Third Degree

Mark Paddon, Chad Williams, Jon Moorman,
Joshua Marcelo, Michael Sanchez, Tim Biggs

California Polytechnic State University
Winter 2010 – Spring 2011
Prof. Zoë J. Wood

Introduction

Video games have been driving hardware and graphics development ever since the earliest video game consoles. Having a senior project that consists of making a substantial video game is an excellent way to learn about and implement many different types of graphics technologies. Building a video game to the scale of a large team project provides experience in all aspects of real world software development. From designing the game to team management, the process of building a video game builds the experience necessary in many aspects of software engineering. My Senior Project involved the creation of a full 3D interactive video game in the action-adventure genre.

The *Third Degree* game began in Cal Poly's Winter 2011, CPE 476++ - Real-Time 3-D Computer Graphics Software Systems class. The focus of the class was to create a video game based within a 3-D environment, with course requirements to integrate certain graphical technologies into the game. In CPE 476++, Mark Paddon, Chad Williams and Michael Sanchez had the initial idea for a side-scrolling platformer built within a 3-D environment called *Third Degree*. Various people assisted in the project in order to provide assets and support by way of music, concept art and models, as well as general story development and voice acting. Tim Biggs, Jon Moorman and Joshua Marcelo then joined their programming team to also work on the game throughout CPE 476++, as well as into the Spring 2011 quarter as a senior project where our work on *Third Degree* continued with improvements to graphical technologies while also fleshing out the foundation for the story.

Project Overview

Genre & Setting

Third Degree is a 3D side-scrolling adventure game. The game takes place in the mind of the main character and the player is continually immersed with story driven game play. *Third Degree* combines story elements, traditional platforming and interesting game mechanics to provide a unique

player experience.

Game Mechanics

The core game mechanic of *Third Degree* is the concept of "mental deterioration" (MD). The status of this "deterioration" state is reflected in the Mental Deterioration Bar, or MD Bar for short. The MD of the player is reflected in many ways in the game. The most apparent affect that MD has is on the game's environment; initially the environment will reflect a Victorian London environment, and will slowly transition to a futuristic setting as the MD increases. Additionally, the higher the MD, the more the distorted the environment will become, furthering the concept of the player's mental state. As the MD reaches nearly full (equivalent to "death"), the player must *focus* in order to bring down their MD and restore the environment to a playable state. If at any point the player maxes out their MD, they will respawn to the last checkpoint.

Aside from the MD, the player will be fighting against enemies in the twisted environment that the game is set in. Enemies are strategically placed throughout the map, and the enemies' attacks will increase the player's MD for each "hit" on the player. For defense, the player is equipped with a melee attack, as well as a simple gun. In addition to the enemies impeding progress in the map, *puzzle objects* are placed throughout to make navigation more difficult. Examples of puzzle objects include trapdoors, swinging platforms, and swinging blades. To pass some of these puzzles, "Fire legs" must be employed. Fire legs is the game's mechanic to allow the player to jump higher than they normally would; the jump height increases with the player's MD, giving the gamer an incentive to balance their MD appropriately.

Story

The game follows the story of a convict kept in confinement who is essentially given a chance at redemption through a special testing program. A recent alien artifact has crash landed on the Earth's surface, and a panel of scientists is conducting specialized experiments to find out what it does. The convict is one in a line of test subjects given a chance of freedom through experimentation. When the artifact is fitted on the convict's head, he is put in some kind of virtual environment that resembles London in the 1860s. The convict, though determined to gain his freedom, soon feels the grasp of insanity closing in around him, and the only way out is to either finish the virtual simulation, or die trying.

Project Design Specifics

As a development platform for *Third Degree*, Microsoft Visual Studio 2008 was used. Visual Studio gave us advanced debugging capabilities and the ability to easily compile a Windows executable to send to others. For our editor UI system we used Nokia's Qt libraries and tools. The Visual Studio plugin was used to integrate Qt and facilitate the design of the Level Editor. The source code was under version control and the primary language was C++. The repository for our entire project was hosted

on Unfuddle.com. All team members were given accounts and access to the repository to commit and update changes to their projects accordingly.

The game uses a lot of graphical shader effects written in GLSL. For this, ATI's render monkey was invaluable as an initial tool to visualize and debug shaders. The game's physics used NVIDIA PhysX. Originally, Bullet was used. However, Bullet's documentation was poor, and few examples were available. Therefore, the decision was made to rewrite the project. PhysX provided us with a professionally written physics library and excellent documentation to incorporate into our game. It is an industry used library incorporated into many commercial video games. For sound, the irrklang library was used.

Related Works

While there were many influences for *Third Degree*, the following works both influenced the gameplay as well as helped to provide examples for how to approach certain tasks for components of the *Third Degree*.

Trine

The general mood and feel of the game were greatly influenced by this side-scrolling platformer, Frozenbyte's *Trine*. Visual inspirations, as well as overall feel of the gameplay mechanics such as movement, puzzle object interaction and elements of the combat system helped in making decisions for *Third Degree*. The Figure below shows an in-game screenshot for *Trine*.

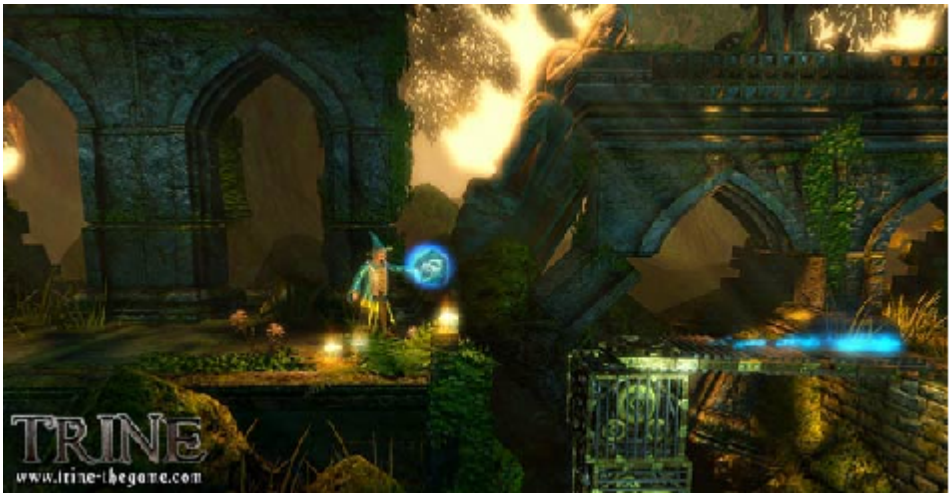

Fig. 1 – Screenshot for Trine

Maya

The transformation tools were modeled after many 3-D graphics software, particularly Autodesk

Maya. The figure below shows an example of the transformation tools used in Maya.

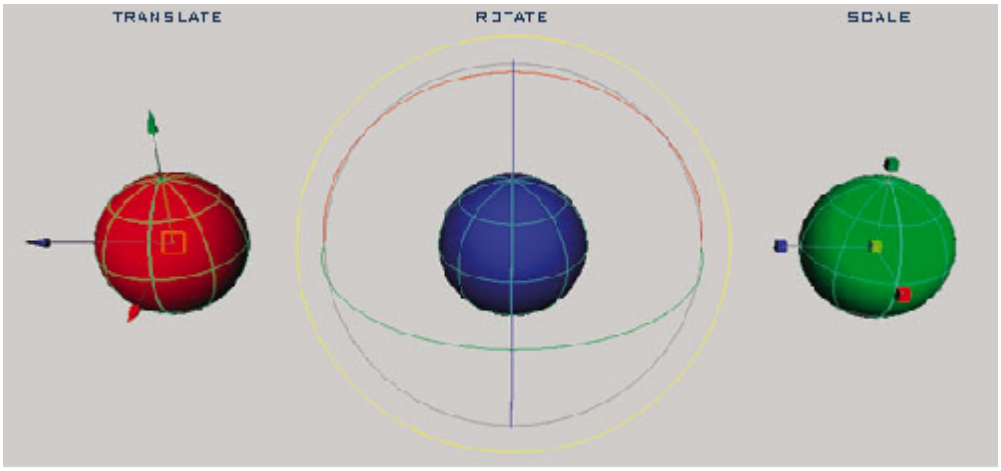

Fig. 2 – Examples of the Translate, Rotate & Scale tools in Autodesk Maya.

Doom 3

The animation in the game utilizes the MD5 format used in a number of 3-D games, most notably in Activision's *Doom 3*. The MD5 structure created for *Doom 3* provides a robust and efficient means of representing animation. The figure below shows an example mesh from *Doom 3* modeled using MD5.

Fig. 3 – Example of a MD5 model used in Doom 3

Algorithms Overview

Project Technologies

Technology	**Authors/People involved**
View Frustum Culling	Josh, Mark
Skeletal Animations	Josh
Enemy AI	Josh
Player Control/Movement	Josh, Tim, Mark
Combat System	Mark, Josh

Editor - Object Transformations	Josh
Editor - Main Functionality	Chad
Deferred Rendering	Chad, Ryan Schmitt
GLSL Shaders	Chad
Core Engine Optimizations	Chad
Particle System Implementation	Chad
High Level Design	Mark, Jon
Map Loading/Saving	Jon
Physics Engine Integration	Mark, Jon, Tim
Object/Joint System (aka Puzzle Objects)	Jon, Tim
Glow Shader	Jon, Chad
Animated Textures	Mark, Michael
Menu System	Michael
OBJ Importer	Chad
Octree	Mark
Sound Design	Mark
Focus	Mark

Algorithm Details

Octree

Third Degree uses a large assortment of detailed assets that were created for the game. With the creation of complex maps composed of these assets, a performance problem arose. The game was rendering many thousands of triangles per frame, causing a drastic reduction in frames per second. To

alleviate this problem, a system to incorporate view frustum culling was implemented. View frustum culling allows us to render only objects being viewed by the camera, thus eliminating everything else from the scene. The following additional criteria were used to select the algorithm to implement:

- It must be fast. We cannot use the naive O(M X N) for culling intersection testing.
- Even though the game is 2.5D we need the utility of a full 3D data structure.
- Culling will also be Incorporated into logic calculations such as AI and Physics, based on proximity to player.
- In addition to geometry, it must quickly cull out lights, particles, and more.

An octree data structure was implemented to incorporate all of these requirements. An Octree is a spatial data structure that divides the map into eight "octants". It is a tree structure where the root node is the entire map, and the children are the eight octants created by dividing up the tree. When we are searching for a collision with a particular object in the world, the tree structure allows us to disregard branches as we narrow down our search by diving deeper and deeper into a specific octant. This gives us a huge performance increase over checking each object individually.

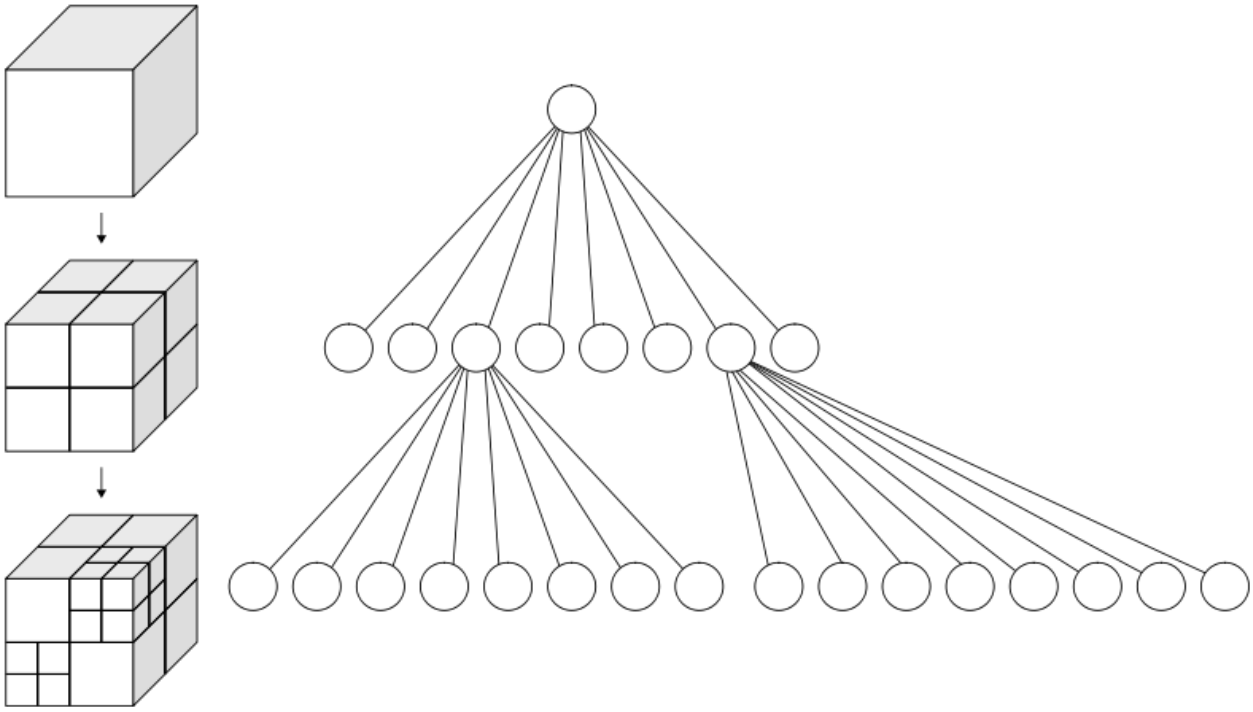

Figure 4: Octree Diagram

Each octant will have a predetermined maximum number of "buckets". A bucket contains a game object defined in *Third Degree*. A game object is the base class inherited by all objects in the game (geometry, particle systems, lights, etc). Every game object has a bounding box, which is used to place the object in the correct bucket. If an octant has more objects than it has buckets (in *Third Degree* the maximum is eight), that octant will be further subdivided into eight more octants. Therefore the map will be heavily subdivided only in places where large numbers of game objects exist.

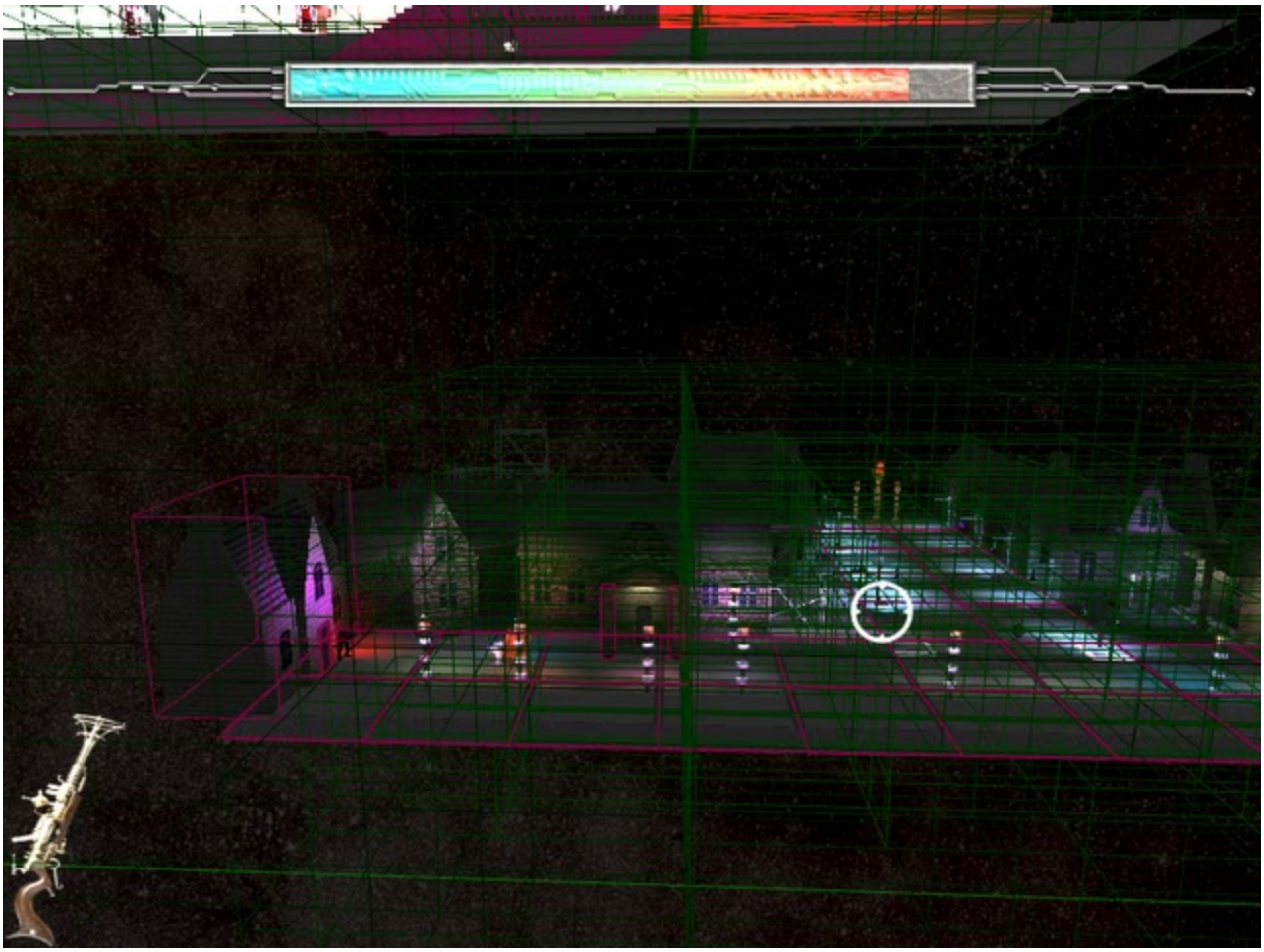

Figure 5: Level with no Culling

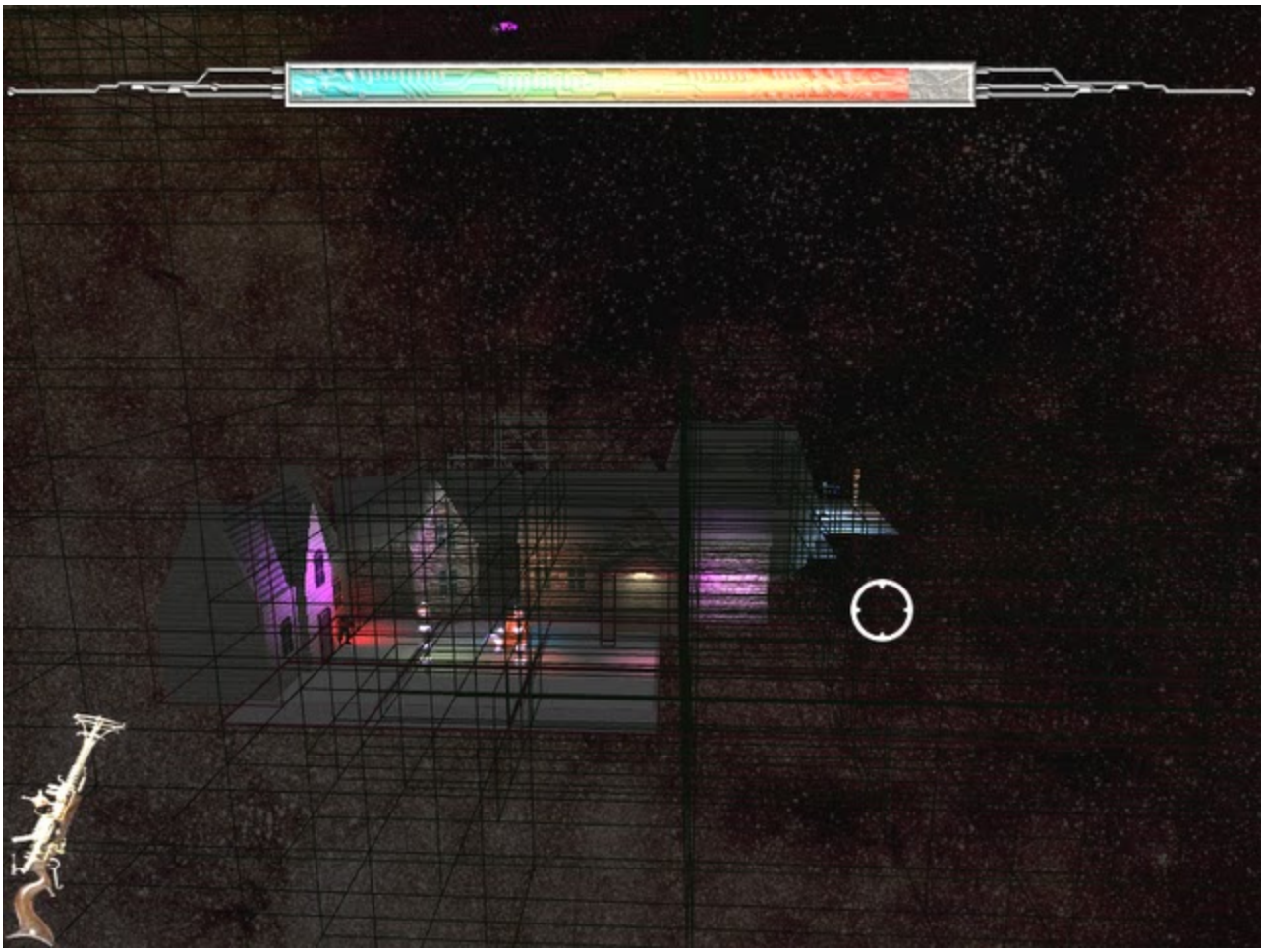

Figure 6: Level With View Frustum Culling

The basic traversal algorithm is based on recursion. The octree class is set up to function as a node, like that of a linked list. Each node contains a list of game objects it contains, the location of its bounding box, and a reference to its eight child nodes if they have been created. The application then has a reference to the root node of the tree. When an object is added, it is recursively sorted into the tree. Starting with the root node, it is tested against each child octant until the appropriate location is found. From there it will either be inserted into that object's list if there is room or that node will be subdivided into eight more octants, and its contents will be sorted amongst the children.

One main difficulty is updating the tree. We must first determine if an object has moved. If it has it must then be moved to an appropriate bucket. This is a very expensive task however because the following must occur:

- All dynamic objects would have to be checked for movement.
- If an object has moved, it must be relocated in the tree to an appropriate spot.
- Moving the object in the tree may involve serious tree restructuring.
- This system must be fast enough to work for a great many objects moving at any given time.

To allow efficient object tracking we first need to figure out which objects are moving. NVIDIA PhysX has a feature called active transform notification. This system gives us a list of physics actors that have moved since the previous update, allowing us to avoid checking all objects in the scene manually. From there we can access the associated game object from a given actor. In addition to this optimization, a timer is also added for updating the octree. We wait for 300ms to elapse before updating the octree. This gives no noticeable to effect to the culling and therefore to the player, but results in an additional large performance boost by rationing these computations. An object is then located in the tree and relocated accordingly.

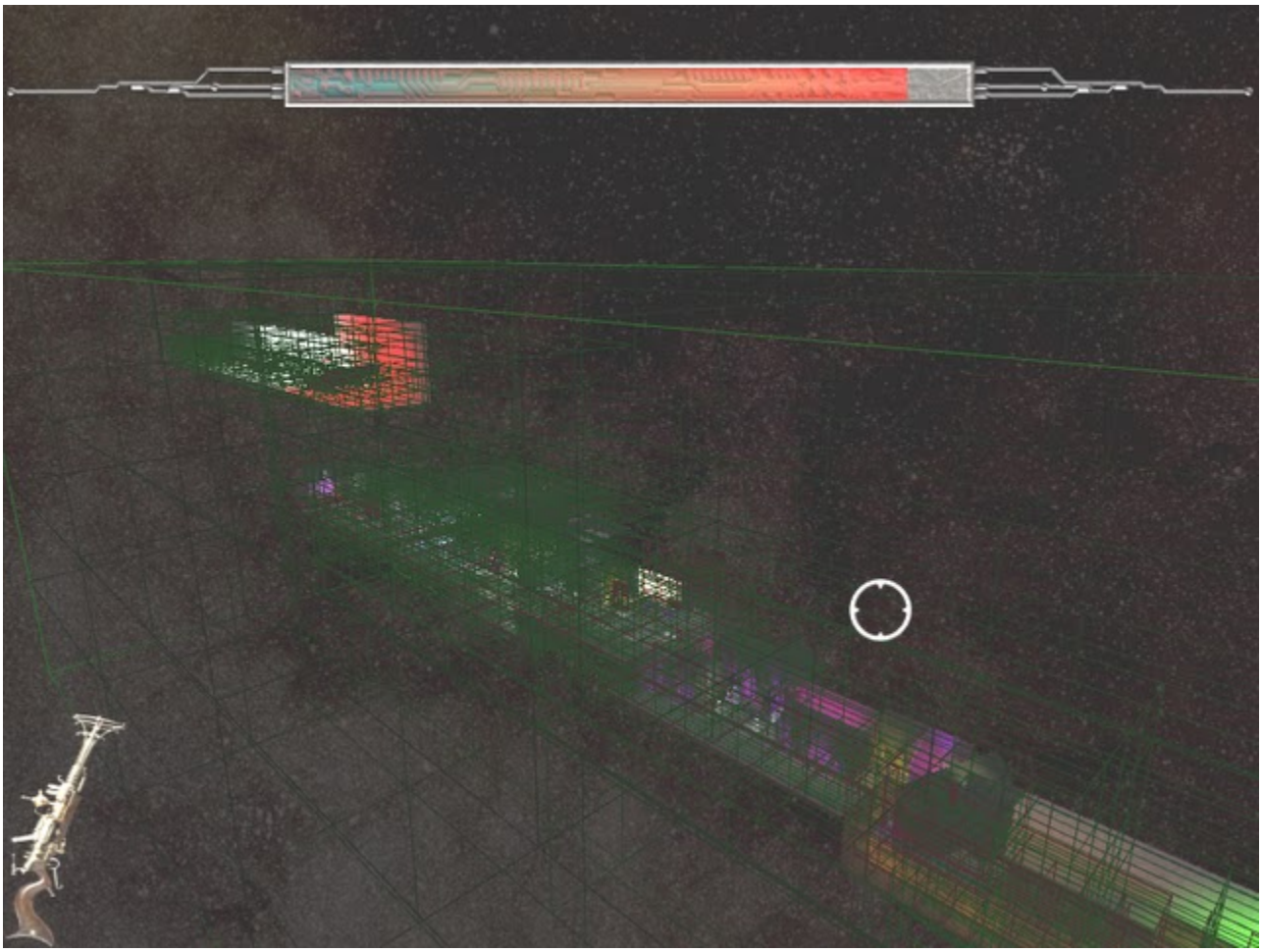

Figure 7: Entire level placed in octree. Notice the dense grid structure near level geometry, but sparse surrounding octants.

Combat System

One of the most important gameplay systems implemented was the combat system. *Third Degree* has a foundation that allows for ranged and melee combat. The initial steps of both of these systems are the same. If a ranged or melee weapon is selected the following occurs on a mouse click:

- Convert pixel coordinates of mouse click to world space. Send this information into the combat system.

- Since the player can shoot or swing a sword in a 360 degree range motion, the angle of rotation is calculated based on the location of the mouse click.
- A ray is then cast from the weapon in the direction of the attack.

Ranged and melee rays cast are handled slightly differently. A ranged attack will travel down the ray until an object is hit up to a maximum range. We will only ever hit a maximum of one object. A melee attack projects a short ray; this represents the reach of the player. A melee ray returns impact with all objects it hits, not just the first.

Math Breakdown

A simple unit circle is used in order to calculate the direction of the ray from the player. If we have the location of the player and the mouse click, we can use the atan2(y/x) arctangent function to find the angle in radians between the two points. Based on this angle, we can cast a ray from origin of the player in the direction of the attack. Now that we have the angle we can find the x and y coordinates on a unit circle by taking the cos and sin of the angle respectively. Since our game moves on a 2D axis, we can assume Z is always 0 and not worry about it. Given the ray equation R(t) = R0 + Rd * t (R0 is the origin, Rd is the normalized direction, and t is the distance from the ray origin) we can use the player position as the origin, and the point on the unit circle as the direction. The ray is then cast from the player and intersection testing is done along ray. Once a ray hits an object a force is applied to the object based on the point of impact and the weapon used. If the ray hits a dynamic object a force is applied to it causing the object to move. If it intersects an enemy the health of the enemy is decreased.

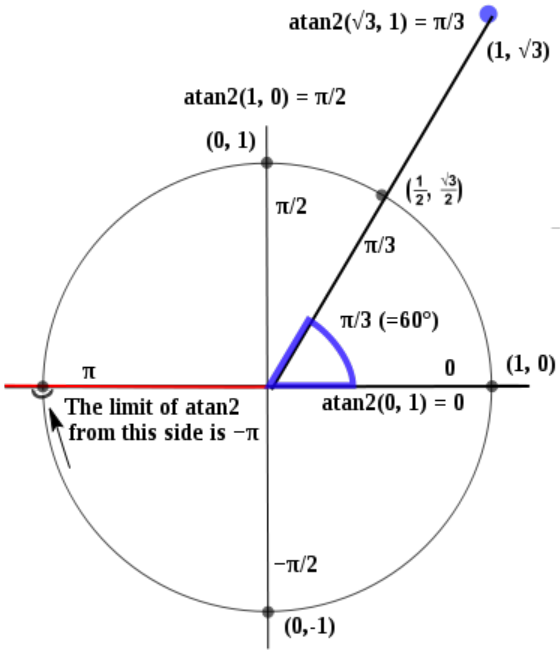

Figure 8: Example illustrating finding a point on a unit circle based off the angle of rotation.

Bullets

In our combat system, the results of a melee attack are handled instantly. Ranged combat

is handled differently, however, as we wait for a bullet to traverse the ray before it hits a target. Fortunately ray traversal is simple. We use a timer to increase the distance of the bullet along the ray (recall the equation R(t) = R0 + Rd) and draw the bullet at the new location accordingly. Once a bullet hits a target it is removed, and a spark particle burst is played at the intersection a point along with the applied force.

Another consideration is correctly drawing the bullet. We use a simple billboard that is rotated in the direction being fired. This is accomplished by first drawing the bullet image at the origin. Each texture coordinate is then multiplied by a rotation matrix, utilizing the angle we calculated above, rotating it accordingly. The texture coordinates are then translated to the final position on the ray and are drawn.

Results

Our team accomplished creating a 3D interactive game from scratch. At its heart Third Degree has a beautiful rendering engine that displays our complex environment. We also Incorporated physics based game play, an introduction to a gripping story, and a first level.

We are most proud of our aesthetics and visuals, and the story element of our game. The game's original assets, combined with the graphical effects of our rendering, produce stunning visuals. Our level design that was made available through our GUI editor allowed us to build a huge and detailed level made up of many props. While our game play has lagged behind our visual accomplishments we believe that by the end of the project we accomplished our goals for game play. Our Trigger system, physics driven puzzles, combat system, and AI combine for a rich variety of gameplay. These tools in concert with our level editor allow us to produce platforming puzzles, and iterate on their design very quickly. We are also proud of core game mechanic, Mental Deterioration. This original concept is central to our game design, and we believe it will be very compelling once fully utilized.

Figure 9: In game screenshot.

During testing there was lots of feedback given to help improve the game and determine what was working and what was not. Some of the aspects that people liked in the game were the graphically rich environment, with emphasis on the lighting. Players also liked the physics effects of the environment objects and the game actually being a side scroller. Some aspects that people did not like included not being able to see the bullets, better control of the camera(there was too much movement from the camera), inconsistent frame rates in certain areas of the map, and the gun of the player being too small.

The development process of *Third Degree* had many highs and lows. At the beginning of Fall quarter our project had thirteen 3D animators committed to producing content for the project. We had weekly meetings and many design talks. We provided them with the specifications needed for our models (format, scale, poly counts) as well as the lists of assets needed. After nearly an entire quarter of regular meetings it became clear that only about three animators were actually contributing to the project. Our team made the decision to keep those who were helping, and drop everyone else. In retrospect, we should have done this much sooner and saved ourselves a lot of trouble. At the beginning of the second quarter our team decided to redesign and rewrite our entire project. The main reason behind this choice was the poor documentation of Bullet, the physics engine we started with. This had

led to so many problems in the code base that it seemed the best option, and we believe it was good decision. In addition to incorporating the superior physics engine we were also able to incorporate the things we learned in the first half of the course into our project.

Conclusion

This project has been an accomplishment for me for many reasons. As the team leader, organizing and coordinating the project took a massive amount of energy and time. For me, it feels as if 90% of my time these last two quarters has revolved around this project. The organizational and leadership role for this project extended beyond the programmers and included all of our artists, sound designer, and voice actors. Leading this multi-disciplined project and being able to effectively communicate between students of all backgrounds was an amazing learning experience.

An incredible amount was also learned in the software engineering side of the project. This project was the largest piece of software ever created by anyone on our team at Cal Poly. The project's solution extends over three projects and includes hundreds of C++ files. The organization and design of the software was paramount and a great accomplishment. This course also gave me a strong mastery of C++, which I had never touched until this past year.

Lastly, learning the graphical algorithms used in this project has been eye opening. From view frustum culling and sophisticated spatial data structures, accomplishing graphical effects such as normal mapping, and being introduced to sophisticated lighting systems like differed shading, the graphics programming has been a valuable learning experiencing.

Future Work

During the summer the team plans on continuing further development on *Third Degree*. The ultimate goal is to publish the game on Steam's game store under the Indie game category. The initial steps towards this goal are to have a very stable version of the game, as well as a highly developed story and corresponding game play. This includes building an adequate number of levels that will allow the story to proceed at a modest rate while at the same time maintaining the game play that was originally planned.

Appendices

Credits

Josh Holland - Art Lead, 2D artwork

Ben Funderberg - 3D modeling

Tom Funderberg - 3D modeling

Mikkel Sandberg - 3D modeling

Ryan Schmitt - Deferred Rendering

Sam Thorn - Sound Lead

References

Akenine-Moller, Tomas, Eric Haines, and Naty Hoffman. Real-Time Rendering. 3rd ed. AK Peters, 2008. Print.

Dunn, Fletcher, and Ian Parberry. 3D Math Primer for Graphics and Game Development. 1st ed. Vol. I. Sudbury: WordWare, 2002. Print.

McShaffry, Mike. Game Coding Complete. 3rd ed. Vol. I. Charles River Media, 2009. Print.
Shirley, Peter, and Steve Marschner. Fundamentals of Computer Graphics. 3rd ed. K Peters, 2009. Print.