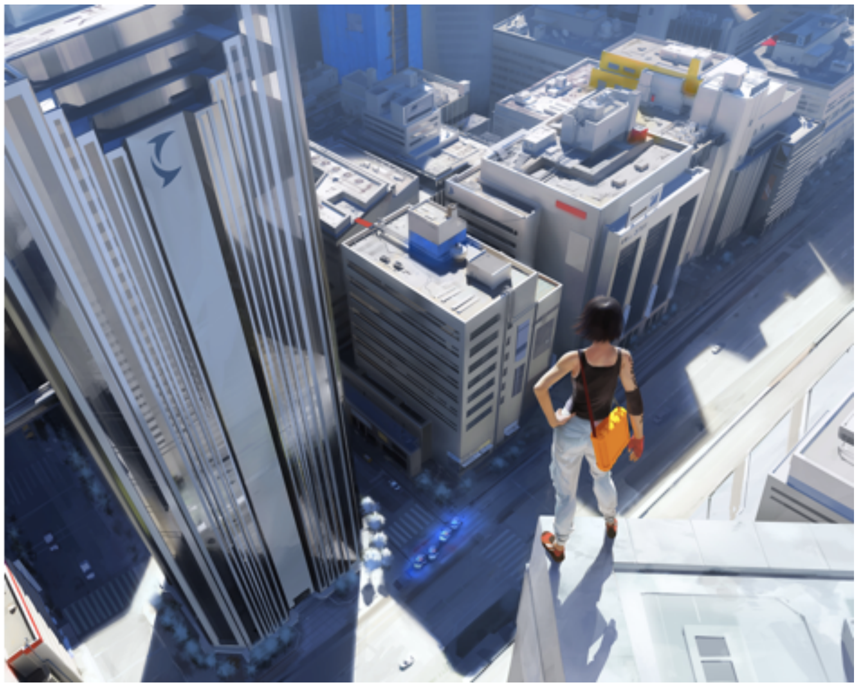
Figure 2: Mirror's Edge

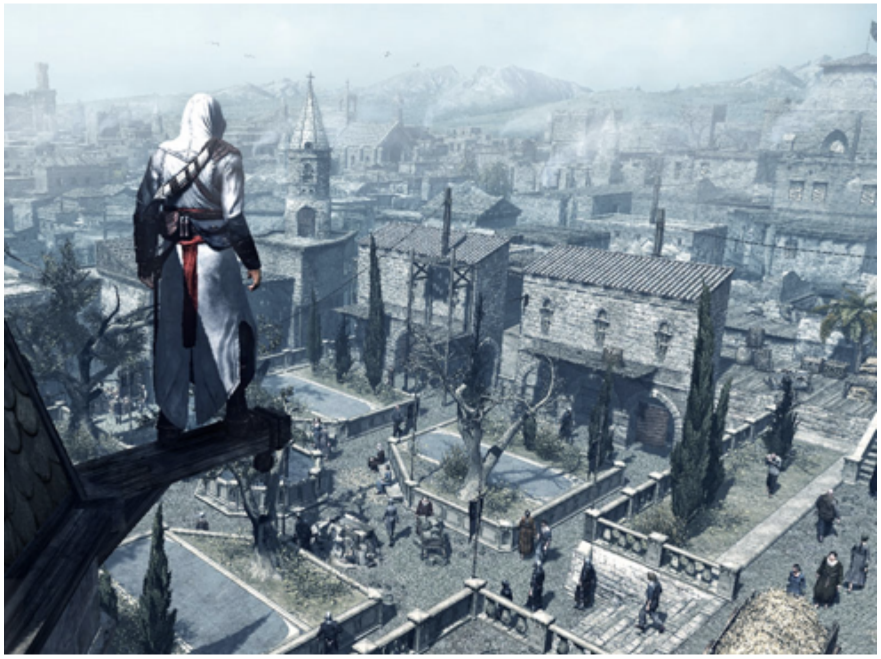
Figure 3: Assassin's Creed

Figure 4: Mario 64

General 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)

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.

Specific Technologies

The following labeled sections are the technologies and aspects to the Lume game that I, Trent Ellingsen, contributed and will be explaining. These detailed descriptions should act as guide and initial step in creating similar aspects for future projects.

Lume Particle System Explosions - Trent Ellingsen

One of the aspects of the Lume game is the interaction the player has with the robots. The 3 ways that the character can interact with the robots in the world are

- being hit and knocked off of the platforms in the world
- freezing the robots, making it so that they cannot move
- sprinting through robots, causing them to be removed from the game

To create a feeling that the player is hitting the robots, an explosion particle system is generated whenever the robots are hit. The particle explosion effect can be seen in Figure 5 shown below.

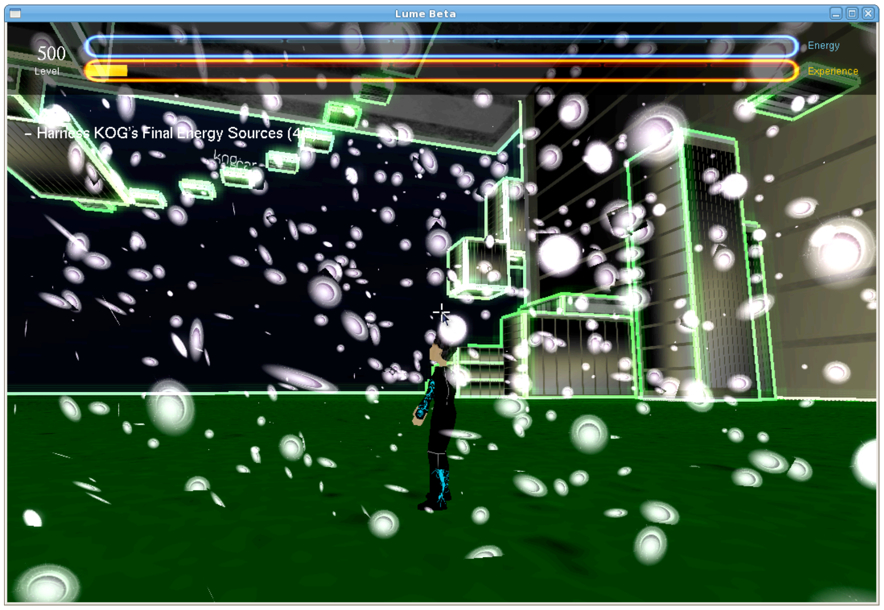

Figure 5: Demonstration of Particle Explosion

Why Particle Systems are Exciting

Particle systems add a major visual feel for games, movies, and in other 3D visual mediums. Particle systems are the driving force behind creating simulated smoke, fire, hair, water, dust, explosions, and other special visual effects. Once the concept for particle systems are understood the ability to add and expand upon it is made easy. The following description will detail and step through the process of creating a simple particle system that is easy to follow.

Particles

This section will describe the general concept of particle systems and what is required to implement one in the way that has been done for the Lume explosion effect.

Elements That Make Up a Particle

A particle system is a large amount of individual pieces acting together to create a single look and feel. To allow for the system to function, each particle in the system requires certain attributes to function correctly and give the ability for manipulability and individual movements. Here is a list of all the elements that make up a particle for Lume:

Representation in general:

- Position Vector
- Velocity Vector
- Acceleration Vector
- Force Vector
- Mass
- Gravity
- RGB value
- lifetime
- size
- change in time (dt)
- Object to be drawn

Representation in C++ (Code snippet):

```cpp
class Particle: public obj {
public:

   // Constructors and method declarations done here
   // pnt3d is a class used to represent vectors

   pnt3d _pos;
   pnt3d _vel;
   pnt3d _acc;
   pnt3d _force;
   float _mass;
   float _gravity;
   float _rgb[3];
   float _lifetime;
   float _size;
   float dt;
   obj2d o;

};
```

Particle elements explained

Each aspect of a particle is needed to create the desired effect for our explosion. The `position` vector is used to determine the current position in the 3D environment that the particle occupies. The `velocity` vector determines the speed and direction at which the particle is currently moving within the world. The `acceleration` vector determines the speed and direction at which the velocity is changing. The `force` is used to determine the acceleration which effects the movement of the particle. The `mass` and `gravity` of the particles are also used in conjunction with the force to determine the acceleration of the particle. The `lifetime` is a value that is used to determine the amount of time that the particle is being drawn in the environment. `Size` is used to set the physical size of the drawn particle. The change in time `dt` that is used to ultimately determine the position. Finally the `object` is used to draw the desired object for the particle.

Physics

To create the particle system, the particles contain motion that are based upon certain physics equations that change the position of the particle's location. The three main physics equations used in Lume's implementation are

General Form:

- new_acceleration = force / mass
- new_velocity = current_velocity + (current_acceleration * dt)
- new_position = current_position + (current_velocity * dt) + (0.5 * current_acceleration * dt * dt)

C++ Implementation Form:

```
/**
 * Physics equation to find the current acceleration.
 */
void Particle::calcAcc() {
   // Done so that floating point exception does not occur
   if (_mass == 0) {
      _acc = pnt3d(0, 0, 0);
   }
   else {
      _acc.x = _force.x / _mass;
      _acc.y = _force.y / _mass;
      _acc.z = _force.z / _mass;
   }
}

/**
 * Physics equation to find the current velocity.
 */
void Particle::calcVel() {
   _vel.x = _vel.x + _acc.x * dt;
   _vel.y = _vel.y + _acc.y * dt;
   _vel.z = _vel.z + _acc.z * dt;
}

/**
 * Physics equation to find the current position.
 */
void Particle::calcPos() {
   _pos.x = _pos.x + _vel.x * dt + 0.5 * _acc.x * dt * dt;
   _pos.y = _pos.y + _vel.y * dt + 0.5 * _acc.y * dt * dt;
   _pos.z = _pos.z + _vel.z * dt + 0.5 * _acc.z * dt * dt;
}
```

Update Function

Every frame of the game the position of all the particles must be updated so that they are in the correct location based on their velocity and have all the other correct properties based on the other aspects of the particle. Only a few things must be done in the particle update function for it to meet all the specifications. The dt variable needs to be passed in, velocity and position needs to be calculated, the lifetime variable is decremented and the size of the particle is shrunk based upon the initial size and lifetime of the particle. Below is the code segment implemented to do these tasks in Lume:

```
/**
 * Updates the position of a single particle.
 */
void Particle::update(float dt) {
   this->dt = dt;
   //calcAcc(); //Turn on if Force changes during animation
   calcVel();
   calcPos();
   _lifetime--;
   _size -= init_size/init_lifetime;
}
```

Note how calculating the acceleration is commented out. This has been done to optimize the code. Since the forces, mass, and gravity of the particles do not change for the explosion effect for the particles, acceleration does not change. The acceleration of the particles is thus calculated only inside the constructor. If the force were to change in some way (i.e. if the particles collided with objects) then a flag could set the calculation of the new acceleration when the forces change so that the code is still optimized to the fullest.

Drawing a Particle

When drawing the particle a few things must be considered when drawing. The color, size and position must be accounted for and then the object can be properly drawn. The Lume implemented code shows the draw method.

```
/**
 * Draws a single particle
 */
void Particle::draw() {
   glPushMatrix();
      glColor3f(_rgb[0], _rgb[1], _rgb[2]);
      glTranslatef(_pos.x, _pos.y + 1, _pos.z);
      glScalef(_size, _size, _size);
      o.draw();
   glPopMatrix();
}
```

Particle Systems

This section will describe how a system of particles is implemented with the use of many individual particles.

Elements That Make Up a Particle System

To create the explosion particle effect that is used in Lume for the robot deletion a few aspects are needed to be defined. Here are the different elements of the particle system:

General elements:

- Source position vector
- Collection of particles
- Integer representing amount of particles in system

C++ definition of elements:

```
class ParticleSystem: public obj {
public:

   // Constructors and method declarations done here
   // pnt3d is a class used to represent vectors

   pnt3d _source;
   std::vector<Particle*> particles;
   int _num_p;

};
```

Creating a System

To create the system of particles used for the explosion, the ParticleSystem class runs a createSystem method that creates _num_p number of particles. This method is shown below.

```
/**
 * Creates a default system of particles. Called in the constructor.
 */
void ParticleSystem::createSystem(char *tex) {
   for (int i = 0; i < _num_p; i++) {
      particles.push_back(new Particle(tex));
   }
}
```

The `new Particle(tex)` uses a specific constructor of the Particle class for the purpose of creating the explosion particle system. The specialized constructor randomizes many of the elements of the particles to chosen specification. The `position` is randomized by a factor of up to 1 unit away from the source in any direction. The `velocity` is randomized to be initially a speed of 0 - 0.25 in any direction. To create the effect that the particles slow down near the end of their life, the `force` is initialized to be the negative of the initial velocity. `Mass` is set to 300 to create the best looking speed and slowdown of the particles. `Color` is set to be a gray value between 0, 0, 0 (black) and 0.15, 0.15, 0.15 (dark gray). `Size` is a random float between 1 - 5 and `lifetime` is between 30 and 80 iterations of the update function. Finally the object is set to be a 2D object with the specified texture. The segment of code is shown below.

```
/**
* Used to make a default Particle with specified texture.
*/
Particle::Particle(char *tex) {
  classId = OBJ_ID;
  _pos = pnt3d(randomizer(1), randomizer(3), randomizer(1));
  _vel = pnt3d(randomizer(1)/25, randomizer(1)/25, randomizer(1)/25);
  _force = pnt3d(-_vel.x, -_vel.y, -_vel.z);
  _mass = 300;
  calcAcc();
  _rgb[0] = _rgb[1] = _rgb[2] = randomizer(0, 15)/100;
  _size = init_size = randomizer(1, 5);
  _lifetime = init_lifetime = randomizer(30, 80);
  o = obj2d(tex);
}
```

Updating the Particle System

Since all the particles of the system have update functions of their own that change the positions of the individual positions, the only requirement for the update function for the system is simply to iterate through and call each particle update function. The update function for the ParticleSystem class is shown below.

```
/**
 * Updates all particles in system.
 */
void ParticleSystem::update(float dt) {
   for (int i = 0; i < _num_p; i++) {
      particles[i]->update(dt);
   }
}
```

Drawing the Particle System

The final process in creating the explosion particle system that is found in Lume is to draw the system. There are a few things to do in the draw method. First the system checks if there are any particles to be drawn. Next it iterates through all available particles and if the lifetime of the particle is greater than 0 then it translates it to the source of the system and draws the particle. If the lifetime of the particle is not greater than 0, then the particle is removed from the system. The way in which Lume implements this is shown below.

```
/**
 * Draws all particles in system.
 */
void ParticleSystem::draw() {
   if (_num_p != 0) {
      for (int i = 0; i < _num_p; i++) {
         if (particles[i]->_lifetime > 0) {
            glPushMatrix(); {
               glTranslatef(_source.x, _source.y, _source.z);
               particles[i]->draw();
            } glPopMatrix();
         }
         else {
            particles.erase(particles.begin()+i);
            _num_p--;
         }
      }
   }
}
```

Animated Textures - Trent Ellingsen

This section describes the functionality of the animated textures that is used in the game Lume in such places as the menu, introduction, growing objects, and other areas.

General Concept

The purpose for using animated textures is to create the feeling of videos being used in the game. The key concepts in creating the Sprite class which allows for this functionality includes a need for a image sequence, a loader method that imports all the images, and a specific update method that iterates through the images with each iteration.

Why Animated Textures are Exciting

In games today, the leaders in the industry are able to create worlds that are dynamic and changing. With the use of animated textures, the world inside Lume comes to life by the character creating new blocks. When the user clicks, a block is drawn on a building to aid them in reaching the top of the city. Along with the block growing out of the building, a “flourish” animation is shown to create the feeling of organic life growing from the block. This aspects brings a great “Wow” factor to the game and makes it more appealing to the eyes. With the animated menu, introduction, loading screen and other aspects Lume feels like a dynamic game due to the use of animated textures.

Creating an Image Sequence

The first thing to do is to use a desired way to create an animation. In Lume, all the animated textures were created using Adobe After Effects. To create animations using After Effects, tutorials may be found at http://www.videocopilot.net. To create the growing object flourishes that are drawn when the character creates new blocks in the game, an after effects pack called Evolution was purchased from the Video Copilot website as well. After creating the animation that will be used an image sequence must be generated.

To create an image sequence, first the desired movie is selected, then opened in Quicktime 7 Pro. Inside Quicktime 7 Pro the user goes to the File menu and selects Export. The settings are then set to image sequence with .jpg as the type of image. This will create an image sequence that can be read in by the Sprite class. To be properly loaded into the program the image sequence must be put into a folder with the same name.

Loading in the Image Sequence

To load in the image sequence, the folder with the images is specified as well as the number of frames that are desired to be loaded. The loader then simply iterates through all the images and loads them using the texture class. The code for the loader can be seen here.

```
/**
 * This loads the textures under the starting name of tex
 * and has the known amount of f frames in the animation.
 */
void Sprite::load(const char *tex, int f)
{
   if (f < 1) return;
   char word[64];
   bool doesExist = false;
   for(int i = 0; i < f; i++){
      sprintf(word, "textures/%s/%s%03d.jpg", tex, tex, i);
      tempInt = LoadTexture(word, &doesExist);
      indexes.push_back(tempInt);
   }
   id = indexes[1];
}
```

Updating the Texture

To create the feeling of the movie, the update function iterates through the frames that are loaded in inside the update function. Every time the update method is called the time variable is increased by dt. If time is greater than the seconds per frame then the frame is incremented. There is an option for the animated texture to be looped. If the last frame is reached then the current frame is set back to the first frame. If the sprite is specified to not be looped then the frame is set to 0 which is created to be the same as the last frame and is specified to not change if set to 0. The code that is described here may be seen below.

```cpp
/**
 * This method updates the current frame based on time.
 */
void Sprite::update(float dt)
{
   if (indexes.size() <= 1){
      return;
   }
   if ((time >= spf)) {
      if (isLooping || !finished) {
         frame++;
      }
      else {
         frame = 0;
      }
      if (frame >= (int)indexes.size()) {
           if (isLooping) {
              frame = 1;
           }
           else {
              frame = 0;
           }
           finished = true;
      }
      id = indexes[frame];
      time = 0;
   }
   time += (float)dt*TIMERINT;
}
```

Logo Design - Trent Ellingsen

To create the best look and feel for the game and to fit the storyline element that KOG has taken over the world the environment has many company and store logos with KOG in the title. This design choice creates the correct feeling for the cities and allows the player to feel like they are within the world that has been taken over by KOG. Figure 6 shows the view in the skyline level of the world.

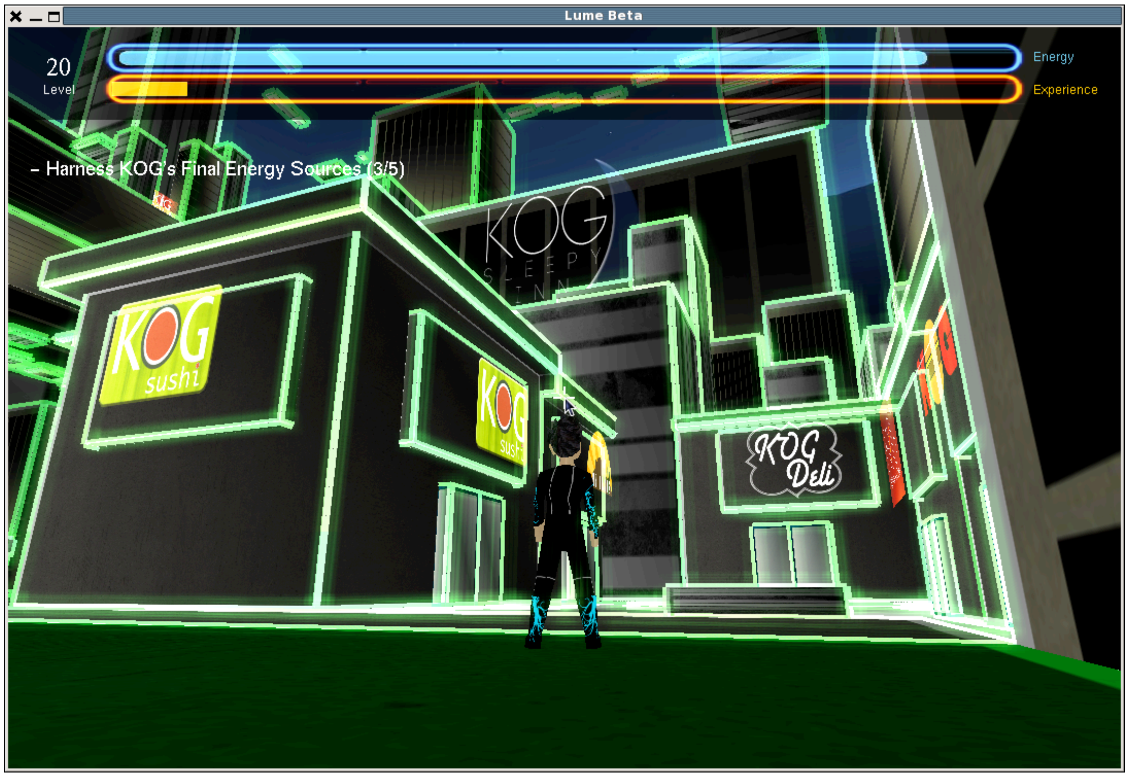

Figure 6: Skyline Logos

There are approximately 40 different logos that are used in the game. These logos were designed using photoshop software with some specific ideas in mind when being created. All logos met the following specifications:

- Pixel Ratio of 300 x 400 @ 300dpi
- Black background (for alpha blending)
- The use of the word KOG
- Imitation of some real world company

Once all the logos were created, the logos were placed in the world in front of buildings to allow for the correct look and feel. To see a sample of the logos see Figure 7 below.

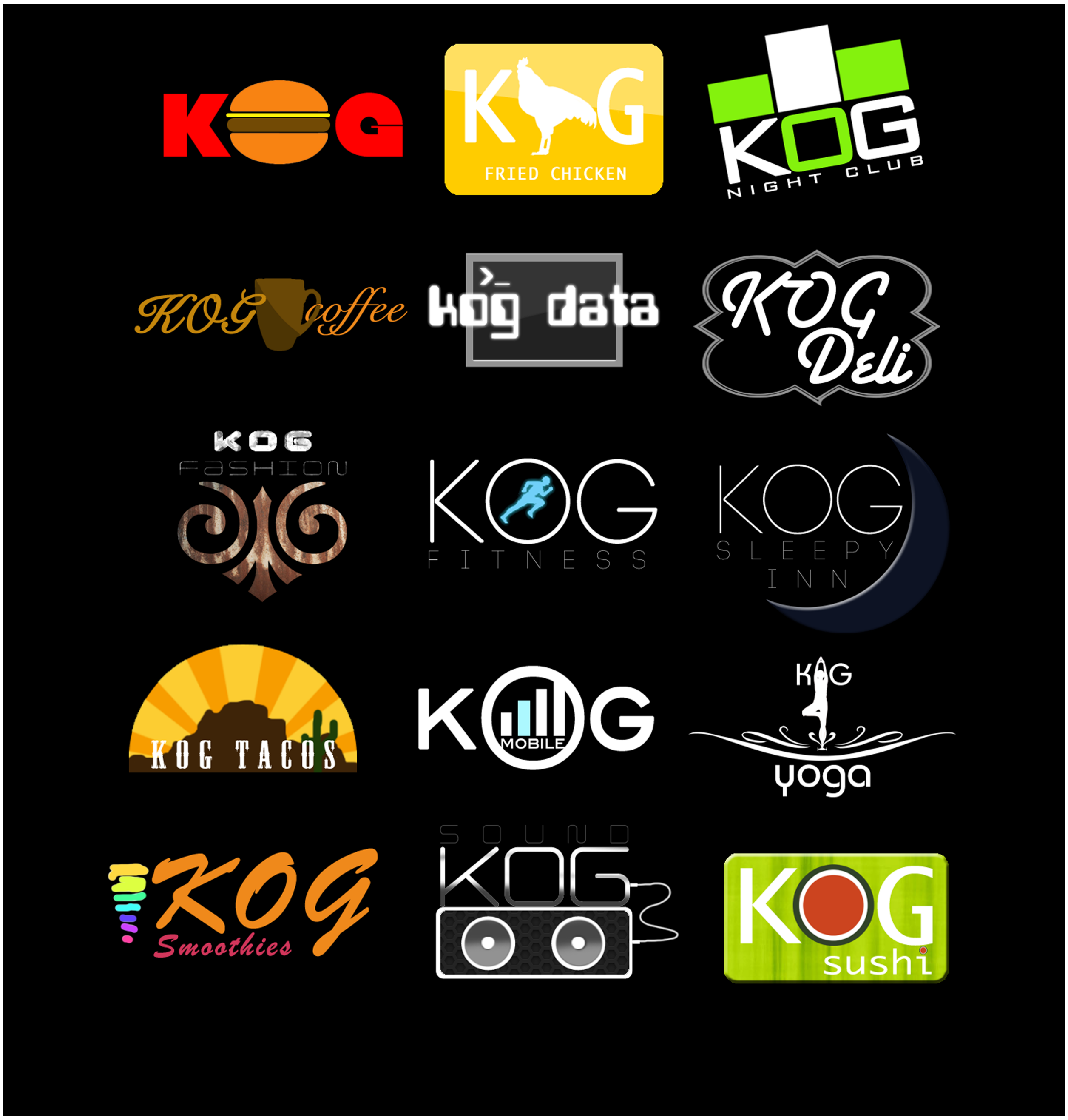

Figure 7: Logos

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 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 creatively use technology to create an appealing look. After completing a level, the world lights up as shown in Figure 8. Robot enemies are destroyed when sprinting through them, creating a particle effect that surrounds the robot as shown in Figure 9. 3D models of trees are animated to be brought back to life shown in Figure 10. A bloom shader is used on top of the trees to give them a pretty glow. An excessive amount of blocks built on a single wall shows the creation and life that is brought to the game seen in Figure 11. 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.

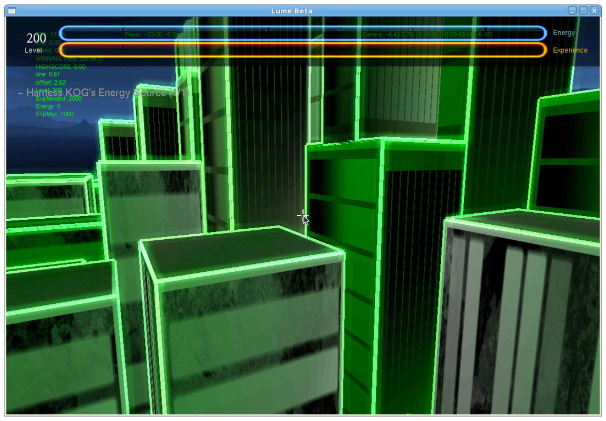

Figure 8: Level Completion

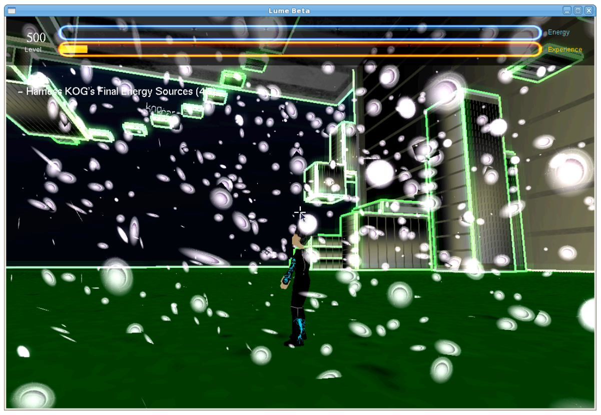

Figure 9: Particle Explosion

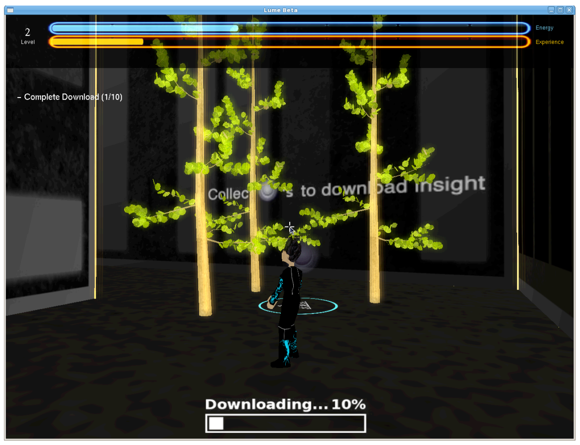

Figure 10: Trees Grown

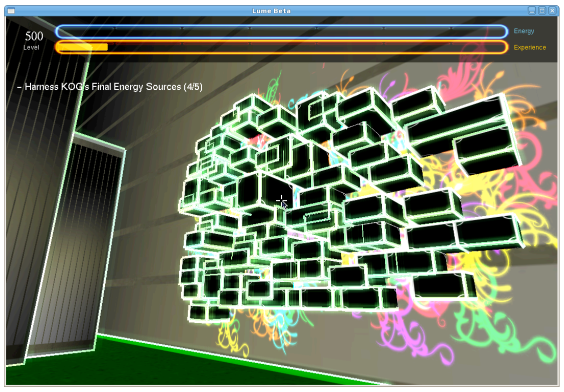

Figure 11: Blocks Created

We were also happy that we were able to create a game with a simple game mechanic that gives the player a lot of 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 built shown in Figure 12. The player can jump on the block to maneuver around the world. In the top left is the current object the player must complete, with a progress bar towards completing the objective below.

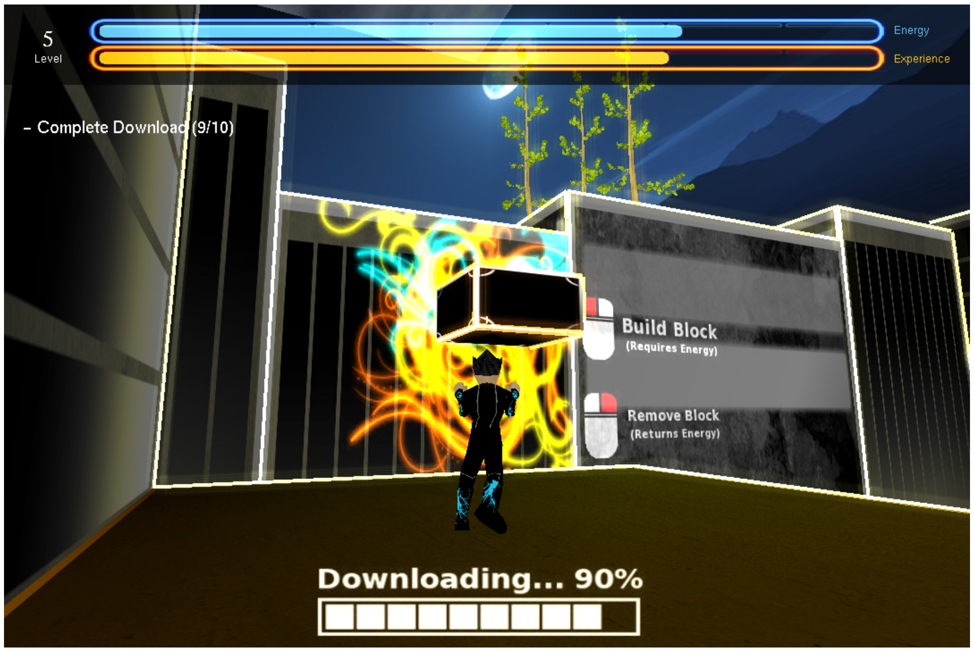

Figure 12: Building

After the creation of Lume, we wanted to get user feedback on the playability, speed, how well the player is able to follow the story, and many other aspects. Listed below are 3 individual play tests and the feed back given by those participating. The feedback shown as well as other feedback given, was used to alter the game in terms of level design, what the look and feel was like, how intuitive the controls were and other aspects.

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?** 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 (similar to Zelda load screen where it shows stats) • Multiple saves for different players • Freebies - 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 - (17/20)	**While playing Game Suggestions** • Make objective more clear - Download insight isn't self explanatory • 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. **Fun Level (1-10) 1- Not fun / 10 - Totally awesome:** 6/7 **Game Crash?** No **Laggy?** Only at 1 point **Bugs seen?** Yes: • 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 to make a difference **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

Play Tester	Comments
Ken Li - (17/20)	**While playing Game Suggestions** • Ability to go anywhere makes me feel unsure if I'm going the right way • Asked what the 2/10 was (didn't recognize that it was an objective) • Camera zoomed in when walking along wall • Want animation for creating the blocks (referenced the portal gun) • "Defeat KOG" in overworld was unclear • Can fall in a place w/o energy and get stuck (maybe use 'r') • Can't feel effects of the +1 to abilities • In suburbs, explain moving platforms will be turned on once you complete the first objective **Fun Level (1-10) 1- not fun / 10 - totally awesome:** 5 **Bugs seen?** Yes: • Leveled up then able to take all blocks back • Bad picking sometimes **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 **Miscellaneous** Stopped out of frustration on bottom of suburbs tower.

Based upon 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 shown in Figure 13.

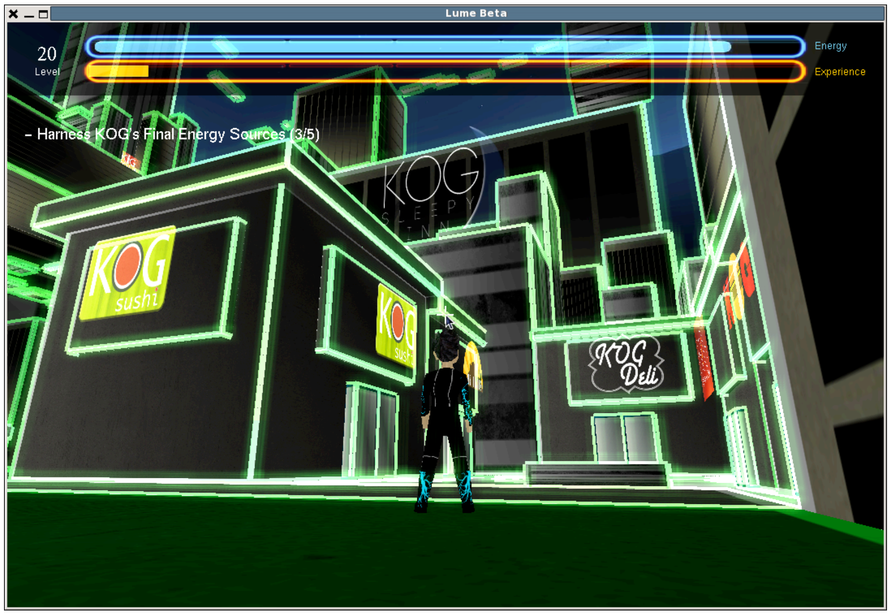

Figure 13: City with Logos

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 14.

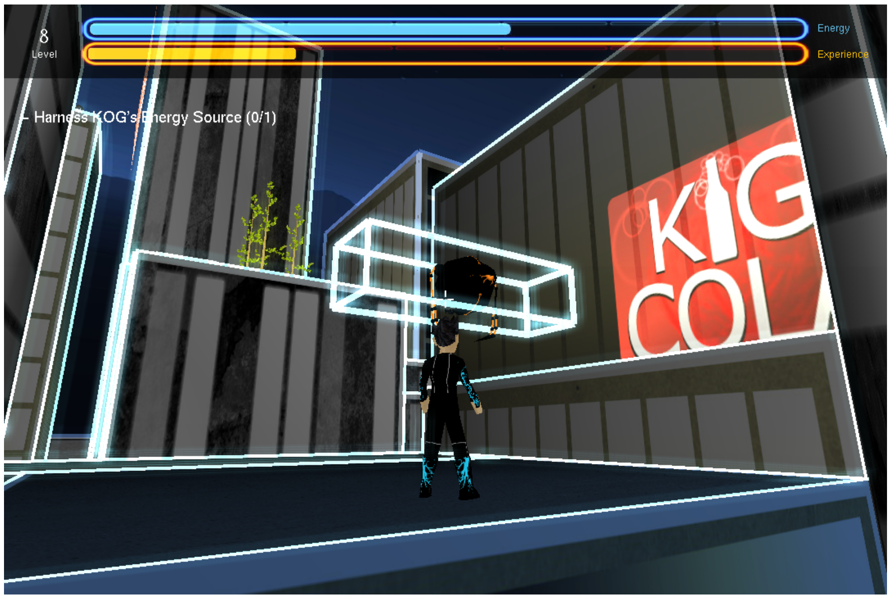

Figure 14: Freezing Ability

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.

References

irrKlang - An open-source library for playing sound files.

The Freesound Project - A repository of free sounds.

Swiftless Tutorials - Tutorial for heads up display.

GLSL - An open-source library for Open GL Shading Language.

Lighthouse 3D - Tutorial for view frustum culling.

NeHe - Tutorial for blur effect.

Journey In The Dark

MD5 to Blender 2.49b Importer and Blender 2.49b to MD5 Exporter

UV Mapping Tutorial 1

UV Mapping Tutorial 2

Maya Tutorials

MD5 Importer