Documentation DisCharge

Brief Description

The Game is based on the ECG_Framework an therefore OpenGl with C++. The structure of the ECG-Framework is still visible but we added a lot of custom classes like the PlayerCamera and Player. We also implemented a Model and a Mesh class to support the data that the Assimp Library puts out. We also added PhysX as described in the tutorial and implemented collision detection of our Player and our created platforms. We then implemented a PBR-Shader and got some free to use Textures and UV-Mapped our Models and Geometries with them. The Win condition is hitting the last platform while the Lose condition is discharging the battery completely or falling down into the ocean. There are a few checkpoints that you can reach throughout the level and then restart from these checkpoints (R) instead of restarting completely from scratch (O). The sky and ocean get darker the less energy you have signaling that you have to either reach a checkpoint soon or hit a power up. When you reach the last platform the background turns green to signalize that you've won the Game. After each checkpoint you have to master a new kind of gameplay mechanic to pass the level flawlessly.

Features

- Textured Objects (All of our objects inside the game have textures for example the robot and the ball that rotates below the robot, the platforms, the planets etc. those textures are either created by us or from sources where they are free to use for our usecases.)
- External Models (Created in Blender by ourselves (Robot, Planets, Ocean Grid).)
- Moving PlayerCamera
- Moving Objects (Ball, Robot, Pickup, Planets)
- Collision Detection Between Player and all generated Geometries (using PhysX)
- Free Movement of Player
- Illumination Model with multiple light sources (colored lighting to signalize checkpoints and pickups)
- Hierarchical Animation of the Ball and the Player
- Energy is discharging while moving with extra costs for jumping, and rechargable with pickup.
- Frame Independency (400+ Frames on our current Setup)
- Win/Lose Condition (Green Sky signalizes win, Wireframe mode signalizes lose)
- Adjustable Parameters via settings.ini

Effects

- Vertex Shader Animation (Ocean Waves recomputing normals correctly according to the sine functions used)
- Procedural Texture (Conveyor Belt textures are generated procedurally and animated aswell)
- Physically Based Shading (All elements but the conveyor belts use PBR Shading)

Controls

The Character (Robot) is controlled in a very traditional fashion.

- **Mouse Movement** rotates the Player and the Camera
- **W** moves the Player forward
- **S** moves the Player backwards
- **A** moves the Player to the left
- **D** moves the Player to the right
- **Space Bar** lets the Player jump
- **R** resets the Player to last Checkpoint they reached
- **O** resets the Player to the Start of the Game
- **Escape Key** Closes the Game
- **F1** Toggles the Wireframe View
- **F2** Toggles the Backface Culling

Tutorials used

- ECG Tutorials
- CG PhysX Tutorial
- https://ogldev.org/
- https://learnopengl.com/
- https://learnopengl.com/Model-Loading/Mesh
- https://learnopengl.com/Model-Loading/Model
- https://learnopengl.com/Model-Loading/Assimp
- https://learnopengl.com/Advanced-Lighting/Normal-Mapping
- https://learnopengl.com/PBR/Theory
- https://community.khronos.org/c/opengl-general/34
- http://www.opengl-tutorial.org/miscellaneous/an-fps-counter/
- https://forums.developer.nvidia.com/
- https://www.pbr-book.org/3ed-2018/Texture/Solid_and_Procedural_Texturing

- http://learnwebgl.brown37.net/10_surface_properties/texture_mapping_procedural.html

Libraries

- PhysX (https://github.com/NVIDIAGameWorks/PhysX)
- Assimp (http://assimp.org/)
- GLFW (https://www.glfw.org/download)
- GLEW (http://glew.sourceforge.net/)
- GLM (https://glm.g-truc.net/0.9.9/)
- irrKlang (https://www.ambiera.com/irrklang/)