CS562 Assignment 2: Cascaded Shadow Maps

OBJECTIVE

In this assignment you are going to implement Cascaded Shadow Maps on top of the deferred shader from the previous assignment.

1. Scene setup

Load the provided scene file that will import a voxelized scenario and a new type of content, the directional light.

```
"directional_light":
{
    "direction":
    {
        "x": 0.3,
        "y": -0.6,
        "z": -0.7
    },
    "color":
    {
        "x": 1.0,
        "y": 1.0,
        "z": 1.0
    }
}
```

2. Algorithm Overview

Cascaded Shadow Maps are a technique to produce shadows on large areas, mainly open landscapes. The idea is based on subdividing the camera frustum into different subfrusta and producing a different shadow map for each of the splits. For the assignment this algorithm will be applied only for the unique directional light loaded from the scene file.

Number of splits will always be 3 with shadow maps of decreasing resolution. The nearest shadow map will have a resolution of 2048x2048 and it will halved for every split (1024x1024 and 512x512). In order to render the shadow maps first, the light's camera and projection matrices need to be computed. Those will depend on the 3 splits of the camera frustum and each cascade will have its own matrices. Once the matrices are computed the whole scene will be rendered to each of the shadow maps and send to the directional light pass. This pass is similar to a regular point light pass implemented in the 1st assignment, but related to the directional light that will produce the shadows. In this directional light pass, based on the distance of the surface to the camera the corresponding shadow map will be selected, sampled and evaluated.

3. Splitting

There are multiple approaches to divide the camera frustum into the 3 splits:

- **Uniform distribution:** each subfrustum will have the same distance from near to far.

$$z_i = n + \left(\frac{i}{N}\right)(f - n)$$

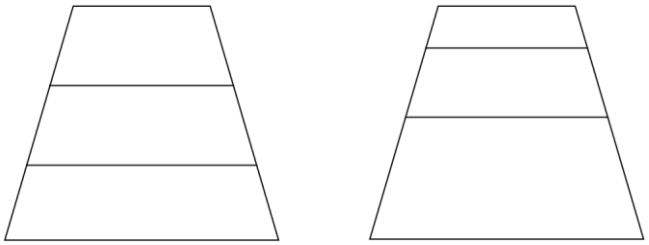
- **Exponential distribution:** splits become wider the farther you move from the camera

$$z_i = n\left(\frac{f}{n}\right)^{\frac{i}{N}}$$

- **Hybrid distribution:** interpolation of the previous approach adjusted with a λ factor.

$$z_i = \lambda\left(n\left(\frac{f}{n}\right)^{\frac{i}{N}}\right) + (1 - \lambda)\left(n + \left(\frac{i}{N}\right)(f - n)\right)$$

z_i: z value for split *i*
N: Number of cascade splits
n: Near distance of the camera
f: Far distance of the camera



Uniform Exponential

4. Percentage Closer Filtering

The general idea of PCF is to calculate a percentage of the pixel in shadow based on the number of subsamples that pass the depth test over the total number of subsamples. This will produce blurred edges on the shadows, making them look softer and smoother.

5. Blend between Cascades

When applying cascade shadow maps, the changes from a shadow map to the next split is usually a visible seam on the scene. The solution is to create a band between both shadow maps that samples both shadow maps and blends them together to produce a smooth transition.

6. Occluder distance

The optimal solution would be to generate the shadow map only with the overlap that most tightly encapsulates each of the subfrusta, but that has an undesired consequence. Occluders that fall outside the frustum will not cast shadows in the geometry that is visible. To avoid this case, the lights near distance will not exactly fit the frustum, but it will add an extra margin distance to generate shadows. This assumes that every object further than that maximum distance may not produce the desired shadow.

INPUT

With the goal of showcasing your work the best possible way the following specifications need to be meet:

- A first-person camera, controllable by the user via the keyboard and/or mouse.
  - WS: move the camera forward/backward along its forward vector.
  - AD: move the camera left/right along its side vector.
  - QE: move the camera up/down along the **global** up vector (0,1,0).
  - Click + Drag: the mouse should tilt the camera around its yaw and pitch.
  - Up/Down: rotate the camera around the pitch.
  - Left/right: rotate the camera around the yaw.
- Pressing the **Control + R** key should reload the scene from the file.
  Pressing **F5** should reload all the shaders.
- You should be able to display the content of any of the shadow maps.
- Enabling/Disabling a rendering mode showing cascade selection as color (red for the closes, green for the intermediate and blue for the furthest).
- The direction of the directional light should be editable, but it should be reset every time the scene is reloaded.
- Adjustment of the shadow bias that will be common for every shadow map.
- Adjustment of the cascade blending distance, the distance of the band that blends multiple shadow maps.
- Adjustment for PCF neighbor count.
- Tweaking of the maximum distance at which an occluder guarantees to produce shadow.

COMMAND LINE ARGUMETNS

The program should allow for the following arguments passed to the main function through the command line arguments.

- `argv[1]`: JSON scene name to load
  - Can be omitted, a default scene (the one provided) should be loaded in that case
- `argv[2]`: Filename of the screenshot to save at the end of the first frame in PNG format
  - Can be omitted and no screenshot is saved in that case
  - Can only be used if the scene is also specified

Example: `cs562_jon.sanchez.exe scene.json screenshot.png`

SUBMISSION

The project will be submitted as a Visual Studio 2019 that will compile and execute without any adjustment. The solution will **only** have the **64-bit platform**, so the 32-bit configuration should be removed. The framework will be created from scratch and will have the following folder hierarchy:

- data -> Folder with all the resources
  - gltf -> Meshes and textures to load **(NOT included on submission)**
  - scenes -> Scene json files **(NOT included on submission)**
  - shaders -> Implementation of the shaders **(INCLUDED in submission)**
- src -> Folder with your code
  - Any header only library to compile (ImGui, stb…)
- cs562_<login>.vcxproj
- cs562_<login>.vcxproj.filters
- cs562_<login>.vcxproj.user
- Required DLLs (only x64 versions)
- include (all include folders for the libraries used)
- lib -> All library files for the libraries used)
- obj -> Folder where the compilation intermediate files when solution is compiled **(NOT included on submission)**
  - Debug
  - Release
- bin -> Folder where the executable will be generated when solution is compiled **(NOT included on submission)**
  - Debug
  - Release
- Solution file (cs562_<login>.sln)
- README.txt -> Text file containing header (see below) as well as controls for the application.

README FILE

Along with your Visual Studio 2019 solution you will attach a README.txt file that will contain the following contents:

1. **How to use your program:** Every control available when executing the program and hints on how to get use it.
2. **Important parts of the code:** Specify source code files and approximate function or lines that were most relevant for the assignment.
3. **Known issues and problems:** Any not completed part of the assignment or known bugs that can be encountered when running the program.

GRADING SCHEME

Section	Item	Points
Shadow map	• Generation	20
	• Selection	10
	• Application	30
Enhancements	• PCF	10
	• Blending	10
	• Occluder max distance	10
Scene and controls	• Correct Scene and interface	10