

# DEFENSE OF THE SKYLANDS



## A 3D FANTASY STYLE TOWER DEFENSE

*Defense of the Skylands* is gonna be a tactical Tower Defense. As the captain of a flying ship called *The Empress* you and your crew are the first ones to reach those legendary floating islands. What you don't know is that there's already various different creatures living there. As you stop to measure the lands and to restock resources you're about to find out that those creatures do not have the intention of sharing those Worlds.

### The structure

The general approach behind our engine's structure is the following: A window shows a certain scene. The scene itself holds one tree of 3D-nodes and a separate tree of 2D nodes, as well as some other instances like a general scene animator. Both the 3D and 2D-nodes themselves possess at least a name and a translation matrix and always update their global translation in respect to their parents. Specialised nodes like Mesh-Nodes, Camera-Nodes or Light-Nodes extend this basic set of functionalities and allow furthermore the appropriate handling of those different objects.

Having a data structure like this is quite common in computer graphics and the standard in Modelling software. It made the implementation of certain features, like the Object Loader a lot easier as ASSIMP is also using a hierarchical structure like that.

## Feature-List

Category	Feature	Description/Usage
Gameplay	<b><u>Compulsory</u></b>	
	<i>3D Geometry (6 Points)</i>	Our 3D node system allows us to simply create 3D scenes, such as the game's levels. Every node uses not only its own transform, but also applies the parent's transform.
	<i>Playable (3 Points)</i>	Players can move the camera around the scene, interact with the environment and influence what is happening by building towers.
	<i>Advanced Gameplay (3 Points)</i>	You may win or loose. :D
	<i>Min. 60 FPS and Framerate Independence (3 Points)</i>	We pass time that has passed since the previous frame (delta time) to every update and render function. This allows us to scale actions and ensure a smooth experience.
	<i>Win/Lose Condition (3 Points)</i>	You win by successfully defending The Empress from hordes of enemies. Should you fail to do so you will lose.
	<i>Intuitive Controls (2 Points)</i>	Most of our game is mouse controlled, so players should be able to figure out things quickly. We try to use hover highlights on buttons and other tricks to make it obvious what objects can be interacted with.
	<i>Intuitive Camera (2 Points)</i>	Since DotS is a strategy game we decided to go with an overhead perspective. The camera can be controlled with the mouse and should feel familiar to what people are used to from other 3D software and games.
	<i>Illumination Model (2 Points)</i>	In addition to the sunlight with shadows (see ShadowMaps) our scene UBO allows us to provide an array of light sources, which will be applied to all 3D objects. For an example of a point light please look at the portal that is spawning enemies.
	<i>Textures (2 Points)</i>	Textures are loaded by our TextureManager and have by default mipmaps and trilinear filtering enabled. Our system allows us to overwrite those standards though as we see fit. (e.g. we don't want mipmaps neither linear filtering for the color swatch)
	<i>Moving Objects (2 Points)</i>	We have a number of moving objects in our game: animated enemies, moving particles, and a handful of objects that are simply following a spline.
	<b><u>Optional Gameplay</u></b>	
	<i>Heads up Display (4 Points)</i>	We have an entire 2D system which we use for UI elements in the game.

	Collision Detection (4 Points)	We use PhysX for everything physics related. Our primary use case is to raycast against geometry objects for mouse interactions. (e.g. placing towers)
	Advanced Physics (6 Points)	We have a very rich physx integration, where physX bodies can already be attached at modelling time. We wanna refer to our Readme.md on our github if further information are required.
Effects	<b><u>Lighting</u></b>	
	Shadow Map with PCF (16 Points)	We implemented stabilized cascading shadow mapping which should be a good fit for most scenarios. Right now we have 3 shadow maps for different camera distances. This allows us to cast shadows not only on static objects, but also on dynamic objects like enemies.
	<b><u>Advanced Modelling</u></b>	
	CPU Particle System (8 Points)	We have a simple ParticleSystem node that allows us to spawn and manage a large number of particles. Information about the particles is handed over to the GPU using instanced arrays. An example for this effect is the dust that is emitted whenever a new tower is built.
	<b><u>Animation</u></b>	
	Hierarchical Animation (4 Points)	Or as we call them: Node Animations. They can be set up in Blender and are imported in the FBX-File. Further information can again be found on the github Readme file. We use them for the enemy movement aswell as the ship intro.
	GPU Vertex Skinning (20 Points)	They are used on turrets for the construction animation but also on the enemies.
	<b><u>Texturing</u></b>	
	Video Texture (8 Points)	They can be seen on the TV-like thing aswell as on the Tower-Building cards. You can see the tower Building cards when pressing on a Tower-Site.
	<b><u>Shading</u></b>	
	PBR (16 Points)	All our materials use a PBR shader for rendering. However, due to time constraints we were unable to make full use of this. The cannonball that is spawned at the start of the game should show what our shader is capable of,
	<b><u>Post Processing</u></b>	
	Lens Flares (8 Points)	Lens Flares are rendered on top of the framebuffer when you look in the direction of the sun. To implement this we draw elements of the lens flare to the screen using quads depending on the camera position and orientation in relation to the sun. We also implemented Occlusion Queries so they are not visible when behind objects.

## The Game

All of the core mechanics are in place and working fine, but we had to cut some additional features since we couldn't get them working in time for the deadline.

Possible future additions are:

Projectiles: We already got physx working with our scene. Once we figure out why Physx starts crashing whenever we spawn too many projectiles we plan to actually shoot stuff at the enemies.

More PBR content: While our engine and shaders are finished and fully capable of handling PBR materials we severely underestimated how time consuming texture painting and baking PBR content would be.

More levels, enemies, towers: Our entire engine was designed to be as modular as possible. Now that everything is ready and battle-tested, adding additional content should be relatively easy.

## Controls

- LM & Drag** ... orbit around current focus point
- RM & Drag** ... strafe
- Scroll +/-** ... Zoom in and out
- LMB** ... interact with buttons and objects ingame

## Libraries

- We use assimp for loading 3d models, scenes and even cameras into our scene:  
<https://www.assimp.org/>
- And PhysX for anything physics related:  
<https://developer.nvidia.com/physx-sdk>
- FreeType to load and create renderable images from TrueType fonts.  
<https://www.freetype.org/>

In addition we are using CC0 textures from TextureHaven & opengameart.org and Blender for any 3D modelling.