

## PTVC Submission 2 - Documentation

### **Admin:**

Group/Game Name: Dunes of Destiny

Github Link: <https://github.com/faberichfuchs/DunesOfDestiny.git>

Students: Sofia Franzelin (12125512), Fabio Fuchs (12024743)

Genre: Jump'n'Run/Adventure-Puzzle (OpenGL)

Goal: The treasure hunter must collect three artifacts to find the secret treasure in the hidden pyramid while avoiding sinking into quicksand.

### **Game Idea and Content:**

**Story:** Tomb raider Rafael Zahir is on a quest for treasure in the hidden pyramid. Although they have already pulled off major heists, this one is meant to seal their fate as a legendary thief. Will they become the most successful adventurer in history? Or will the Dunes of Destiny swallow them, just as they have many before?

**Gameplay:** Rafael Zahir can run and jump in all directions. With the Torch of Truth, they can find magical objects. Their unparalleled instincts always lead them in the right direction once they have gathered a few clues.

### **User Interaction:**

- WASD: Move Rafael Zahir
- Mouse: Adjust camera orientation
- Spacebar: Jump
- ESC: Exit the game
- F1: Toggle wireframe mode
- F2: Toggle back-face culling
- F3: Toggle frustum culling
- F4: Toggle Frustum culling logs
- R: Reset the game to the beginning
- 0 and 1: Toggle win/lose HUD
- P: Toggle point light
- M: free the mouse from controlling the camera

### **Effects:**

#### Gameplay:

#### Mandatory:

- 3D Geometry (6 points):
- The game includes 3D geometry elements such as a pyramid as the target area, chunked surface Geometries generated using a heightmap, a character model for Rafael Zahir, three artifacts (scroll, dagger, and potion), and the Torch of Truth, which serves as a light source and is held by Rafael Zahir. For this, different lists are created

and with different “generate-methods” towards the end of the Main class and use of the Geometry class the different geometries are created.

The player model, the three artifacts and the torch are 3D models, loaded with the assimp library in the Player and ModelLoader classes, the ModelLoader is used to read the model out of the according scene.glTF and in Player scaled and rotated. The models are in glTF format. The assimp library is located under lib/assimp-master and was downloaded here: <https://github.com/assimp/assimp>

The 3D models are located under assets/models/.

- Playable (3 points):

The game is playable with a third-person camera that follows the character and can be moved. Character movement is controlled using WASD.

- Min 60 FPS and Framerate Independence (3 points)
- Win/Lose Condition (3 points):

The win condition is to collect all three artifacts, enter the pyramid, and steal the treasure. The lose condition is sinking into quicksand.

To win: search and collect all three artifacts moving in the game world without stepping into quicksand. When all three artifacts are collected, the pyramid appears, as soon as the player collides with the pyramid, thanks to collision detection the win HUD appears.

To lose: When moving in the world and stepping into the quicksand, thanks to collision detection a small animation (isCutscene in renderloop) appears with the player sinking into the quicksand. Afterwards the lose HUD appears.

- Intuitive controls (2 points):

Controls are intuitive using WASD and the mouse for camera control.

- Intuitive Camera (2 points):

The camera can rotate and move up and down, controlled by mouse movement. The mousewheel controls the zoom.

- Illumination model (2 points):

The illumination model consists of a primary light source (the moon in the sky) and the torch in the player's hand. The point light in the torch is pulsating lightly.

- Textures (2 points):

Textures are applied to the sand on the ground and the pyramid. And of course the 3D models already have a texture.

- Moving Objects (2 points):

Moving objects include the player model with the torch and ominous moving sand Quicksand Surface Chunks(Vertex Shader Animation).

The three artifacts are spinning around their origin.

- Adjustable Parameters (1 point): Resizable and fullscreen.

#### Optional:

- Collision Detection (Basic Physics) (6 points):

The game will feature collision detection with basic physics, ensuring the player moves and jumps across the terrain without falling through the ground or leaving the map.

- Advanced Physics (4 points):

Advanced physics will cause stepping into quicksand to trigger a game-over event and walking into the pyramid to cause the win event.

- View-Frustum Culling (6 points):

View-frustum culling renders only objects inside the camera's view. The culling can be toggled on and off using the F3 key.

- Heads-up Display (4 points):

A win heads up display appears when winning (going inside the pyramid after collecting artifacts), and a lose HUD will appear when losing (stepping into quicksand or going inside the pyramid before collecting all artifacts). The lose HUD can be made visible during the game when pressing key 0 and the winHUD when pressing key 1.

For the HUD we used the ImGui library: <https://github.com/ocornut/imgui> located under lib/imgui and the two methods renderWinHUD and renderLoseHUD are located at the end of the main file.

For the two HUDs, the zoom factor and the camera movement with mouse control are deactivated (scroll\_callback), making it possible to use the mouse to press the buttons in the HUD.

- Camera Object Tracking (2 Points): The camera tracks the player.

#### **Effects:**

##### Animation:

- Vertex Shader Animation (8 Points):

The quicksand has an ominous wave-like animation and it is slowly sinking, making it more easily to spot the longer you play, using vertexAnimationShader.vert.

(<https://www.youtube.com/watch?v=KEMZR3unWTE>)

##### Advanced Modelling:

- GPU Particle System using Transform Feedback (12 Points):

Implemented in particleSystem class with 4 shaders: particle.vert, dummy.frag, particle\_render.vert and particle\_render.frag. There is a possibility to define a lifetime for a particleSystem independent from the others. The particles are floating and rotating (defined in particle.vert). The particles are generated in the Main class and updated and rendered in the renderloop.

#### Terrain:

- Tessellation from Height Map (12 Points): We did not use Tessellation shaders, but our surface Geometries are dynamically generated using a previously generated Heightmap (The Heightmap is generated using the FastNoiseLite library <https://github.com/Auburn/FastNoiseLite>) which is generated inside WorldMap.cpp.

#### Texturing:

- Procedural Texture (8 Points):

Implemented with new shader marble.frag and already existing texture.vert. It has a marble like optic, the pattern is defined in the marble shader. This texture is not visible at first but only after the artifacts are collected, because only then the pyramid is visible (due to our gamelogic). The shader in Main is called marbleShader and with the marbleMaterial the pyramid can be created as a Geometry.

#### Libraries:

- <https://github.com/Auburn/FastNoiseLite>
- <https://github.com/ocornut/imgui>
- <https://github.com/assimp/assimp>

#### Demo-Notes:

We spawned the artifacts close to the player for easier debugging of the game functionality. In the real game they will be more separately placed on the map. Also the quicksand has a different texture but if you want to see the way it is supposed to look, change the texture from "woodTexture" to "sandTexture" on Line 381 in Main.cpp

We are having trouble with the Particle System in our Executable, but on both our Laptop it works ok. You can see in the video how It looks and we will try to fix it ASAP.