LUME

An Interactive Real-Time 3D

Adventure Game

By: 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.

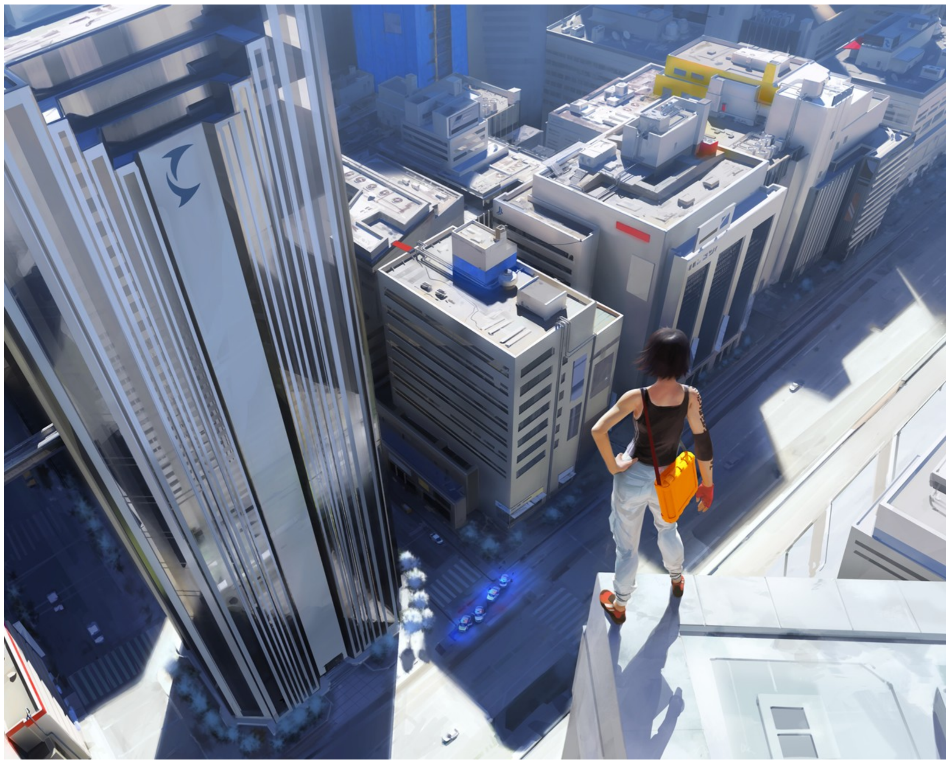

Figure 2: Mirror's Edge

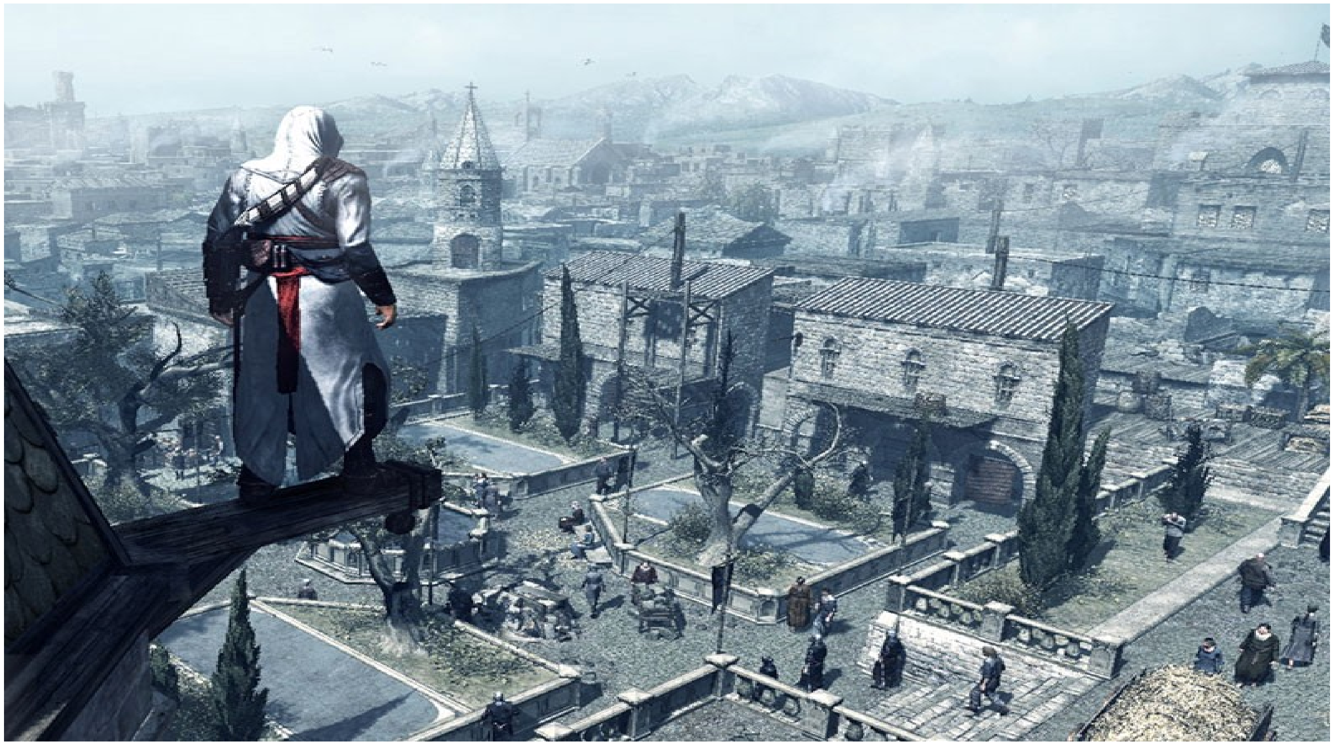

Figure 3: Assassin’s Creed

Figure 4: Super Mario 64

Algorithms Overview

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.

Algorithms Detail

Player Class:

Lume needed a way to represent the current state of the player as the user ventured through the KOG universe. The Player class contains all the information regarding the player's current level, current energy, position in the world, whether or not the player is jumping, building a block, or removing a block, where the last checkpoint was collected, and even what direction the player is currently running.

Level - The player's current level determines how high they can jump as well as how fast they run. Experience is gained by building blocks and collecting energy. When a player obtains enough experience for a level-up their base stats increase which allows the user to get a feeling of progression as they play throughout each city and increase their stats.

Energy - The player's energy determines whether or not they can build blocks. Energy is incremented each time a player collects an energy orb.

Position - The player's position is their current x,y,z coordinates. Keeping track of the player's position allows the user to move through the level by constantly checking collisions with the world against the player's position. Using the player's position the character model can be drawn and the third person style camera can be centered behind the player.

Animation States - Animation states are simple Boolean values that are used to determine whether the player is currently building a block, currently walking, currently jumping, or currently destructing a block. These states allow different animations to be drawn when the player is performing different actions. Animating the character helps provide a sense of realism to the user.

Velocity - The player's velocity is used to determine what direction and with what speed the player is currently headed. User input determines which direction the player moves and the player's level primarily determines with what speed the player moves. A simple model of gravity is used when a player jumps. Variable jump height was also implemented so that the user could vary how high the player jumps based on how long they held the jump key.

Checkpoint Position - Every time a user collects an energy orb the checkpoint position is set so that when the user falls from a platform and dies, or decides to hit the restart key, they will respawn at their last saved checkpoint position. Checkpoints are particularly necessary in platformer games where traversing a difficult section of platforms may take several tries, having the ability to quickly respawn and try again keeps the user engaged and attempts to reduce frustration.

Bounding Spheres and Collision Detection:

User interaction in Lume was made possible by the idea of bounding spheres and collision detection. The player is constantly checked for collisions between buildings, orbs, and triggers to provide a sense of real time interaction. Without collision detection the player would simply fall through the world. Nearly everything in Lume stems from a base obj class. Each obj has an x,y,z position. Using these x,y,z values new dx,dy,dz values are calculated by dividing x,y,z by 2. A bounding box is then created by giving each individual dimension (x,y,z) minimum and maximum values. The minimum and maximum bounds for the x dimension are x - dx and x + dx respectively. Once the same process is repeated for the y and z dimension the obj will have a bounding box with which allows collisions to be tested. One of the key methods in this class is the hitCollision method.
bool obj::hitCollision(obj object) - This method returns a Boolean true if a collision occurs between the base obj that is calling the method and a different parameter obj and false if no collision occurred.

Using the bounding box obtained in the obj class testing hit collisions becomes a simple task of elimination.

1) First check:

```
        if (( x + xmax ) <= ( object.x + object.xmin ));
```

This line of code states that if the base obj’s maximum x position is less than the given obj’s minimum x position, then no collision can possibly exist between these two objs and the method returns false.

2) Repeat a similar check for if the base obj’s minimum x position is greater than the given obj’s maximum x position, if so then no collision can possibly exist between these two objs and the method returns false.

3) Repeat those two checks (steps 1 and 2) for the y and z dimensions.

4) Lastly in order to compensate for slopes in Lume a final check must be done for the y dimension.

```
if ( object.y + object.ymin - y <= fx * ( object.x - x ) + fz * ( object.z - z ) + fy );
```

This line of code checks the y dimension against the x and z dimensions multiplied by slope constants. If this check returns true, then there is a collision between the two objects.

5) If none of the previous checks passed, then a collision does occur between the two objects and the method returns true.

The image below (Figure A) outlines the normally invisible bounding boxes for both the player and the objective in front of him. When the player's bounding box intersects with the objective's bounding box then a collision occurs and the game is signaled that an objective has been collected.

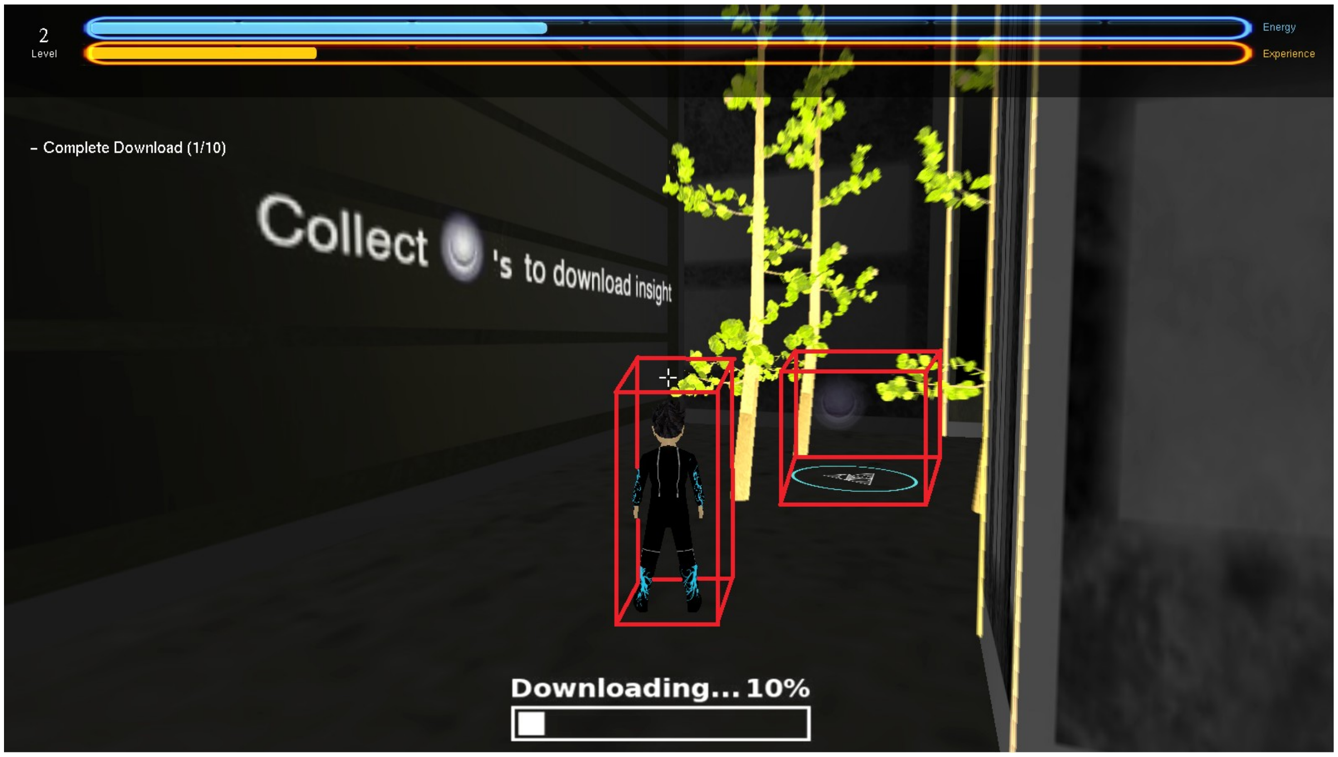

Figure A

View Frustum Culling

View Frustum Culling is a quick and efficient way to help optimize the performance of any game where the user does not see every object in a world at the same time. Without VFC, every single object in a world is sent down the graphics pipeline to be drawn regardless of whether or not the user can currently see the object. Using VFC resources and computing time are not wasted on objects the user cannot currently see and only objects that are currently within the view frustum are sent down the graphics pipeline to be drawn.

Using OpenGL and GLUT the six planes needed to simulate the view frustum can be obtained by extracting the GL_MODELVIEW_MATRIX and the GL_PROJECTION_MATRIX. Once both the Model View and Projection matrices have been extracted some matrix math can be used to convert them into planes.

```
// Extract The Numbers For The RIGHT Plane
 planes[0].a = m[ 3] - m[ 0];
planes[0].b = m[ 7] - m[ 4];
planes[0].c = m[11] - m[ 8];
planes[0].d = m[15] - m[12];
// Normalize The Result
t = GLfloat(sqrt( planes[0].a * planes[0].a + planes[0].b * planes[0].b + planes[0].c *
                        planes[0].c ));
planes[0].a /= t;
planes[0].b /= t;
planes[0].c /= t;
planes[0].d /= t;
```

Once a similar process is repeated for the left, top, bottom, near, and far planes using the plane equation $Ax + By + Cz + D = 0$ can be used to determine if a point is inside or outside the frustum. By sending every vertex of every object in the world through a pointInFrustum method which returns true if the vertex is inside the frustum and false if it is outside allowed us to cull any objects outside of the player's viewing volume. However, an issue arose where even though each vertex was outside of the player's current viewing volume, the object itself should be located within the player's current viewing volume. This situation occurred with particularly large objects in the world where the vertices would be located on opposite corners of the city and while the object should clearly be in the scene it was being culled because no particular vertex was within the viewing frustum. To solve this problem, a radius was added to every obj which represented the length of the longest dimension of an object. Now when checking if a point was within the frustum, the radius was also compared to the planes of the viewing volume to ensure that for particularly large objects with vertices outside the frustum the radius would still lie close enough to the frustum to prevent them from being culled.

Level Design and Objectives:

Level design is an important aspect of any game because if different levels are too difficult or too easy the user can become bored or frustrated. Through the development of Lume, three levels were fully completed before the addition of objectives. Levels initially had one goal: reach the top of the city. However, developing a game where the sole purpose was to reach the top of the current level became stale and repetitive. Objectives provided a way in which to better develop the storyline but also meant that levels needed to be completely redesigned around the objectives.

Objectives - Objectives for each level are loaded into Lume through the importMap file. Objectives are saved in a separate objective file and contain information about the name of the objective, the type of the objective, and the total number of triggers needed to complete the objective. An objective consists of an std vector of triggers which have a position and a bounding box to test player collisions against. The first level contains one objective: to download the AI Insight. A player completes a level by completing all the objectives for that particular level. Downloading Insight allows the player to receive feedback and hints about the storyline as well as provide helpful clues on how to complete objectives in various levels.

Camera Intro Path and Free-form Camera

After receiving feedback from users play testing Lume, it became clear than many play testers were confused about where the objectives were located. The solution to this problem was the inclusion of a camera intro scene that began at the last trigger in the first objective for a level, followed a set path loaded via the importMap file in a similar fashion as objectives, and ended at the player's spawn position which was intended to give the player a direction and simple path of where they should venture towards.

Camera Intro Path - The camera intro path was created by providing different x,y,z coordinates that represented different points on the path. While the intro scene was active, each camera update would move the camera position towards the next path point. The mouse can be used to look around the map intended to allow the user to plan their own route through the level. The most difficult aspect of following the camera path was ensuring that the camera actually reached one of the path coordinates. The equation used to ensure that the camera crossed one of its predetermined path coordinates was achieved by estimating a miniature bounding box around the path coordinates.

```
if ( (currentCameraPosition / predeterminedPathPosition ) >= 1.01 &&
  (currentCameraPosition / predeterminedPathPosition ) <= 0.99){
        then proceed to the next path position
}
```

The above equation checks to see if the camera's current position is within 0.02 in x, y, and z of the predetermined path position. If the current position is within that range, then the predetermined path position is incremented to the next position in the camera path position std vector. This process is repeated until the vector is empty and the camera has reached the player's spawn position.

Free-form Camera - Although the free-form camera was never used in the game play of Lume it was used to obtain some nice screenshots in game from any position in the world. Lume already had functionality for entering ‘first person mode’ to control moving objects and platforms. Using this idea, an invisible trigger object is created in each level that allows the player to take control and move around the world in ‘first person mode’ in any x, y, and z direction.

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 (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.

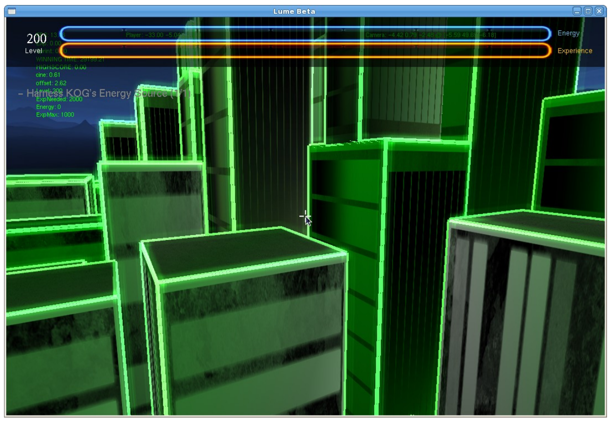

Figure 5

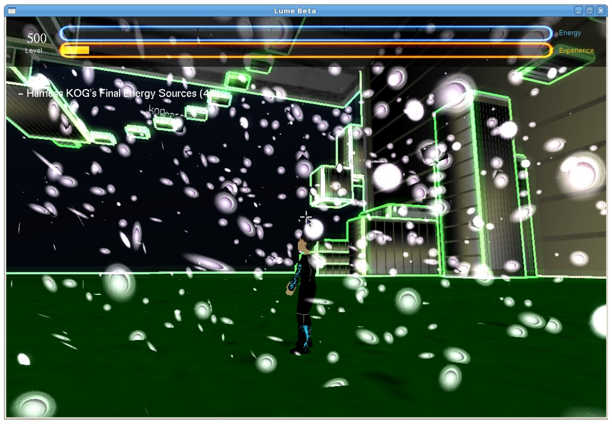

Figure 6

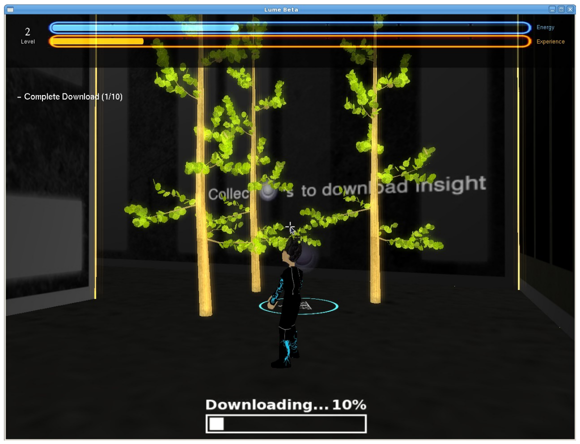

Figure 7

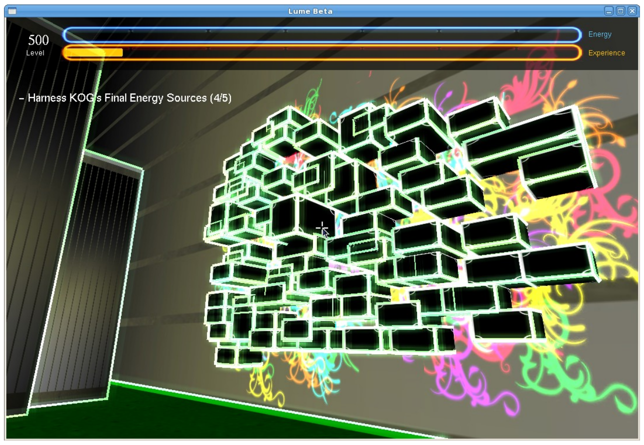

Figure 8

We were also happy that we were able to create a game with a simple game mechanic that gives the player a lot of freedom to explore the world while still having set objectives that keeps the player interested. Below is a picture of the blocks that the players build (Figure 9). They can ladder 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.

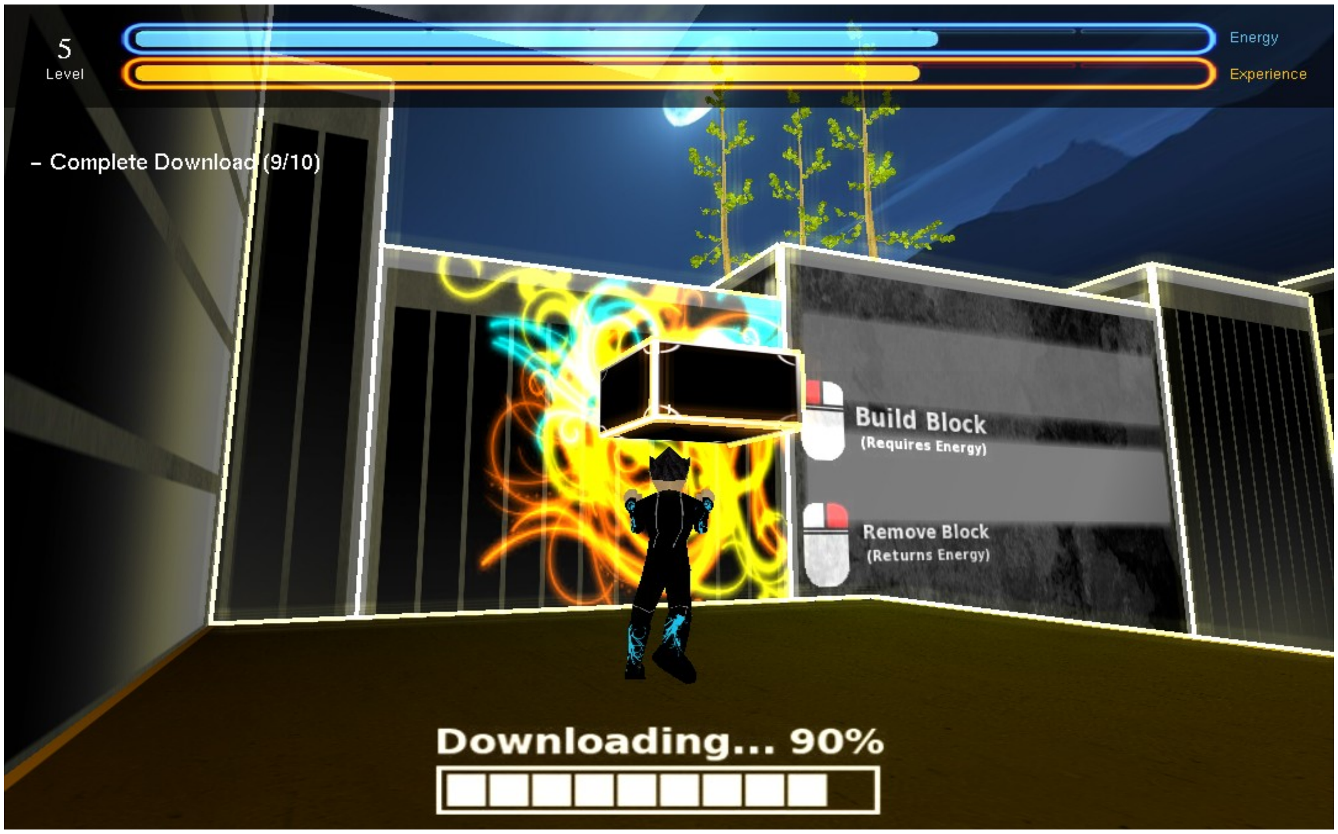

Figure 9

Play Tester	**Comments**
Joanne Mark	**Fun Level (1-10)**
(Week 15/20)	**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

Story Interpretation

Save your city from evil KOG by getting all those energy light thingies

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)

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?

at one point -> see bug section below (otherwise no lag)

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

Story Interpretation

KOG took world - you kind of bring life back

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

Ken Li

Week (17/20)

While playing Game Suggestions

- Ability to go anywhere makes me feel unsure if I’m going the right way
- Thought skybox was a wall and jumped to death
- 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 and thought I didn't complete 1st world (maybe it could come up after other worlds)
- Can fall in a place w/o energy and get stuck (maybe use 'r')
- Can't feel effects of the +1 to abilities
- NEED MOMENTUM TO CARRY AFTER SPRINT RUNS OUT
- Frustrated on suburbs because of the lack of energy
- Walk animation odd
- 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

Game Crash?

No

Laggy?

No

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

Story Interpretation

Parody version of Tron

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).

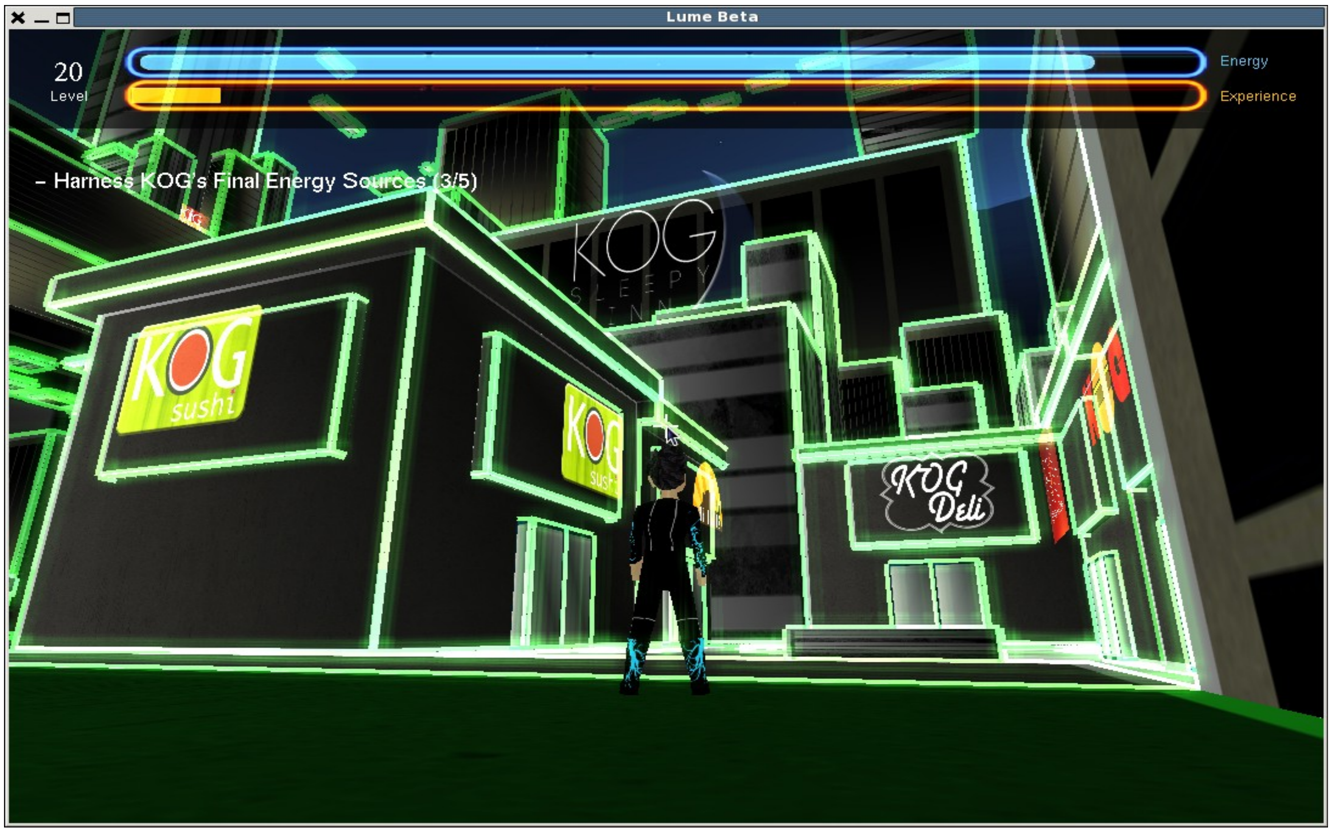

Figure 10: Logos in City

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.

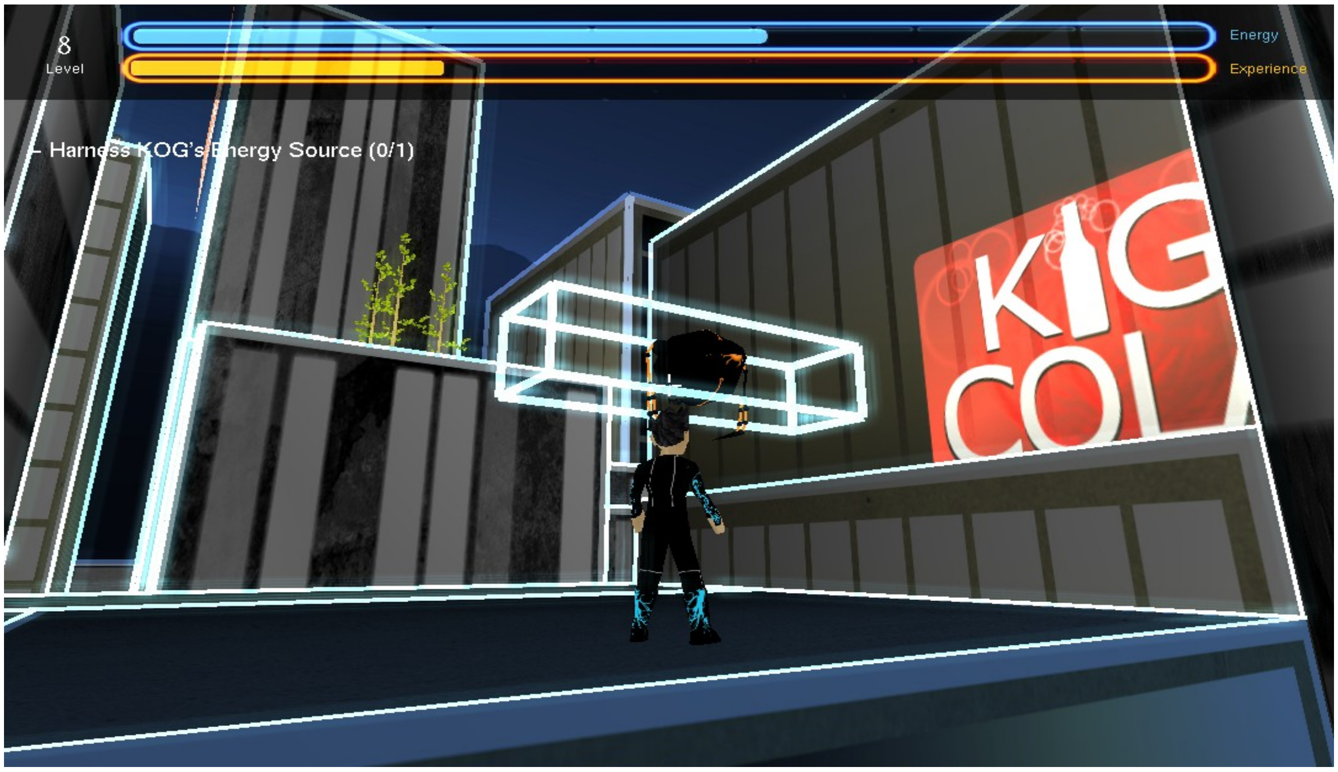

Figure 11: Freezing Ability

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.

Conclusion

The biggest surprise throughout the two quarters of designing and implementing Lume was the amount of work a team of seven students can accomplish. When Lume was pitched six months ago it was a simple idea and after two quarters has become a fully fleshed out 3D game. The time spent working with a team on Lume has given me the opportunity to improve my collaboration skills. Nearly every piece of our software was touched by multiple individuals. This project has also vastly improved my knowledge of the graphics pipeline and various technologies commonly used by industry for creating computer games. Lastly, the aspect of this project that I am most grateful for was that the entire process was built upon a platform that was conducive to using one's creativity and imagination.

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

Lighthouse 3D Tutorial on VFC - http://www.lighthouse3d.com/tutorials/view-frustum-culling/

irrKlang - Open Source library for playing sound files - http://www.ambiera.com/irrklang/

Quick Reference for Open GL Shader Language - http://www.opengl.org/documentation/glsl/

Basic bounding volume descriptions - http://en.wikipedia.org/wiki/Bounding_volume

Camera transform reference - http://www.opengl.org/resources/faq/technical/viewing.htm