

Submission 2

Group Name: **Veil of Fear**

Students: Maria Elisa Barillas Molina, 01527156,
Dominik Fuchs, 00649613

GitHub Repository: <https://github.com/codefuchs1/cgue23-veiloffear>

Brief description of implementation:

Veil of Fear is a first-person survival game. With the use of 3D Geometry, we set the scene in foggy woods, where the player suddenly sees shadowy figures slowly approaching. The player can move around, run, jump and crouch. The effects implemented in our game are a CPU particle system, specular maps, PBR and Bloom.

Additional libraries:

tinyobjloader for model loading, stb_image for texture loading, physX 4.1 for physics.

Gameplay:

Mandatory:

- 3D Geometry: Our game uses 3D models we found on the internet (ghosts [1], trees [2], and the backpack [3]), or models we created ourselves in Blender (groundplane, stones). All models have been imported with tinyobjloader [4]. In the current state of the game, they are loaded individually, which affects the starting time of the game immensely. But this doesn't affect the in-game performance (always over 60 fps in our tests). We plan to optimize via instancing until the game event.
- Playable: The player in our game can move in all directions with the WASD-keys and change the view and moving direction using mouse movements. For running keep hold the SHIFT-key, and for jumping press the SPACEBAR. The player can also go into a crouch position with the C-key.
Jumping is only possible while not crouching.
- Advanced Gameplay: The game is set in a light forest veiled in fog. During the game, more and more ghosts appear on random positions and follow the player. They also are always facing the player (except when frozen). After some time, ghosts are also able to teleport to the back of the player to block his escape route. Trees and stones can also be obstacles to the player.
There is also a backpack hidden somewhere in the foggy woods. The player can collect the backpack to freeze the ghosts for 20 seconds for better chances to survive.
- Min 60 FPS and Framerate Independence: The settings.ini file in the assets folder holds the most important settings to start the game. These can be configured. The parameter "refresh_rate" represents the desired frame rate.
This value is inputted to the glfwWindow in the main.cpp file. A method "showFPSinWindowTitle" has also been implemented to test the framerate during the game. The delta time which measures the time time that is used for each frame is

updated for every frame and forwarded to all relevant code parts such as physic steps, camera movements as well as updating and rendering (dynamic) game objects. To realize one step per frame an accumulator has also been added to the *stepPhysics* method according to the references at the end of this report.

- Win/Lose Condition: To win, the player has to survive in the woods for 2 minutes. Every time the player is touched by a ghost, he loses one of 3 lives. If all lives are lost, the player loses. They also lose if they fall off the ground plane.
- Intuitive controls: The player in our game can move in all directions with the WASD-keys, run with shift and jump with the spacebar, combined with looking around via mouse movements. Classical first person shooter controls. With the C-key the player is able to crouch and therefore moves slower. Additionally, to the movement, the ESC-key can be pressed to quit the game. And pressing the F3-key (de-)activates the fog.
- Intuitive Camera: The camera has been implemented according to the opengl tutorial and code snippets linked in the references. In our implementation the class *FPSCamera* provides the manipulation and transformations for the camera/view. The callbacks for the input controls via keyboard and mouse have been implemented directly in the *Main.cpp* file. In the common “play” mode the camera is stick to a cinematic capsule controller, which is basically the physical rigid body of the player. Depending on the parameter “freeCamera” in the settings.ini the camera moves differently and the viewport changes. When the value is set to true it can float freely around the sky without gravity or restrictions. The whole area is visible without fog to get a better impression of the textures and models. When the value is false the camera stays in the ground plane. This is the above mentioned “play”-mode which is the common setting for playing the final game.
- Illumination model: We implemented a directional light for main illumination and a point light (as the moon). The backpack and ghost have the same material coefficients, set in *Mesh.cpp*, but they have different types of textures implemented. They use are the Phong illumination model in the *texture(_XXX).frag* shaders. The stones, trees and groundplane materials are based on physically based rendering, adjusted in Blender. All normals were calculated in Blender, tangents are calculated in the fragment shader for the PBR models.
- Textures: All the objects in our game have assigned textures, loaded using *stb_image*. As they are loaded in *Model.cpp*, mipmapping and trilinear filtering is enabled. Texture coordinates were all calculated in Blender. Based on the types of textures available, the model is rendered with PBR or using the Phong illumination model either with the texture shaders or the pbr shaders.
- Moving Objects: Besides the player itself there are several dynamic objects in our game that are moved differently. The ghosts are cinematic dynamic actors that are following the player using the *setKinematicTarget* method. The ghosts also always face the player. The moving vectors and rotation angles are calculated using simple algebra.
The two types of stones can be pushed/rolled around by the player (and ghosts). Each of the types has set an individual mass resulting in different behaviour when

using applying force to them. The backpack has a continuous rotation around the y axis.

- Adjustable Parameters: Based on the (ECG) provided configuration and the settings.ini we added a class *GameConfig.h* file that holds all relevant settings for the application. It reads the settings.ini and if any parameter is faulty or not provided it falls back to a default value. Among other things the window size of 1280x768 (if not in fullscreen mode) and the frame rate of 60hz are set/loaded here. These settings are fed to GLFW and other usages in the *main.cpp* file. There are also other parameters distributed in the code that affects the gaming experience, such as the number of trees and stones or the velocity of the ghosts. But these are usually not being options to be changed by the player. These parameters would be a great starting point for different difficulty levels in a future version.

Optional:

- Collision Detection (Basic Physics): The Nvidia PhysX library version 4.1 has been build and integrated to our program according to the tutorial provided in TUWEL. It is used to create interactions between all objects. Basically, the player cannot walk through static objects. The player cannot walk through trees or stones, but can jump on/over stones. All special collision events are implemented in the *Character.cpp* file.
- Advanced Physics: The player and the stones have some and gravity and fall of the ground when reaching its end. The player falls again after jumping to the ground. There is also a non-linear jumping curve implemented, to achieve a more realistic physics experience. Jumping is also not possible while crouching or something above the player is blocking him. When running/moving against stones, a force is applied to them. They start rolling and behave depending on their mass. Big ones roll slow, slow ones roll fast.
The ghost is also implemented as a kinematic rigid body that follows the player. When the player and a ghost collide, the player loses a life, and the ghost is repositioned for a new hunt. (Also, particles are firing.) A collision with the backpack results in freezing ghosts and repositioning the backpack.
- View-Frustum Culling: A view-frustum is generated with the parameters from our fps game camera. To know which objects should be rendered, each object has a bounding sphere that is used to calculate if the object is inside or outside of the view-frustum. Since we implemented instancing to render a large number of trees on our game scene, only one of the trees has a bounding sphere that can be detected by the frustum. So, if the bounding sphere is not inside of the frustum, no trees are rendered at all and vice versa. Additionally the number of displayed- and total objects in the current scene are outputted to the console. The frustum culling can be activated resp. deactivated by pressing the F8-key.
- Heads_Up Display: A timer is visible in the bottom left corner of the window. This timer gives the time that is left for surviving and winning the game. In the bottom right corner are a number of black hearts visible. These give number of current lifes. When collecting the grail or getting hit by a ghost, the number of hearts increases respectively decreases. Additionally info messages are displayed over the screen when winning or losing the game. The title of the game is also displayed for some seconds at game start.

Effects:

Advanced Modelling:

- CPU Particle System: Starting off with the tutorial for “GPU Particle System using Transform Feedback” we made one step back and decided to implement the simpler CPU particle system and upgrade if there is some time left. The tutorial and code snippets have been used as base for our implementation (some parts have been adopted). The particles use their own shader files and a simple dot image as texture which is replicated as billboards. Every time a ghost hits the player it “disappears” and at the position where the ghost was appears a particle fountain for a few moments. The particles of this fountain are reduced over time until none are created anymore.

Texturing:

- Specular Map: The backpack includes a specular texture, that is loaded using `stb_image` and implemented in the fragment shader. The implementation can be seen in *texture.frag*, or the other *texture_XXX.frag* shaders. It is implemented using a simple phong illumination model.

Shading:

- Physically Based Shading: This is implemented in our game using the Cook-Torrance model, using the tutorials linked below. Each model using this shading model has an albedo texture, a normal texture, a roughness texture and an ambient occlusion texture.

Post Processing:

- Bloom/Glow: Bloom is implemented in our game using the gaussian blur and HDR. HDR is used not only as a part of the PBR, but also implemented in the *texture_bloom.frag* and *texture_all.frag*, used so that the ghosts will glow a little bit more than the rest of the scene. The whole scene is rendered into a framebuffer using two textures in *Main* and *GameCore.cpp*, the color/hdr texture, and the brightness texture, where all colors under a certain value are filtered, so that only the brightest are kept. This is then blurred multiple times with “Ping-Pong framebuffers” using the Gaussian blur (*gaussBlur.frag*) and added to the hdr/color texture to create a bloom effect (*framebuffer.frag*).

Other special features: With use of the depth buffer, we apply a foggy effect on the rendered scene, to set up a spooky atmosphere. Through the fog, there are shadowy figures that can be seen, slowly approaching the player (similar to our poster). The fog is implemented in most of the shader files using a background mix and depth calculation based on the tutorial provided in the references. With pressing the F3-key, the fog can be disabled or enabled in our shaders. Sometimes it's easier to find the backpack in this way.

Also the starting position of all objects in this game (except character and ground plane) is never the same. All positions are randomly computed and distributed over the map. Additionally, the trees also are randomly scaled and rotated in order for the forest to appear more realistic. Therefore the player never plays on the same map twice.

To get more atmosphere into the game, background music as well as some sound effects have been added (using the audio library irrKlang). Sounds are played when collecting items, hitting a ghost, rolling a stone or die/win.

Loading time and performance were improved by implementing instancing to render the trees. It was important to us that we could have a large amount of trees to create an immensely atmospheric setup. So we load one tree model, and generate a number of model matrixes, which are passed to the vertex shader in a VBO. With `glDrawElementsInstanced()`, we can then draw as many trees as we want, but with much faster loading time and better performance.

Walk-through: When the game starts the player finds themselves in the middle of foggy woods. As the timer starts a ghost appears and starts to hunt the player. If the player runs using the shift-key, it is easier to escape from the ghost. But watch out: every 10 seconds appears a new ghost for the hunt. If the player finds and collects the backpack all ghosts freeze for 20 seconds and the player can run far away. This increases the chance of survival. Find your way through the trees and over stones. Be aware that the player can only take three hits from the ghosts. After the third hit the player gets possessed by the ghosts and dies. Also not watching your step and falling into a ground can lead to an immediate death.

If the player is able to survive for 3 minutes, the game is won. After that the game restarts for a new haunted veil of fear.

References:

[1] Ghost model: <https://www.turbosquid.com/3d-models/3d-room-1456414>

[2] Tree model: <https://sketchfab.com/3d-models/lowpoly-dead-tree-06b6d911f5154f3e8dd1ba50855bfe19>

[3] Backpack: <https://sketchfab.com/3d-models/survival-guitar-backpack-799f8c4511f84fab8c3f12887f7e6b36>

[4] Holy Grail: <https://sketchfab.com/3d-models/holy-grail-9d1a968b9b5b4116b2b78a2298acacac>

[4] HUD Font: <https://www.dafont.com/holstein.font>

[5] Textures: All free -> No licence issues

[6] Sounds: StoneSound: <https://pixabay.com/sound-effects/search/stones/>
Everything else: Self-created by Dominik Fuchs (me -> so free for use)

Libraries:

- tinyobjloader: <https://github.com/tinyobjloader/tinyobjloader>
- stb_image: https://github.com/nothings/stb/blob/master/stb_image.h
- PhysX:
 - <https://github.com/NVIDIAGameWorks/PhysX>

- <https://tuwel.tuwien.ac.at/mod/book/view.php?id=1838317>

3D Geometry:

- <https://learnopengl.com/Model-Loading/Mesh>
- <https://learnopengl.com/Model-Loading/Model>
- <https://marcelbraghetto.github.io/a-simple-triangle/2019/05/05/part-11/>
- <https://blog.42yeah.is/rendering/2023/04/08/tinyobjloader.html>
- https://vulkan-tutorial.com>Loading_models

Illumination Model:

- <https://learnopengl.com/Lighting/Materials>

Textures:

- [OpenGL Tutorial 6 - Textures, Textures in OpenGL](#)

Shading:

- <https://learnopengl.com/PBR/Lighting>
- https://github.com/JoeyDeVries/LearnOpenGL/blob/master/src/6.pbr/1.2.lighting_textured/1.2.pbr.fs
- [Computer Graphics Tutorial - PBR \(Physically Based Rendering\)](#)

Post Processing:

- <https://developer.nvidia.com/gpugems/gpugems/part-iv-image-processing/chapter-21-real-time-glow>
- [OpenGL Tutorial 18 - Framebuffer & Post-processing](#)
- [OpenGL Tutorial 24 - Gamma Correction](#)
- [OpenGL Tutorial 29 - HDR \(High-Dynamic-Range\)](#)
- [OpenGL Tutorial 30 - Bloom](#)
- https://github.com/JoeyDeVries/LearnOpenGL/blob/master/src/4.advanced_opengl/5.1.framebuffers/framebuffers.cpp

FPS settings and physics accumulation:

- <http://www.opengl-tutorial.org/miscellaneous/an-fps-counter/>
- <https://gamedev.stackexchange.com/questions/116030/implementing-time-step-in-main-game-loop>
- <http://gameprogrammingpatterns.com/game-loop.html>
- https://gafferongames.com/post/fix_your_timestep/

Ghost rotation:

- <https://gamedev.stackexchange.com/questions/188984/rotating-an-object-to-face-the-same-direction-as-another-object>

Particles:

- <https://ogldev.org/www/tutorial28/tutorial28.html>
- <http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/>
- https://github.com/opengl-tutorials/ogl/tree/master/tutorial18_billboards_and_particles

Depth buffer/Fog:

- <https://www.youtube.com/watch?v=3xGKu4T4SCU>
- <https://github.com/VictorGordan/opengl-tutorials/tree/main/YoutubeOpenGL%2014%20-%20Depth%20Buffer>

FPS camera and movement:

- <https://learnopengl.com/Getting-started/Camera>
- https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/camera.h
- https://learnopengl.com/code_viewer_gh.php?code=src/1.getting_started/7.4.camera_class/camera_class.cpp
- https://learnopengl.com/code_viewer_gh.php?code=src/1.getting_started/7.5.camera_exercise1/camera_exercise1.cpp

Character controller and collision detection:

- <https://nvidia-omniverse.github.io/PhysX/physx/5.1.0/docs/CharacterControllers.html#creating-a-character-controller>
- <https://gameworksdocs.nvidia.com/PhysX/4.0/documentation/PhysXAPI/files/classPxControllerDesc.html>

HUD:

- https://github.com/opengl-tutorials/ogl/blob/master/tutorial11_2d_fonts/tutorial11.cpp
- <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-11-2d-text/>
- <http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/billboards/>

View-Frustum Culling:

- <https://learnopengl.com/Guest-Articles/2021/Scene/Frustum-Culling>

Instancing

- https://www.youtube.com/watch?v=TOPvFvL_GRY

Audio:

- <https://learnopengl.com/In-Practice/2D-Game/Audio>
- <https://www.ambiera.com/irrklang/downloads.html>
- <https://ambiera.com/forum.php?t=1013>