

# Escape the Island

Computergraphik, 186.831, UE, 2017S

*Maria Schimkowitsch (0525297)*

*Benedikt Tomaselli (0926048)*

## Controls

WASD	Moving around
Space	Jumping
Mouse	Look around
E or Left Mouse Click	Use / Interaction
R	Toggle: Torch on/off
G	Toggle: Render AABBs as wireframe
F2	Frame Time on/off
F3	Wire Frame on/off
F4	Texture-Sampling-Quality: Nearest Neighbor/Bilinear
F5	Mip Mapping-Quality: Off/Nearest Neighbor/Linear
F6	Normal Mapping on/off
F7	Virtual Displacement Mapping on/off
F8	Viewfrustum-Culling on/off
F9	Blending on/off

## Implementation

Goal of the game is for the player to restore power and use a radio to escape the island, before a mysterious monster catches up. To restore the power, the user will have to switch on power stations scattered on the island. After all power stations are up and running, the player needs to find the radio to call for help. The player has 5 Minutes to accomplish this goal.

The game consists of a **terrain**, that is generated from a Terrain-map and Height-map, including rocks, grassland, beach and sea. During loading, the required vertices are generated, as well as the UVs for the individual terrain parts, and the Normals, which are first calculated per triangle-face and then averaged per vertex. The required Tangents and Bitangents are then calculated after the normals were averaged, using the Gram-Schmidt

orthogonalization. The terrain is then added to the bullet-physics world, hence functions as a base plane for the player to walk around.

The Terrain is populated with two types of **trees**: pines and leaf trees, as well as bushes. The positions are contained in the terrain-map. For rendering instancing is used, which means that each tree is loaded only once from the .obj file and rendered up to 100 times with different translations. The trees are represented by simplified boxes for the physics simulation.

Additional structure's positions, like the fence, walls, and the power stations are as well provided in the terrain map.

There is also a small ruin, showing the effect of virtual displacement mapping (Parallax Mapping) on it's walls. (can be turned on/off with F7)

The radio, which needs to be found and used to win the game, has different positions on the map, chosen randomly on each start.

The terrain also includes a **sea** piece. The water is represented by a plane, which is distorted using two perpendicular sine-waves on the gpu. The terrain is physically separated from the rest by an additional, invisible box.

### Free-moving camera

The player is represented by a capsule and a camera. The capsule is required for the physics simulation, and hence is responsible for the final positioning of the camera. To move the camera, the user first moves the capsule in the desired direction, and the actual position is then calculated by the physics simulation, taking all potential obstacles into account. The action "Jumping" applies a short-timed force, directing up, onto the capsule.

The player can also sprint for a short time, using the SHIFT key. After the sprint, the player needs to recover before sprinting again. The ability to sprint comes in handy, as the monster becomes faster than the player at a certain point.

### Moving objects

There is a mysterious creature peacefully wandering the woods, but as soon as the player toggles the first power station, the monster gets angry and starts hunting the player down. This can easily be observed by the monsters eyes, which turn red when it gets angry and also by the fact that it's moving in the player's direction quickly.

The hierarchical aspect of moving objects is the head of the monster. It can rotate on the monster's body independently of the bodies moving direction. This can be observed when the player gets into a certain range.

There are also two testing balls jumping around in the world, which were included as testing objects for the bullet-physics world.

The monster will be moved around by a simple AI, i.e. state machine. States are:

- Idle state: move to predefined positions in the world
- Active state: Hunt player, activated when player toggles first power station

## Textures

Most Models were generated with Blender and Maya. Each model also provides UV coordinates which are used for applying textures. Each model has

- **Diffuse** textures for the base colouring
- **Normal** textures for Normal mapping
- **Specular** texture for specular highlighting

## Light Sources

- **Directional Light** - Moon light, static and basic light source
- **Point Light** - a small floating sphere, that also shows fire particles, which can be switched on and off

## Viewfrustum-Culling

Can be switched on/off with F8-Key. The amount of triangles rendered is displayed by billboards showing a 2D-Text (in green).

To determine which objects have to be rendered and which not, the Bounding boxes of the individual objects are either tested once (when the object is rendered only once) or several times at different positions (when the object is rendered several times via instancing) against the view frustum of the camera. For that, the bounding box's coordinates are transformed into Projection space, and then checked against the planes of the frustum. If all coordinates are outside relatively to only one plane, the whole object has to be outside, and hence is not rendered this frame.

## Basic Gameplay

The player has to switch on three power stations and find the radio before the time is up (indicated by a count-down in the upper right corner). The **Count-down** is shown by billboards, with vertices calculated directly in screen space coordinates, and uvs which are generated per frame, depending on the to be depicted characters (each letter corresponds to a certain position in the provided font bitmap).

After switching on the first power station, the monster starts hunting the player down. With each toggled switch, the monster gets faster than before and eventually faster than the player. The buttons can not be turned off again.

After the last switch, the power is restored and the radio can be used.

## Interaction with Environment

For collision detection is the bullet library used, for instance when the player and the enemy collide which determines the end of the game.

Interacting with the buttons was separately implemented. For a successful hit detection, i.e. the player activates a button, three checks have to be fulfilled. The first check verifies that

the player is within a certain distance to the button. The second check verifies that the angle between the camera's, i.e. player's, looking-direction-ray and the ray between the camera's and the button's centre, is not too big. The third and final check performs Ray-Triangle intersection tests, between the button's triangles and the camera's looking-direction-ray.

## Remarks

It is possible that the enemy can get stuck in a wall. This is a known bug and related to our implementation of the collision detection with the bullet engine.

## Neat Extensions

If we would keep developing this game, we would also randomise the power stations positions. Additionally we would adapt the countdown-time, the amount of power stations and the speed of the enemy to different levels of difficulty, which could be chosen by the player himself. There would also be more than one terrain map to choose from.

## Effects

### Visual Effects

- **Shadow mapping** - includes Terrain, Trees, Bushes, Walls, Roof, Ruin and the enemy, and is visible on them
- **Normal mapping** - visible on Terrain and Trees
- **Plane animation via Geometry shader** - visible on the water surface (waves)
- **CPU particles** - simulating fire, moves with the floating sphere
- **Light Mapping** - visible on the wall object near the beach
- **Omni-directional shadow mapping** - visible on terrain that some objects (roof, walls) cast shadows
- **Virtual Displacement mapping (Parallax Mapping)** - visible on the ruin. Original model is flat. Virtual displacement is done in the Fragment shader and results from transforming a ray (from the camera to the individual vertex) into Texture space, and then sampling that ray against a height texture. The values returned from the sampling are used to distort the UV coordinates, with which the final texture is sampled.
- **Displacement mapping** - visible on the monster's body, which originally is a simple sphere. The spikes result from displacing the vertices in the Tessellation Evaluation shader.
- **Spot light** - visible on the three power stations (red light cone). Turns green when power switch is toggled.

### Sound Effects

- Ambient soundtrack: background music
- Action: jumping, toggle power stations
- Events: monster gets angry, monster gets faster, power restored, losing, winning

## Used Libraries

- **glew** ... OpenGL extension wrangler library
- **glfw** ... OpenGL Framework library
- **glm** ... OpenGL mathematics library
- **assimp** ... as modelloader, currently: only triangulated models in .obj files possible
- **SOIL** ... image loader library, used as aid for texture generation
- **SFML** ... simple image loading library, used for sampling the terrain-map and height-map for the Terrain generation, and structure positioning
- **irrklang** ... sound library
- **bullet** ... physics library