INM376 Computer Graphics

Project report

Leonidas Antoniou – 170040646

01/04/2018

1. Project Overview

The projects theme is a futuristic sci-fi race game. It is based in space, to give the sci-fi feeling, and has space elements decorating the map and give variety.

The game concept is time trial racing. The player has to complete three laps along the created path so he can save the earth from the alien "invasion" and be triumphant. Whilst trying to achieve that he has to avoid obstacles along the path and collect boosts to achieve a better time.

2. Implementation:

2.1 Game Controls

Move forward – "W"

Turn left – "A"

Turn Right – "D"

Change camera view (Top-down view) – "3"

Change camera view (Third person) – "4"

Change camera view (Side view) – "5"

Change camera view (First person) – "6"

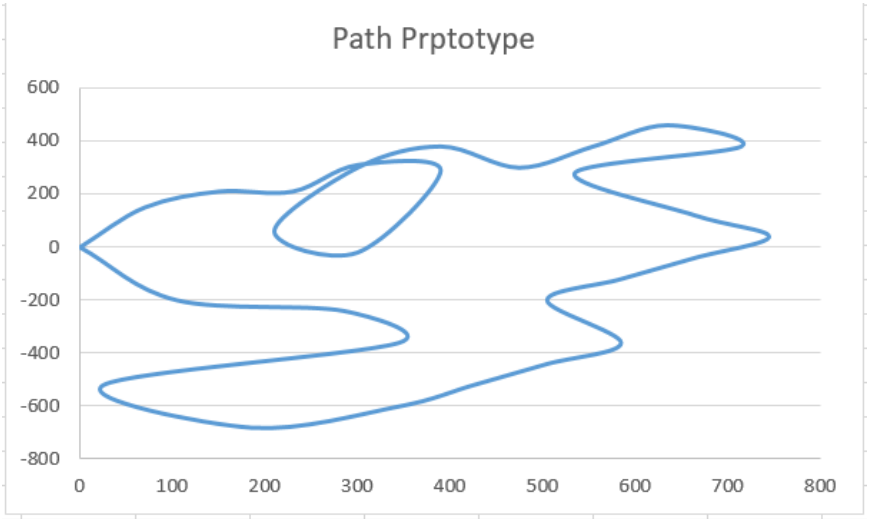
Dark mode/Light mode – "2"

2.2 Features Implemented

2.2.1 Route and Camera

Route

The route was created using Centripetal Catmull-Rom spline algorithm which was partly implemented inside the project template used for the project. A number of control points were set to give the outline of the path. Then the control points are uniformly sampled, meaning to produce a number of samples that are uniformly spaced. They way this works is by computing the lengths of each segment setting the spacing based on the control polygon.



The centreline is created inside the CreateCentreline method by calling the methods for the above actions. A VAO and a VBO are created. The centreline points are added on the VBO along with texture coordinates and the normal coordinates. The VBO is then uploaded onto the GPU.

Afterwards the left and right offsets are created. The offsets are calculated by calculating the Frenet Frame (TNB) and using the N vector to determine the position of the offset point (left or right). Once calculated the points are added in two VBOs, left and right, and the uploaded onto the GPU along with texture coordinates and the normal coordinates. There are two special cases for the offset curves, before adding the points, the first two points are added and at the end the last two are added again as well, the texture coordinates and normal as well. This is so the last quad is connected and rendered correctly.

Inside the CreateTrack method the left and right offsets are added into a single vector in order and are added on a VBO along with texture coordinates and the normal. The last two points are added first before the addition of the rest and the first two points are added again at the end so that the last quad is rendered correctly. Then the VBO is uploaded onto the GPU along with texture coordinates and the normal coordinates. A texture is also bound.

The only part of the path that is rendered is the track by calling RenderTrack method that also binds a texture on the path. The texture is loaded in the CatmullRom constructor. The path is rendered with gl_triangle_strip.

Since the centreline is not rendered an indication of the centreline must be specified. For the purpose an imported mesh is used for the visualization of the centreline. Moreover, the track edges have been demarcated by the use of a primitive, a cube that is created in the class Cube. The cube has correct normal and texture coordinates. The cube doesn't have a texture in this case but is using a toon shader. Moreover, the cube is rotating on its y-axis by multiplying its y-axis with the game time in the update method of the Game class.

Camera

The camera was already implemented inside the project template and needed no further development. The Set method from the camera class was used to set the camera position for the following camera views:

- First person view
- Third person view
- Side view
- Top view

In order to set the camera to the above views the Frenet frame (TNB) was used. The TNB was calculated by firstly calculating the T vector which is the tangent vector which points in the direction of the next point on the centreline. It takes the current position and deducts it from the next position and normalizes the result. Secondly, the N vector which is the normal vector which is the normalised cross product of T with the up vector points, in this case the up vectors are facing up on the y-axis y default. The N vector is used to determine the direction on z-axis. Lastly, the B vector is calculated which is the binormal vector and acts as the up vector and is the normalised cross product of N and T vectors.

For the changing of the view of the camera a string member is defined in game header and initialised in the game cpp as "first" to indicate that the default camera view will be first person view. The value changes by pressing the appropriate keyboard keys assigned for each view. The value is checked by

the ChangeCameraView function, which is called in the update method, and depending on the value of the string member it sets the appropriate camera view by using the TNB frame and Set method of the camera class.

2.2.2 Basic objects, meshes and lighting

Basic Objects

There are two primitive objects created and used throughout the map, a cube and cylinder. The cube is used to demarcate the tracks edges and the cylinder is used as a tunnel in some points of the track.

For the creation of the cube the six sides of the cube were defined in the cube class, the four texture coordinates, one for each edge of each side, and the six normal. The normal coordinates were set based on the direction that each side faces. Everything was then added on the VBO and uploaded on the GPU. Afterwards, the cube was rendered with gl_triangle_strip using six calls to the gl_drawArrays. Also a texture was bound on the cube which was loaded in the in the cube constructor.

The cylinder was calculated based on the cylindrical coordinate system. In the cylinder class the segments of the circle are defined. Looping through each segment the circle is calculated based on the type x = ncos(angle) and y = nsin(angle). Two circles are created based on the y-axis. The normal coordinates are calculated by deducting the point on the y-axis from the circle points and then normalizing the result to get the direction. The texture coordinate is calculated by dividing each segment (which is an integer) by 2*M_PI. The coordinates are then added on the VBO and uploaded on the GPU. The cylinder is rendered with a gl_triangle_strip that connects the points of the two circles. The cylinder is an open cylinder because of its use as a tunnel in the game.

Meshes

There are five meshes loaded in the scene of the game. Every mesh that was imported was first imported inside Blender and has either been textured or had its scale and rotation corrected.

These include:

- *Asteroid*

  The asteroid can be found in various points around the map floating and rotating along its y-axis. The rotation is happening inside the update function of the game cpp. The asteroid is rendered using the discard shader, which will be explained under the advanced rendering section.
- *Sphere*

  The sphere is used as a way to visualize the centreline along the path on which the gameplay occurs.
- *Spaceship (player ship)*
  - The spaceship is used as the player ship, controllable by the player. Its position is controlled by the function PlayerMovement inside the game cpp. The PlayerMovement method changes the position according to which keyboard key is pressed. It calls to functions for the movement, ShipAdvance and ShipStrafe. ShipAdvance method accepts and argument of type double which determines the direction. It defines the speed of the ship and multiplies game time with the speed and adds the value to the y-axis of the ship's travel distance, moving it forward. The same goes for the ShipStrafe method only this time it adds the value to the x-axis of the ship's travel distance. The position of the ship is updated inside the SetShipPosition method which is called in the Update method. Using the Sample

method from the CatmullRom class we measure the distance travelled and the ship position. The TNB frame is recalculated for every sample to determine the spaceship orientation. Then using the N vector for direction the spaceship position is updated.

- *Resonance (observer ship)*

  The Resonance is a ship that follows a circular course and "observes" the scene. The course is set by the SetObeserverPosition. Inside the method a speed for the ship is defined and a radius for the circular orbit. To define the observer position, the same method is used as the calculation of the cylinder (cylindrical coordinate system) so it has a circular course. Moreover the TNB frame is calculated for the observer ship's orientation, is stored in a glm::mat4 member and is multiplied onto the model view matrix.

- *Road lights*
  - They act as spotlights when in dark mode. They are placed along the centreline in a fixed distance.

Lighting

The general lighting was included in the project template. The shader was modified to use the BlinnPhongModel which is used to render spotlights. A struct, LightInfo, is holding the information for the lighting and a list of twelve LightInfo elements was created to hold the various spotlights and also the general light. Both methods PhongModel and BlinnPhongModel were modified to accept a third argument, a LightInfo argument, to define the type of light. BlinnPhongModel method uses the halfway vector which makes the spotlight bigger. Three extra members were added to calculate the exponent, cut-off and the direction of the spotlight.

There are a total of 11 spotlights in the scene, 10 along the path and 1 on the spaceship acting as the header light. The spotlights on the centreline are magenta whereas the one attached on the player spaceship is red.

2.2.3 HUD, gameplay and advanced rendering

HUD (heads up display)

A HUD has been implemented displaying the FPS (framerate per second), time elapsed from when the game started and lap count. The font has been changed to fit the sci-fi theme of the game.

Gameplay

A player controller has been implemented in the project that allows the player to move along the path.

Collision detection was not implemented due to time constraints.

Advanced rendering

At least two shader techniques have been implemented, these include:

- *Animation using discard in the fragment shader (Discard shader)*

  The Discard shader reads the colour from the texture that is bound on the object and uses a timer to create a disintegration effect, by returning without colour. This is achieved by setting a timer, the game time is passed as a uniform, and calculate the fract of the timer. The fract returns the fractional part of the float the argument. The fragment shader checks if the red value from the colour is smaller than the fract of

the timer and uses discard if it is. When discard is used the fragment shader returns without setting a colour.

- *Toon shader*

  The toon shader limits the output colour based on the intensity. Intensity is calculated inside the vertex based by adding the diffuse and specular intensities. The intensity value is then passed to the fragment shader where it is used to check if the intensity is lower or higher than the specified values, to determine if dark, medium or light colour will apply. This creates a toonified rendering of the object.

3. Scene assets

Models

- Sphere - https://www.cgtrader.com/items/82986/download-page (21-03-2018) (Royalty free license)
- Simple spaceship (player spaceship) - https://www.cgtrader.com/items/798624/download-page (21-03-2018) (Royalty free license)
- Asteroid - https://www.cgtrader.com/items/703528/download-page (23-03-2018) (Royalty free license)
- Resonance (observer ship) - https://www.cgtrader.com/items/622446/download-page (23-03-2018) (Royalty free license)
- Road light - https://www.cgtrader.com/items/655962/download-page (19-03-2018) (Royalty free license)

Skybox

- Space skybox - http://www.custommapmakers.org/skyboxes.php (20-03-2018)

Textures

- Road texture - https://www.pinterest.com/pin/25755029098508268/ (20-03-2018)
- Sphere texture - http://www.onlygfx.com/wp-content/uploads/2016/08/dark-red-watercolor-4.jpg (21-03-2018)
- Cylinder texture - https://opengameart.org/content/4096-scifi-hex-tiles-pbr-texture (22-03-2018) (CCO license)
- Pick up box - https://llexandro.deviantart.com/art/Sci-Fi-Texture-158-549695018 (25-03-2018)

For two of the models, Sphere and Simple spaceship, Blender was used to fix rotation and scale. Also to apply textures.

4. Reflective Feedback

In the time provided, the finished project has all the basic features and functionalities that were requested. The path created has a nice variety and the length is just right but the scene although it is populated by some meshes it could have more meshes or primitives that would create a better variety and not be so dull. The textures provided fit the game well because they give it a sci-fi feeling as intended although multi-texturing could have been implemented for a better look. Further types of lights could have been implemented. More advanced rendering techniques could have been

implemented to make the game visually more appealing such as bloom effect, shadows, blur. Apart from the weaknesses of the game, it is a basic sci-fi time trial racing game that has achieved the look and the functionality.