Lost in Abyss

Story of the Game

One day you find yourself stuck deep in an abyss. You have no idea what happened and lost all memories of your past self. The only thing you know is to go back up to the surface. But beware, time is running out!

Gameplay and Controls

You can control the character from the first person perspective, so you can run and jump through the environment. If the timer runs out, the game is over and you have to start over.

Controls:

- WASD: Movement of character
- Space: Jump
- Mouse: Rotate camera
- ESC: Quit
- F1: Toggle wire-frame mode
- F2: Toggle back-face culling
- F3: Toggle heads-up display
- F4: Toggle normal shading
- F5: Toggle lights (only the point light under the base platform is activated)
- F10: Restart Game
- F11: Full-Screen

Brief Description of the Implementation

The main class is Main.cpp, where all needed assets are loaded and initialized, such as shaders, materials, lights, objects and the physics engine. The scene is loaded as a whole from a blender file, which consists of the platforms and the outer hull, the balls and the cubes at the bottom are simple geometry objects. The light cubes are simple geometry objects as well, colors and positions are set according to point lights. The render loop polls the user input, handles the physics simulation and basic gameplay, renders the objects and updates the video textures. To calculate shadows objects have to be drawn to the shadow map first, only afterwards the normal draw method is called.
Collision detection is handled by Bullet Physics Engine, which is imported as an external library. Different classes such as BulletBody.cpp and BulletWorld.cpp are responsible for the implementation.
Special functions are encapsulated in their respective classes, different types of textures are handled in the Texture.cpp class, except for shadow texture which has its own class. Loaded models are handled in the Modelloader.class, whereas the player is encapsulated in the CameraPlayer.cpp class and physics functions are handled in the bullet folder.

"Features" of the Game / Implemented Effects

Compulsory Gameplay:

- 3D Geometry
    - The environment (platforms, plants and hull) are imported as a whole as a blender object. For each point light a light cube is generated, the cubes at the bottom and the walls behind the screen have normal map textures attached.
- Playable
    - The character (1st person view) can be controlled by WASD+SPACE keys
- Advanced Gameplay:
    - The player has to jump from platform to platform to reach the top most platform to win the game and also before the time runs out.
- Min. 60 FPS and Frame Rate Independence:
    - The game runs with at least 60 FPS currently and is frame rate independent.
- Win/Lose Condition:
    - You win the game if you touch the top level of the platforms and lose if the time runs out before.
- Intuitive Controls
    - The control is similar to controls used in real games, for example holding a key longer creates a continuous effect, such as movement.
- Intuitive Camera
    - The camera is the view from the player and can be moved freely with regards to the physics engine (gravity).
- Illumination Model
    - There is one directional light and several point lights, which can be seen as light cubes. Textures of objects are either assigned manually in the main.cpp file or initialized during the loading process of blender objects. For each object normal vectors are provided as well. You can turn the lights off with F5, to only see the white point light under the base platform.
- Textures
    - All objects have textures, except for the light cubes.
- Moving objects
    - The player itself moves and also balls at the base of the environment are moving by collision.
- Adjustable parameters
    - The settings.ini file (assets/settings.init) is used for adjustable parameters.

Optional Gameplay:

- Collision Detection
  - Collision detection is covered by the Bullet physics engine.
- Advanced Physics
  - Collision callback is used to detect the win condition (collision with the win platform) and physically plausible collision resolution is implemented (objects collide with each other and roll away in opposite directions).
- Heads-Up Display
  - Simple text display implemented with FreeType and can be toggled with a key.

Effects:

- Lighting (Shadowing Mapping with PCF)
  - Directional lights are taken into account in the shadow calculation which in result projects shadows in the scene. Shadow acne is fixed by applying a bias, whereas Peter panning is by using front face culling when rendering the depth map. Furthermore, PCF is used to smoothen the edges.
  - https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping
- Texturing (Video Texture)
  - There is a screen at the most top platform and one at the base platform, the one at the bottom is a 8 second video with 24fps. Both videos are updated in real time. At first images are loaded to a data structure and afterwards an image texture is created (similar to image texture). Depending on the time and framerate a different image from the data structure is played.
  - https://learnopengl.com/Getting-started/Textures
- Post Processing (Bloom/ Glow)
  - The point lights (as light cubes) glow, also bright regions will outshine darker regions. At first a separate framebuffer has been implemented (first link) and afterwards draw a quad which spans over the entire screen. Using the scene quad and the blurred brightness texture as inputs, those two textures are combined (second link). The bloom effect is better visible with less lights activated, you can “turn” the light off with F5, only the white point light under the base platform is activated.
  - https://learnopengl.com/Advanced-OpenGL/Framebuffers
  - https://learnopengl.com/Advanced-Lighting/Bloom
- Shading (Simple Normal Shading)
  - The moveable cubes at the bottom and the wall behind the screen at the base platform have textures with normal maps attached, this effect can be toggled with F4. The shading is also better visible with less lights activated, you use F5 to toggle the lights.
  - https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Additional Libraries

- Bullet: https://github.com/bulletphysics/bullet3/releases
- Assimp: https://www.assimp.org/
- Freetype: https://www.freetype.org/