Assignment 4 Report

CSC2508

Yasser Khan

Student #

Tanzim Mokammel

996155208

Uncharted Space is an interactive space exploration game. The player takes control of an artificial intelligence that must find his ship in a deserted, barren planet in order to continue on his adventure in deep space.

# Game World Generation

One of the main goals of assignment 4 was to convert the game world from assignment 3 into a sphere. This required modifications to the terrain, character traversal, and camera adjustments. Tanzim focused on terrain and world generation of the project.

Various alternatives were considered in creating a spherical world, including gluSphere, and open source sphere creation algorithms. Upon research, careful consideration, and trial and error, using a folded cube and converting it into a unit sphere proved to be the best choice due to the following reasons:

* Lack of visible artifacts (distortions at the poles)
* The ability to fold the original plane into the cube

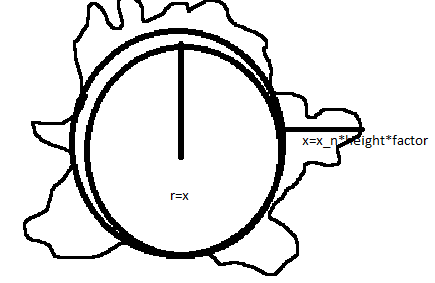
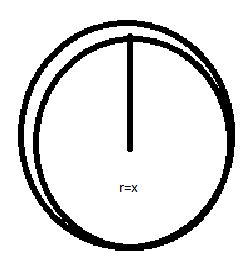
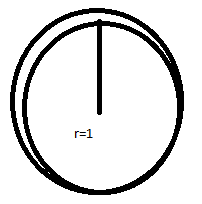
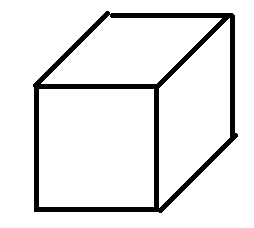
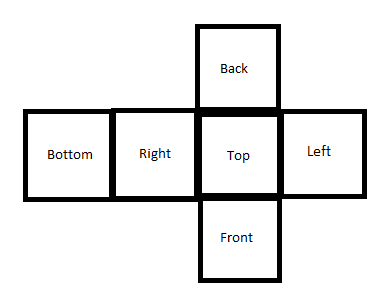
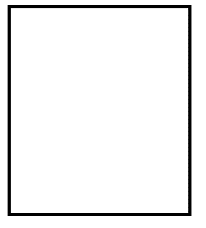
Using this method, the first step was to section the original plane into squares. This was done on a pixel by pixel basis to allow the most control. Once this was completed, only parts of the original terrain were rendered to generate a 2D cube map. The next step was to fold the cube map into a cube, about the origin. There were 2 choices in performing this step: use the glRotate functions outside of the world generation function, or simply perform the rotations on a pixel by pixel basis inside the world generation function. The latter was selected due to improved control, and for containing the world generation into a single, customizable function. Once each subsection of the plane was rotated, translated, and formed into a cube about the origin, it was ready to be transformed into a sphere.

To convert to a unit sphere, the cube’s vertices were normalized about the origin. To retain the shape of the world, the normalized vertices were then multiplied by a selected radius. The normals were also used for the terrain lighting. The next step was to heightmap the sphere. This was done with the following steps:

* Obtain the height value using the plane’s original x-z coordinates
* Apply the height value in the sphere’s normal direction

The heightmap was designed in GIMP, containing 8 bit grayscale values. The heightmap is essentially a cube map, which gets applied to the corresponding faces of the cube converted to a sphere. One constraint with this method is that the borders of the cube must be at zero height to maintain a spherical appearance.

The below diagram shows the world generation process:



Texturing was initially a challenge for this process. However, this was resolved by texture mapping the coordinates of the original plane instead of the rotated sub-planes. Multi-texturing with blending was also attempted. However, this broke the lighting functionality of the world, and further debug was not possible given the time constraint.

The world has a few customizable features to alter visual fidelity and performance. The step size for traversing the height map can be selected, which alters how many squares or triangles are used to draw the terrain. Selection between triangles and cubes are also possible. The number of times a texture is repeated is also customizable. A tiled lower resolution, instead of a single high resolution texture was used in the final game to achieve reasonable performance. The original textures were made seamless using GIMP to achieve a more visually pleasing tiling effect. The user may select between two worlds with distinct heightmaps and textures in-game.

In addition to the main world, there is a textured sphere orbiting the main planet, acting as a light source and allowing a day night cycle. There is also an additional textured sphere encompassing the universe.

During the project, Tanzim learned valuable skills by overcoming challenges. Mathematics for fundamental rotations and translations were strengthened during pixel by pixel transformations on the world. Image editing, height mapping, cube mapping, and texture mapping for world generation were also a valuable skills learned. Traversal of the game world was also a big challenge that is currently improving critical thinking and 3D geometry skills. The next steps for the world generation would be allow heightmapping between borders, achieve a blended multi-textured effect, adding objects to terrain (plants, landmarks etc), and finally improving game performance by only rendering selective portions of the world. Overall, spherical world generation was a challenging, but enjoyable experience.

# Character Generation and Game Mechanics

Yasser continued the work that was started in Assignment 3. He first cleaned up and added more keyframes to the character animation for running. He also fixed issues with some body parts. Next, multiple keypresses were enabled by using a bit array of keys, each element of the array corresponding to a specific key. If the bit is 1, then the key has been pressed. This allows multiple keys to be registered and used in other places in the code in tandem with other keys. Next, he created fuel cells as an in game collectible. The fuel cells are generated at random locations every time the game starts. The amount of fuel cells is also easily changeable by a single variable. The fuel cells are made of four cylinders of different colours, and it is roughly half the size of the player character. Also, they bob up and down slowly as an animation. This was done using a counter and translating every fuel cell a certain amount every frame. Each fuel cell has x y z location and flags such as if it has been picked up. The picked up flag is enabled once the player’s location matches with the location stored in the fuel cell array.

Next, he added enemies. They are just basic cubes, with randomly generated locations every time the game starts, and the amount of enemies can be modified easily by changing one variable. They are called “Sentinels”, and move erratically around the world. This erratic effect was achieved by randomly picking which direction the enemy is going every frame. Like the fuel cells array, each enemy has a flag indicating if the character has touched it. Once the flag is turned on, all fuel cells get returned to the map, and the fuel cell location get randomly generated again. The flag is then kept on until the character moves away from the enemy. The enemies need to be avoided if the game is to be completed.

The ship location also gets randomly generated. A certain number of fuel cells are needed before the ship is able to fly away, unlike in assignment 3 when the player just has to reach the ship. There is a global variable that keeps track of the number of fuel cells the player has. Once the player has obtained the required amount, the ship will fly away when the player reaches it. To display the number of fuel cells remaining, Yasser implemented a drawing function using gluBitmap. Using gluBitmap, any message printed on screen will not be larger or smaller depending on how close it is translated to the camera. This is good for displaying player information. It allows the information to be rendered extremely close to the screen so it does not clip behind the world and character when the camera is rotated or zoomed. Yasser ran into trouble after translating the rendered sentence to the bottom left corner of the screen for aesthetics: when zooming in and out, the text would disappear and reappear at different positions. This is most likely due to the origin moving with the translation to move the print to the corner. This issue was not solved due to time constraints.

Tanzim converted the plane to a sphere as he described. Yasser decided to keep the character traversing on the plane since existing code could be leveraged. He clamped the movement of the character to within the portion of the plane that was being folded into a cube. The first problem that exposed itself was that the character needed to move from edge to edge since the planes become connected once it is “folded” into a cube. To solve this, several edge cases were coded that would move the character to another point once an edge was reached. For example, once the character would reach the edge of the “right” face, the character would be instantly moved to the other end of the plane to give the illusion of constant movement on the sphere.

The next problem was that moving between certain faces would require the player to alter the angle of movement to keep the illusion of constant movement. For example, moving from the left face to the top face would mean that on the plane when reaching the edge that connects the left to top face the character needs to rotate 90 degrees to keep the illusion of constant movement on the sphere. To make these rotations easier, movement was altered to become rotation and forward movement, instead of strafing. This would mean that the character would move in the direction he was facing, and the strafe buttons would just rotate the character about the y axis to change the forward direction. It was then trivial to add the appropriate degree when teleporting between faces.

The final problem encountered when using plane to sphere mapping for character movement was rotating the character appropriately as he moved around the sphere. For example, on the side faces of the sphere the character (and the camera) have to rotate 90 degrees to keep the illusion of being on the top of the surface. Up to this point, the character and camera stay in the regular orientation that the y axis is up. This proved very difficult since at every point the normal of the corresponding point on the sphere would have to be gleaned and used to determine the angle the character needs to be rotated by in the characters x and z axes. This required storing the normals of the sphere at every corresponding point on the plane. This was a significant memory requirement and traversal cost would increase exponentially as the map size increased, which was something we wanted to do. Secondly, the effect was not smooth as we needed to interpolate the angles in between the step size (since the normals were only stored on the step size points).

Due to unsatisfactory illusion of movement on the sphere when traversing the plane, Yasser decided to swap out the plane movement and focus on spherical movement. This proved to be a breakthrough since it was much clearer and smoother to rotate about the origin of the planet instead of traversing in X and Y. Strafing was re enabled, with the W and S keys increasing and decreasing the rotational angle about the X axis (giving “forward” movement) and A and D giving sideways movement (rotation about Z axis). This inherently allowed the camera and character to rotate properly, giving the illusion that gravity and the character are always on the top of the sphere. It turned out to be trivial to move all the enemies, objects, and all interaction code to spherical coordinates. Instead of x y z coordinates, all objects now stored rotational angles. If the character’s rotational angles matched then they are interacting.

A big problem Yasser had to overcome by moving to spherical coordinated movement was attaching the camera to the character. Both the camera and the character need to be rotated together so the movement variable dictating their movement are the same variables. The camera rotations move their axes as well, which affects the axes of the character. The character’s rotations do not affect its axes, so it caused a movement mismatch of the character and the camera. To prevent this, the axes of rotation of the character in the strafe had to be modified by the current angle of the propel variable. This attached the camera to the character properly but resulted in the sideways movement of the character to be always attached to the z axis; when the character reached 90 degrees forward movement, strafing would essentially become rotation. This issue still needs to be solved. Do to this odd behavior of movement, mathematically there are multiple angles that can be used to get to a single point on the sphere. This made object interaction confusing, as there were several times where the character would be physically in the same place as an object but the angles taken to get to that point were different for the character and the object, resulting in no interaction happening. This was worked around by using trigonometry to convert the spherical to Cartesian coordinates for the purpose of interaction code.

Now that traversal was complete, the next problem was traversing the height map without actually being on the height map. This was attempted by converting to Cartesian coordinates and checking the height map at that point on the sphere. This proved very difficult since the height map on the sphere is essentially still a plane but manipulated into a spherical shape. The team is still working on this problem to traverse the height map.

The final task Yasser undertook was to create an intro screen and a completion animation. He built on top of the printing function he created by using glutStrokeCharacter. This OpnGL function allows for text that can be scaled to become larger or smaller. On initialization a flag is set saying that the introduction is happening. When this flag is set, the camera is fixed to a cinematic position and slowly rotated around the planet. Also, several lines of text are printed using glutStrokeCharacter, with the title being scaled to be much bigger. Pressing any key will disable the flag to allow regular gameplay. On game completion (fuel cells obtained and reached the ship), a countdown is initiated. The center of the screen shows a countdown from 3, with each number slowly getting scaled to become bigger. During the countdown, the ship is rotating but not going up. On countdown completion, the ship will start to fly and the camera will be free to move and view the ship until the game automatically closes.

Yasser learned several skills during this project, and assignment 4 specifically. Working around problems and thinking outside the box was one of those skills. Also, memory management and many technical skills relating to pointers and arrays were learned. Game creation and animation was also very interesting and challenging. Overall he enjoyed the experience.