

Survive Until Daylight

Made by Jakob Staudinger & Gabriel Sperrer

Gameplay

The game features up to four players (either one using keyboard and mouse or more using controllers), a nice landscape and goblins. The main goal is to survive until daylight and to fight the goblins until then.

How to play

The gameplay is as in many other first-person games: Use W-A-S-D to walk (or the left joystick on the controller), the mouse to look around (or the right joystick on the controller) and the left mouse button (or A on the controller) to attack. Using shift (or RB on the controller) one is able to run faster to escape the goblins. You can jump by pressing Space (or B on the Controller).

Implementation

A lot of our knowledge comes from two books, [Game Engine Architecture](#) and [Introduction to 3D Game Programming with DirectX 12](#). Those have been helpful for a lot of the following and won't be mentioned every time.

Structure

We've structured our engine into modules, each of which more or less independently working on a specific section of the game.

Options

In the options screen (accessible from the pause menu) you can change some settings of the effects. The adjustable options are:

- Ambient Intensity: Controls the overall brightness of the scene
- Occlusion Radius: Controls the sampling range of SSAO (how far away geometry can be before it does no longer influence ambient occlusion)
- Bloom Radius/Bloom Sigma: Controls the parameters of the blur applied to the bloom effect (gaussian blur)
- Bloom Threshold: Controls how bright a part of the screen has to be for it to bloom
- Cloud speed: Controls how fast the clouds move
- Cloud cutoff 1|2: Controls the "density" of the clouds

- Debug Shadows: Controls whether to display the slices of the Cascaded Shadow Maps in color

By pressing the "Save" button, the settings are saved to a file and loaded the next time the game starts.

Effects

All Effects can be found in Engine/Rendering/Effects (cpp code), in Engine/Internal Resources/Shaders as well as in Resources/Shaders (.hlsl code)

Multi-Light Scenes

We support Point- and Directional lights and have no problem rendering a lot of them as we are using deferred rendering for calculating the lighting. However, only one or maybe two of these light sources should be casting shadows, otherwise the performance will drop drastically.

Cascaded Shadow Maps

We support Cascaded Shadow Maps for directional light sources. We are closely following this paper by NVIDIA:

https://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf. To see the different cascades more clearly, a debug switch can be toggled in the Graphics options in the game.

SSAO

We implemented a Screen-space Ambient Occlusion Algorithm very similar to <http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html>. The occlusion radius can be adjusted in the Graphics Options (or turned off by setting it to 0).

Terrain

We are loading a DDS heightmap of 32-bit floats and then using it as a displacement map for our terrain. The colours are calculated based on how high the current vertex is and how steep its rise is.

This was inspired by <http://thedemonthrone.ca/projects/rendering-terrain/>.

Animations

We are loading skinned meshes and animations using the glTF file format. We are interpolating between the keyframes for each joint every tick on the CPU, and transforming the vertices based on their weight on the GPU using the resulting matrices.

Procedural Textures

All textures used by the Skysphere are procedurally generated. The cloud generation uses a similar technique as shown in <https://lodev.org/cgtutor/randomnoise.html>. For the stars we simply take a random noise texture and discard values outside of a very small range ([0.5, 0.505]).

Physically Based Shading

For PBR we followed this tutorial <https://learnopengl.com/PBR/Lighting>. However, we did not implement IBL from the following tutorial.

Bloom

We use a simple Bloom algorithm as a Postprocessing Effect. Only values above a certain threshold (adjustable in the Graphics Settings) are kept and blurred using two Gaussian blur passes (horizontal and vertical). The sigma and kernel size of the gaussian blur can also be adjusted in the options.

Day / Night Cycle

We've implemented a fully functional day/night cycle as it matches the topic of our game. The sky adapts to the rotation of the sun and also stars and the moon start appearing or disappearing depending on the sun position.

Libraries & Tools

For creating the Engine behind SUD, we used the Physics Engine [PhysX by Nvidia](#), the [Microsoft GLTF SDK](#) and [Pix for Windows](#) (only for debugging purposes).

The only external tools involved were Blender and Adobe Photoshop, although we wrote custom tools when necessary.