

The Arbiter - Documentation

Group/Game Name: The Arbiter

Brief description of implementation: The Arbiter is a third person wave shooter game, akin to Risk of Rain 2 and Helldivers 2, where your goal is to cleanse useful planets of their useless population and then extract yourself from said planets after fulfilling your duty. The game runs on a custom engine built from scratch using C++, Vulkan 1.3 and the following libraries.

Additional Libraries:

- GLFW for the application's window: <https://www.glfw.org/>
- GLM for various linear algebra functions: <https://github.com/g-truc/glm>
- stb_image and stb_truetype to load texture/image and font files: <https://github.com/nothings/stb>
- TinyGLTF to load (.glb) mesh files: <https://github.com/syoyo/tinygltf>
- Bullet for physics simulation: <https://github.com/bulletphysics/bullet3>
- miniaudio for sound: <https://miniaudio.io/>

GAMEPLAY - Mandatory:

3D Geometry (6 Points)

We utilize TinyGLTF to load meshes. Through the use of our asset manager we load mesh data (stored in .glb files) as "libraries". Examples for "non trivial" 3D objects are Clanker (the player) or the Torii gates. All models were created in Blender by us.

Playable (3 Points)

You can start the game loop by pressing the "start game" UI button or pressing "space" in the start screen. Then you can freely start moving around, there are text based hints on the top left on the screen to guide you. Whenever you die or successfully escape, you will once again be presented the option to start the game. You can start the program by opening the .exe or running the config.bat in the appropriate folder.

Min. 60 FPS and Framerate Independence (3 Points)

We compute delta time which allows framerate independence. You can see the current framerate and frame latency in the application's window title bar (Note: using the physics debug line mode might cause the framerate to dip under 60 FPS, there also might be stuttering during prolonged play because of the higher number of entities.)

Win/Lose Condition (3 Points)

The player has a health bar, if it goes to zero or below, you die and lose. The player can win by "freeing" a certain amount of enemies and making it to the extraction point alive. Additionally, we also have a score counter in case you want to compare / show off your performance.

Intuitive Controls (2 Points)

There is an input class which holds a keymap and polls keypresses from the window through the use of GLFW functions. We also added gamepad support, however, only DualSense controllers were properly tested and we cannot guarantee other types work as they should.

The keyboard mappings are:

- WASD - Movement
- Space - Jump (in-game), restart game (start screen)
- Shift - Slide (if grounded), Dash (if in air)
- Escape - Open the pause screen
- Left Mouse - Shoot
- Right Mouse - Scope
- C - Locking aiming direction
- H - Steer camera to rocket
- F11 - Toggle fullscreen
- R - Switch to free cam
- T - Toggle wireframe mode
- P - Toggle physics collider visibility
- U - Toggle UI visibility
- K - Kill yourself (as in, like, instantly loose all health to end the game)
- M - Toggle superior audio

Intuitive Camera (2 Points)

The default camera is similar to the arcball camera from GCG, but without zooming and the extra panning. By pressing R you can switch to a free cam mode. The free cam is mainly there for debug purposes, which is also the reason why the player is invincible and frozen in place in this mode.

Illumination Model (2 Points)

We use a physically based illumination model with basic shadow mapping alongside a metal workflow, meaning every mesh's surface has its own material applied based on a metallic and roughness factor. Because we export our models directly from Blender as a .glb, data like materials, textures, UVs and normals are baked in. In theory we have implemented spot, point and directional lights, however, we ended up only using a "global" directional light for our scene. For demonstration purposes we placed a blue spotlight to indicate the spawn point and next to it a purple-ish point light, which may be hard to see depending on the current terrain color.

Textures (2 Points)

We implemented a texture asset system using stb_image. For each texture loaded using this system mipmap is generated through blitting, every texture uses trilinear filtering, anisotropic filtering is enabled.

Textures can be seen on the turret enemy, the skybox, and the Kreister.

Moving Objects (2 Points)

The player, the enemies and bullets all are able to move.

Documentation (1 Point)

You are currently reading it.

Adjustable Parameters (1 Point)

We provide the option to set window "height" and "width", "fullscreen", "music_volume" and "sfx_volume" through command line arguments or a config. For the former we also provided a bat script, but in case you are wondering, the format for our cmd args is "--parameter=int".

You can also adjust screen resolution using regular window controls (dragging, minimize, maximize) because of swap chain recreation (, but for some reason our swap chain

recreation discriminates against the PTVC team and crashes the game for them. It works on all of our devices).

GAMEPLAY - Optional:

Advanced Gameplay (5 Points)

The player has an objective to kill a certain number of enemies to unlock their escape rocket. To achieve this task, the player is able to shoot bullets. However, the enemies can defend themselves, so you have to be careful to not die via sliding, dashing, jumping or walking out of harm's way. Enemies drop loot that improves the player's stats; specifically health, movement speed and attack damage. Every 30 seconds a new wave of enemies spawn, that get increasingly more tough and plentiful. Once the objective is fulfilled, the player can choose to escape, delay their extraction if they haven't satiated their lust for killing or die if they have a skill issue. There is also a boss fight if you manage to free enough enemies.

Collision Detection (Basic Physics) (6 Points)

We implemented collision detection between the player and the environment (ground, trees, etc.), so the player is not able to move / fall through it, using the Bullet Library. (Also the case for other objects like enemies, but this is in reference to the task description.)

Advanced Physics (4 Points)

For the interaction with enemies we have utilized more advanced physics: The bullets you are able to shoot to hurt enemies use ray casting to detect hits. Items and drones use a collision call back to check if they have made contact with the player, i.e. they were picked up or managed to crash into them. We also added gravity.

Heads-Up Display (4 Points)

We created our own UI system that is able to render important player info like the current score and health. Additionally we created a live rendered start screen, pause screen (press "Escape" during the game loop) and end screen. We improved upon these concepts by adding interactable buttons ("(re)start game" buttons) and text rendering via stb_truetype. The HUD is toggleable with "U" like requested.

EFFECTS:

Lighting: Shadow Map with PCF (16 Points)

For the sun (our main directional light source) we have implemented shadow mapping with PCF. There was no issue with peter panning, but we still use preventive measures against it (using front face culling), shadow acne is dealt with via Vulkan's inbuilt depth bias for pipelines.

Terrain: Tessellation from Height Map (12 Points):

During runtime we generate a height map based on 2D Perlin noise, which gets passed to the GPU. In the Terrain class we create a plane consisting of "chunks" (basically a quad of vertices), which get instanced and offset in the Vertex Shader, and get tessellated based on the height map in the Tessellation Control Shader and Tessellation Evaluation Shader. Additionally, we implemented basic LOD using the camera position. You can check the LOD by going into free cam mode (press "R"), toggling terrain wireframe mode (press "T") and flying around ("WASD", "Space", "Shift").

Animation: GPU Vertex Skinning (20 Points):

Through the aforementioned asset system we implemented a "Skin" and "Animation" asset, a skinning system then applies the bone structure and interpolates the object based on the applied animation.

The player's character, Clanker, uses vertex skinning for their walking animation.

Texturing: Procedural Texture (8 Points):

Because of the terrain system, we already generate a perlin noise texture at runtime. We reuse this texture for the turret enemy to demonstrate this procedurally generated texture more obviously.

Shading: Physically Based Shading (8 Points):

Already mentioned in the illumination model section, but we use the Cook-Torrance model for every object in the scene. The objects all have their own material properties, Clanker (the player's character) in particular has both rough and metallic surfaces. Other examples for metallic objects are the oil can item and drones, for non-metallic objects the environment's foliage. Neither student has done the PBR bonus task during the GCG course.

Other Features:

- **Fog:** Basic fragment shader exponential fog, randomized via multiple color and intensity pre sets, inspired by OGLDEV (<https://youtu.be/oQksg57qsRA>)
- **Audio:** Nearly every relevant action has sound, there is also background music. Press "M" during the game loop to switch to superior sounds.
- **ECS:** To manage game objects behind the scene.
- **Asset & Resource Managers:** To make working with external assets and Vulkan resources easier.
- **Environment / Cube Map:** Utilized for the skybox, but does not count because we both have done the GCG environment map bonus task.

Walk-Through:

Open the game using the provided bin/ folder and .exe. You will find yourself in the start screen. Press "space" or click the start button to start the game loop. You will spawn in a randomized, procedurally generated world with enemies coming at you. At the top left there is text that tells you what to do. When the game play loop ends, the end screen appears, tells you your score and gives you the option to restart. For more information about the gameplay and keybinds, read the documentation above.

Points Summary:

Mandatory: 27
Optional: 19
Effects: 64
Total Points: 110