Island Siege

Features and implementation details

****Same as Submission 1****

- Randomly generated terrain with mountains and flat regions. Erosion is used to make the terrain look more natural.
- Physics for simulating interactions between enemies, buildings, projectiles and terrain. This is done in a separate thread to not block other logic.
- The logic for enemies, buildings, projectiles is also done in a different thread. This includes things like aiming, spawning enemies and projectiles. The main reason why another thread is needed is because for each building which can shoot, the area where shooting is possible is precalculated. This involves casting rays in all directions to check if, for example terrain is blocking a shot.
- The cannons predict enemy movement and shoot at the point where the projectile is most likely to intersects with the enemy. This prediction includes bullet drop.
- Pathfinding for the enemies is done by calculating a grid of movement directions. The calculations include things like the steepness of the terrain, so that enemies don´t just walk in a straight line to the nearest target. Walls also change the pathfinding so that enemies try to walk around them if this doesn't take too long. Fully calculating this movement grid can take around a second and it has to be recalculated if a building gets placed or deleted/destroyed therefore a different thread is used to do this.
- Rendering the game is done in the main thread and doesn´t include computationally heavy logic. Only things like GUI interactions or camera movement are done in the rendering thread. This means that the other logic running in the above-mentioned threads doesn't have much effect on the rendering performance.
- Creating a new world is done in a temporary thread to not interfere with the render thread. This can be seen by the loading status.

Used libraries

All libraries except vulkan are stored in a git submodule to separate the implementation with external dependencies. Following libraries are used:

- **Bullet Physics SDK:** Used for physics. https://github.com/bulletphysics/bullet3
- **Assimp:** Used for 3D model importing. https://github.com/assimp/assimp
- **GLFW:** https://github.com/glfw/glfw
- **GLM:** https://github.com/g-truc/glm
- **GLI:** https://github.com/g-truc/gli
- **ImGui:** Used for the GUI. https://github.com/ocornut/imgui
- **nlohmann json:** Used for loading configurable parameters. https://github.com/nlohmann/json
- **LYGIA:** Used for procedural portal effect https://lygia.xyz/

Gameplay

When a world has been created, an intro is played to showcase the planet. This intro can be skipped with ENTER and ends above the main castle. The camera can be moved with WASD. Flying up and down can be done with Space and Ctrl. Shift increases the movement speed.
On the bottom left the main toolbar can be seen. Using this toolbar, buildings can be selected and placed. In addition, the sell, repair, and upgrade tool can be used.
On the top right the next wave can be started or the currently running wave with the remaining enemies is displayed. During a wave the player is not able to change the base in order to not abuse e.g. the repair tool.
On the top left the game can be paused or a speed of 1x, 2x, 4x can be set. Note that speeds above 1 with too many enemies can overload the physics.
The game can be won by completing wave 15 without losing the main castle. Otherwise, the game is lost.

In the settings menu following things can be changed:

- Mouse sensitivity
- Cheat mode (Infinite money, spawning enemies, no restrictions)
- Window Settings:
    - Fullscreen or Windowed
    - Resolution (In windowed mode this can only be changed by resizing the window)
    - Refresh mode (V-Sync or Unlimited)
    - Refresh rate (Only in Fullscreen and when changing resolution away from native)
- Shadow quality
- Enable/Disable Frustum Culling (Also with F8)
- Opening the debug window (Stats like framerate and rendered object count can be seen)

In the debug window, some stats like framerate are visible. Also, frustum culling can be shown by offsetting the camera.

Compulsory Gameplay notes:

- For the portal frame a texture is used.
- Brightness can be adjusted by changing the exposure in the sky settings. This can be accessed in the debug window. For the change to take effect “Use optimized shaders” must be disabled.

Following optional gameplay features have been implemented:

- Collision Detection (Basics Physics)
- Advanced Physics
- View Frustum Culling (Can be seen by enabling showcase in debug window or object render count)

Effects

Following effects have been implemented:

- Shadow Map with PCF
- Vertex shader animation (Ocean)
- Procedural Texture (Portal effect)
- Contours via Edge Detection (Hovering a building when no tool is selected)

Additional effects:

- Gerstner waves with reflection of the sky and refraction of underwater terrain. Water also implements light scattering and absorption. Water mesh is generated by projecting a mesh onto the planet and offsetting the vertices in the shader.
- Sky with one-time scattering atmosphere. This is also used for ambient lighting.
- Stars implemented by skybox which is blended into the atmosphere.
- Shadow map also implements cascades to improve quality over varying distances.

Additional Remarks:
The sun angle can be changed by using the left or right arrow key. This can be sped up by additionally pressing Shift.