Documentation

Group/Game Name: Night Souls

Brief description of implementation:

You are in an Arena with an opponent. The Camera is centered on the Player Character from a top down view (45 degrees). You can attack, and the goal is to kill the Boss, while avoiding being hit by him.

Controls:

WASD: Move the character

Q: Drink flask to heal

Left Mouse Button: Attack

Scroll Wheel: Zoom camera in and out

Additional libraries:

- freetype: to render text

Gameplay:

Mandatory:

- **3D Geometry:** We implemented an object loader that uses .obj files and the connected .dds files to create a Geometry object and a connected texture material.
- **Playable:** You can move and fight an opponent.
- **Advanced Gameplay:** The Boss makes decisions when to attack based on a random value and based on how close he is to the player. The player’s attacks consume stamina which slowly regenerates over time.
- **Min 60 FPS and Framerate Independence:** Framerate can be shown with “F4 , and delta time is used to calculate the movement.
- **Win/Lose Condition:** When you hit the opponent he loses life and when his life reaches 0 you win. When you are hit by the opponent you lose life and when your life reaches 0 you lose.

- **Intuitive controls:** WASD controls the character movement and a mouse click lets you attack. “Q” to drink a life flask.
- **Intuitive Camera:** The camera follows the player, but can be switched to a free camera by holding “C”. The camera can also be zoomed by scrolling.
- **Illumination model:** There are multiple light sources . Each imported object also imports normal vectors and material properties for the textures.
- **Textures:** The imported objects also import the .dds textures.
- **Moving Objects:** The player can move and the opponent moves around automatically.
- **Documentation:** This.

Optional:

- **Collision Detection (Basic Physics):** The player, the player attacks and the opponent have axis aligned collision boxes.The AABB method is used for the collision detection. When the player collides with the opponent the player loses health. When the players attack collides with the opponent, the opponent loses health. The Bounding boxes of the player, the boss and the player’s last attack can be visualized by pressing “F3”.

(https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection)

- **Heads-up Display:** A HUD displays the player’s health, stamina and the amount of life flasks he has left. The HUD also displays the opponents health.

Effects:

Advanced Modelling:

- **CPU Particle System:** Particle System that is initialized with certain variables. They are updated on the cpu and rendered on the gpu using instancing (also lerps color and size in gpu)

Terrain:

- **Tessellation from Height Map:** A heightmap is read from the grayscale values of an image file. The corresponding geometry is created and the player and the boss move on top of the terrain. The player and the boss can't move over a specific height value and therefore can't move out of the map.

Texturing:

- **Procedural Texture:**

Torch Wood Texture:

Hash: Pseudo Random number generator

Noise: Noise based on position

Fractal Brownian Motion(fbm): Layering noise to make it look more natural.. (https://www.youtube.com/watch?v=cWiLGZPwXCs)

Wood Grain: sinus based on fbm and added knots also based on fbm.

Shading:

- **Physically Based Shading:**

The Texture Fragment Shader uses PBR to render the lighting. It uses roughness, metallicness, and albedo to calculate more realistic lighting.

(https://www.youtube.com/watch?v=XK_p2MxGBQs)

The floor and enemy are non Metallic and the player is Metallic.

- **Simple Normal Mapping:**

A normal map texture is read from a file and passed to the texture.frag shader. There the normal value is read from the texture and taken for the calculation of the normal vector. (The normal mapping can be toggled on and off by pressing F5)

https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Post Processing:

- **Bloom:**

I added a Framebuffer i use to render the normal image onto a texture and another texture that only receives the brightest pixels. Then i blur the bright texture. Finally the 2 textures are added together and rendered onto a plain that covers the screen.