Complications On The Blasted Grog - Submission 1

Implementation

- **GameActor.h**: Stores stats for an actor (i.e. player, enemy) like name, hit points and speed
- **RenderedObject.h**: Data Structure for 3D Objects having the following members
    - mesh: Mesh - The mesh to be rendered
        - **Mesh.h** stores vertex and index buffers
    - transformation: mat4 - transformation matrix of rendered mesh
    - color: vec4 - color of the mesh
    - matDataBuffer: VkBuffer - Buffer for color
    - desc: VkDescriptorSet - Mesh's descriptor set
    - material: vec4 - (ka, kd, ks, alpha)
    - texture: ImageInfo* - texture of mesh
    - flags - phong shading, is ui element and is active (= should be drawn)
- **GameObject.h**: Implementation of interactive RenderedObject, lets the Meshes move

    -
      ```
      struct GameObject
      ```

      This is the base struct for every implementation and includes the following members:

        - _name: String - The name of the GameObject
        - _renderedObjects: RenderedObject*[] - The List of the actual Meshes of the GameObject
        - _transformation: mat4 - The transformation Matrix of the GameObject (combination of translation, rotation and scale vectors)
        - objType: ObjectType - enum value of ObjectType for collision detection
        - Direction Vectors for movement control

      Every GameObject and Implementation of GameObject has a `parent` and `children` hierarchy, every child's transformation is dependent of the parent's transformation. Every GameObject can have multiple children and one parent.

    -
      ```
      struct PhysicsObject : GameObject
      ```

      Implements Basic Physics using Bullet.

      The first implementation of GameObject, which includes logic for the physics implementation via bullet library. It has two extra members, which are a pointer to a `btRigidBody` and a `btTransform` which stores the actual transformation of the RigidBody. The `move()`-function applies a given force to the RigidBody, whilst the `step()`-function updates the `btTransform` with the calculated values of the physics simulation.

    -
      ```
      struct CharacterObject : GameObject
      ```

      Implements Movement using Bullet.

    The second implementation of GameObject, which implements logic for an actor in the game. A CharacterObject has a `btPairCachingGhostObject` member, which is responsible for collision detection and physics simulation, which is used by the `btKinematicCharacterController`. The Kinematic Character Controller is a simple way to implement a working character, because it includes useful functionality for jumping or moving around for example.

  -
    ```
    struct ParticleObject : GameObject
    ```

    Implements Particle Simulation using ParticleSystem implementation described now.

    The used ParticleSystem calculates the new positions, colors and sizes on the CPU and sends the data of alive particles to the GPU to process.

- **ImageInfo.h**: Stores data for Image Loading
- **ImageSource.h**: Describes the members for image loading using stb_image.h
- **ImageSource.cpp**: Implements the members from **ImageSource.h** and loads in images.
- **ROMan.h**: Describes members for creating 3D Objects and loading 3D Objects from files using Assimp
- **ROMan.cpp**: Implements the following members:
  - `create_rendered_object()` - Creates a RenderedObject (= 3D Object Data Structure) and stores its buffer.
  - `create_ui_object()` - Creates a RenderedObject with 2D Texture and UI Flag
  - `destroy_rendered_object()` - helper function for cleanup
  - `load_rendered_objects()`

    This function loads a 3D Object from file using the Assimp Importer and converts every loaded Mesh to a RenderedObject.

  - `getAllRenderedObjectsFromNode()` - helper function for converting loaded Objects from a `aiNode` to `RenderedObject`s
- **LUAHelper.h**: Describes structure for basic LUA loading, can load files, read ints, strings, floats and table values
- **LUAHelper.cpp**: Implements the functionality of LUAHelper.h
- **FunctionLUAHelper**: Implementation of LUAHelper for processing LUA functions. Usage: **Enemy.h**
- **Enemy.h**: Implementation of CharacterObject which tracks the player's position and moves the Enemy Actor accordingly. Enemy Movement is controlled in external **Enemy.lua** Script.
- **ParticleSystem**: The Particle System calculates the Particle Data on the CPU and sends it through three Uniform Buffers (one for particle position and size, one for particle color and one for camera data (view, projection)). It renders up to 1000 instances of the same squad with different data using instanced draw (vkCmdDraw's third argument - see draw_particles function in main.cpp).
- **Animation Vertex Skinning**: The RenderedObject Manager uses AssImp to find out if a Mesh contains bones and if so load the mesh with all data with MeshSource::aiSkinnedMeshToMesh() function. Update animation in game loop by adding up the deltatime and calling ROMan::update_rendered_object_animation() function.

Features

- Camera movement: Mouse
- Toggle Mouse Mode: Left ALT
- Move Character: WASD
- Walk faster: Left SHIFT
- Attack: F
- Jump: Spacebar
- Exit Game: ESC

The basic combat is implemented very minimally. You can defeat the enemy by hitting them with the sword multiple times.

Implemented compulsory Gameplay

- 3D Geometry - Player Model and Ship Model from opengameart.com
- Playable - Simple Movement implemented
- Advanced Gameplay - Enemy can be damaged, when enemy dies game closes
- Min. 60 FPS and Framerate Independence - Deltatime is calculated using std::chrono library
- Win/Lose Condition - Either you kill the enemy [WIN] or enemy kills you [LOSE]
- Intuitive Controls
- Intuitive Camera - Changeable in settings.ini file
- Illumination Model (Press M for moving lights)
- Textures - UI Elements have a texture
- Moving Objects - Enemy, Player
- Adjustable Parameters - settings.ini
- Documentation

Implemented optional Gameplay

- Collision Detection (Basic Physics)
- Advanced Physics - Collision Callback, Push brown box with sword (Press F)
- Scripting Language Integration - LUA Binding, edit enemy's movement behaviour in `assets/lua/Enemy.lua` at runtime
    - See BlastedGrogSolution\src\LUAHelper.h for the implementation
- Heads-Up Display (functional Healthbar)

Implemented Effects

- **Advanced Modelling**: CPU Particle System - Ship Particles, Jump Particles, Enemy Hit Particles
    - BlastedGrogSolution\src\ParticleSystem.h & BlastedGrogSolution\src\ParticleSystem.h
    - BlastedGrogSolution\assets\shader\particleShader.vert.glsl & BlastedGrogSolution\assets\shader\particleShader.frag.glsl
- **Animation**: GPU Vertex Skinning
    - BlastedGrogSolution\src\MeshSource.cpp::aiSkinnedMeshToMesh()
    - BlastedGrogSolution\assets\shader\skinnedVertexShader.glsl
- **Shading**: Cel Shading (*BlastedGrogSolution\assets\shader\phongFragShaderCel.glsl*)

Libraries used

- Assimp for 3D Object loading
- Bullet for physics simulation and collisions
- Irrklang for Audio
- Lua not used currently
- stb_image.h for 2D Image Loading
- Freetype not used currently
- **Vulkan** Graphics Library