FruitRunner

Gameplay Tasks

- *3D geometry*: To load the objects we created in blender into our game, than implemented an object loading method, that takes an .obj (wavefront) file as input and stores the values from that file into vectors for positions, normals, UVs and indices. As Blender does the indexing a little different (using separate indices for positions, normals and UVs), these lists must be rearranged so that only one index vector is needed. After these vectors are properly arranged, We used them to create a new GeometryData object, which I could then use further to create a drawable Geometry object. To increase performance, the bullet collision shapes are not created using the blender mesh, but consist of simple geometric shapes like boxes, cylinders, and spheres.

  Tutorials used: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/

- *Playable:* See Controls and Camera sections below
- *Framerate Independence:* Game is framerate independent at 60 FPS (can drop at some parts of the game depending on the devices capabilities).
- *Win/Lose Condition:* The goal of the game is to reach the end of the course without losing all the lives, in a quickest way possible. Lives are lost when colliding with the obstacles (see section "Moving objects") or when touching dangerous terrain (on the ground). For more information see the "Collision detection" section.
- *Intuitive Controls:* Main movement is done by WASD keys that control the direction of movement. SHIFT toggles sprint and SPACE is used for jumping
- *Intuitive Camera:* It is a 3rd person fixed distance camera that follows the player. Using the Mouse, the direction at which the player is facing can be changed, thus controlling the cameras rotation angle, pitch, and yaw.
- *Textures:* All objects in scene have texture associated with them. For more information, see sections "3D geometry" and how Physically Based Shading was implemented.
- *Moving Objects:* Apart from the player and the camera, there are numerous moving obstacles that the player must avoid (saw blades, maces, blades, axes…)
- *Documentation:* You are reading it
- *Adjustable Parameters:* Refresh rate, Fullscreen, Brightness and Screen Resolution can be adjusted in the Settings.ini file.
- *Collision Detection:* This was achieved using the "Bullet" Physics engine. Direct implementation is the class GameObject.cpp, that handles all the geometry in our game as rigidBody Bullet objects.

Effects

Environmental Mapping (SkyBox)

Implemented as a way of generating background graphics for our game by using a cubemaps. Images are loaded using the stb_image library. The Skybox is always cantered around the player giving the impression of an extremely large surrounding, and takes input from the camera position to display the textures properly.

Tutorials used:

- https://learnopengl.com/Advanced-OpenGL/Cubemaps
- https://www.youtube.com/watch?v=QYvi1akO_Po
- https://www.youtube.com/watch?v=Z4I_EEskNK4&t=113s

CPU Particle System

Used for falling "rust" particles to add the atmosphere to the game. To improve the performance, the particles are only rendered in a small area around the player and not on the entire level. If you run fast enough, you can "out run" the particle cloud, allowing one to look at the sky and slowly see it coming back to player. Each particle has their own position and is represented as an instance of one single base mesh, and we update a small buffer each frame (the centre of the particles) and not a huge mesh.

Tutorials used:

- http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/
- https://www.3dgep.com/simulating-particle-effects-using-opengl/#The_ParticleEffectRender_Method
- https://learnopengl.com/In-Practice/2D-Game/Particles

Physically based shading:

We implemented Physically Based Shading using the Cook-Torrance model. The shader works by using textures that are available freely on https://freepbr.com. Those textures use an albedo texture, a roughness texture, and a metallic texture. We are using those values to get the right roughness and metalness Values for each fragment, according to the UV coordinates of that fragment. In the game you can see physically based shading applied to the stone platforms, the tiled ground platform, the small wood platforms, and the spinning sawblades.

Tutorials used:

- https://en.wikipedia.org/wiki/Specular_highlight#Cook.E2.80.93Torrance_model
- https://learnopengl.com/PBR/Theory
- https://learnopengl.com/PBR/Lighting
- https://www.youtube.com/watch?v=5p0e7YNONr8&ab_channel=BrianWill

Hierarchical animations:

We used hierarchical animations for the maces, as well as the spinning blades. Both objects have a “stand” on which they are mounted. This stand moves along the x axis and the mace/blade takes the movement of their parent object, which is the stand, and applies it to itself, while also adding rotation. That way, the objects move together along the x axis, while the child object can rotate freely. As mentioned before, you can see this effect applied on the spinning maces as well as the spinning blades.

Tutorials used:

- https://tuwel.tuwien.ac.at/pluginfile.php/2415211/mod_page/content/32/Animation_SS18.pdf

Important notes:

- Currently the player has 5 lives, and one is lost when colliding with the spinning maces, axes, blades, etc. Additionally, the game is over if the player falls to the most bottom platform (ground) or falls completely off the level.
- There is no HUD implemented yet (Planned to be added for the Game Event), so the information about whether the player has been hit or not, or if he/she won or lost a game is displayed in the console
- Additionally, we plan to add (For the Gaming Event) different models to the Banana, each smaller than the other, to simulate parts of the Banana being cut of, allowing one to visually track the number of remaining lives