CASCADED SHADOW MAPS

Ivan Iglesias

December 2019

1 Introduction

Cascaded shadows maps are the best way to face one of the most problematic errors with shadow mapping: perspective aliasing. Perspective aliasing is a common artifact in shadow maps, that occurs when the mapping of pixels in view space to texels in the shadow map is not a one-to-one ratio. This happens because pixels close to the frustum's near plane are closer together.

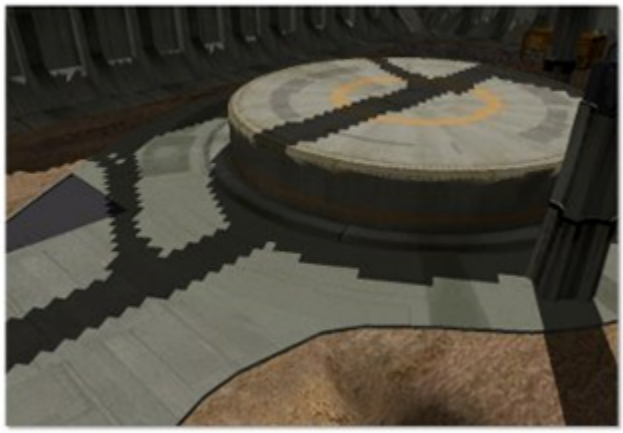

Figure 1: Perspective Aliasing

2 CSM Paper

The paper proposes using Cascade Shadow Maps to solve the aliasing problem. The method consists on splitting the frustum in different planes and create new frustums for each partition. The algorithm steps are:

1. For every light's frustum, render the scene depth from the lights point of view.
2. Render the scene from the camera and choose an appropriate shadow map based on the rendering fragment z-value.

3 My implementation

3.1 Splitting the camera Frustum

The algorithm starts by computing the camera corner points and splitting the frustum. The simplest method to approach this is simply dividing the total distance from the near plane to the far plane by the number of splits. Although this splitting method won't produce a uniform distribution of perspective aliasing. In my case, I have used a splitting method (proposed by Nvidia) that computes the splits using a logarithmic and uniform split schemes. This method is designed to produce an optimal distribution of perspective aliasing over the whole depth range.

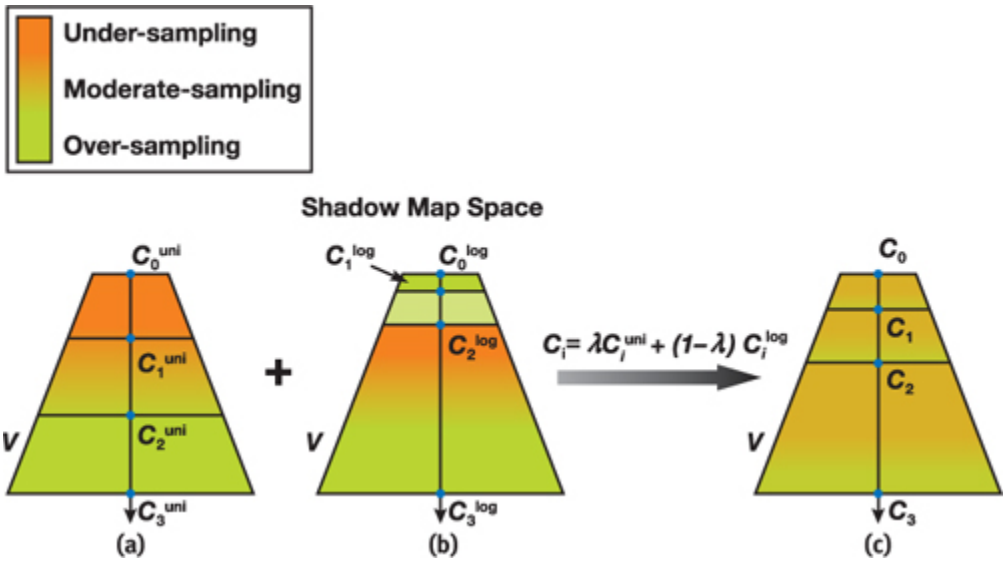

Figure 2: Splitting Scheme

3.2 For each Split

3.2.1 Compute Split Frustum

We need to compute a frustum per split. We will start with a base AABB corners defined between -1 and 1. Now, we need to move its coordinates to world space by multiplying the corners by the inverse of the View-Projection Matrix. Now that we have the frustum in world coordinates we will use the previously computed split distances to set the frustum defining points.

3.2.2 Create AABB for the orthogonal matrix

The orthogonal matrix will be computed from the AABB that contains the hole splitting frustum. As we will place the frustum in the direction of the camera we will need to ensure that the frustum will still be contained in the AABB. This means that we will computed the AABB from a sphere. We will add all the corners of the AABB and divide it by eight

to obtain the center of the frustum. Once we have this center we will create the sphere that contains the frustum by looking for the biggest distance from the center to the corners, this distance will define the radius of the sphere. Now we can use the radius of the sphere to compute our final AABB.

3.2.3 Create Light View and Orthogonal Matrices

The light View matrix will be computed with the light direction positioned at the center of the frustum.

The orthogonal matrix will be computed from the extents of the AABB from the previous step.

3.2.4 Round Matrix

In order to avoid light shimmering we need to create a rounding matrix. With this rounding matrix we will move in texel sized increments and match the shadow map with the orthographic matrix. For this, we need to compute the origin position in the shadow map, round the resulting vector and subtract it with the not rounded position. The result of this subtraction will be the offset we will need to apply to the orthogonal matrix to fix the shimmering.

```
glm::mat4 CSM::getRoundMatrix(glm::mat4 Ortho, glm::mat4 View)
{
    glm::mat4 shadowMat = Ortho * View;
    glm::vec4 shadowStartPos(0.0f, 0.0f, 0.0f, 1.0f);
    shadowStartPos = shadowMat * shadowStartPos;
    shadowStartPos = (shadowStartPos * ShadowMapSize) / 2.0f;

    glm::vec4 roundStart = glm::round(shadowStartPos);
    glm::vec4 offset = roundStart - shadowStartPos;
    offset = glm::vec4(glm::vec2(offset * 2.0f / ShadowMapSize)
            , 0.0f, 0.0f);

    Ortho[3] += offset;

    return Ortho;
}
```

Figure 3: Create Round Matrix

3.2.5 Render Scene Depth

The last step of the shadow map part is to render the scene depth. The vertex shader will simply take the LightViewProjectionMatrix (View * Orthogonal) and the Model matrix of the object.

3.3 Choose the correct ShadowMap

When rendering the final scene and applying the shadows we will need to choose between all the splits. The selection is just checking the distance of the fragment in view space and compare it with the splitting values that we calculated and passed to the shader.

```
uint findCascade(float zDepthValue)
{
    uint cascadeIdx = 0;

    for(uint i = 0; i < numOfCascades - 1; ++i)
    {
        if(zDepthValue < cascadedSplits[i])
        {
            cascadeIdx = i + 1;
        }
    }
    return cascadeIdx;

}
```

Figure 4: Choosing Shadow Map

4 Demo

Figure 5: Demo Screenshot

This image shows 4 Cascaded splits represented with different colors. The camera and the frustum are frozen at some point on the right, which makes the split cuts diagonal to the camera of the image. On the left part we can see the depth maps of the four splits. Now, let's take a closer look to the shadows.

(a) Split 3 (b) Split 2 (c) Split 1 (d) Split 0

Figure 6: Different Split Shadows

In these images we see different shadows of the same object at different split zones. The image at the left is the farthest one while the closest one is on the right. It can clearly be appreciated how the shadow gets worst the farther we are from the near plane.

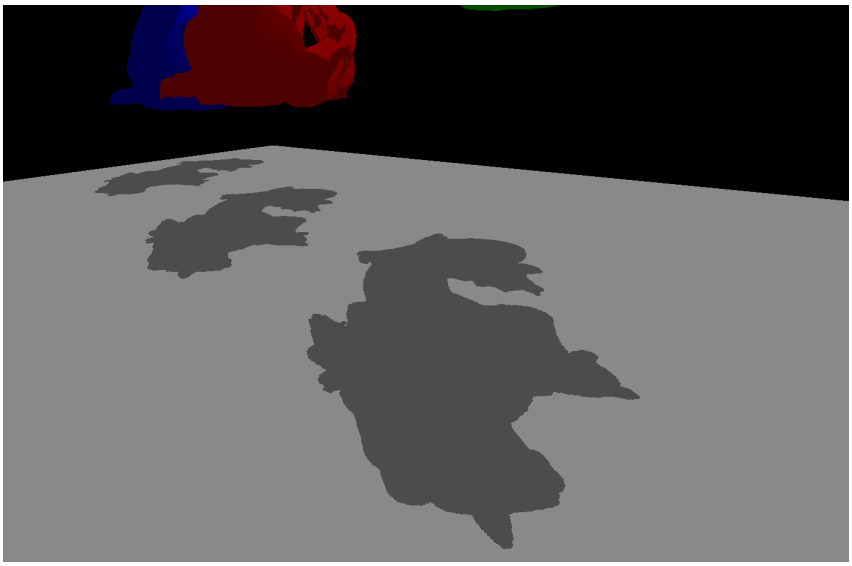

Figure 7: Demo Screenshot

Here we have the final result of all the splits in the same plane from the main camera point of view. The closer and the farthest objects seem to have the same quality on the shadows when they actually don't.

4.1 How to use the Demo

The demo is simple to use, just open the ".exe" file and freeze the camera to look around and see the differences in the shadows. There are some values on the GUI that can be tweaked.

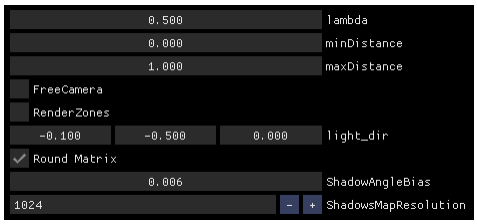

Figure 8: Demo GUI

Lambda, MinDistance and MaxDistance are values to offset the splits of the Frustum. FreeCamera, freezes the main camara and let's you move around with a temporal camera, the shadow maps are going to still be rendered from the main camera. Render zones, just renders the splits with different colors. LightDir controls the directional light. Shadow angle bias, modifies the bias angle when applying the shadow maps. Shadow map resolution, changes the resolution of all the shadow maps.

5 Other Approaches

5.1 Variance Shadow Maps

The VMSs work by rendering the depth and the depth squared to a two-channel shadow map. This two-channel shadow map can then be blurred and filtered. It poorly tries to solve the perspective aliasing. It can also be combined with CMSs for better results.

6 Problems during development

It was a bit hard for me to understand all the matrix multiplication. How the frustum end ups after each step.

7 Conclusion

I think that Cascaded Shadow Maps are the best solution to perspective aliasing. It is an intuitive method that makes the extra cost of processing it worth. I also think that it is one the best approaches for shadowing with global light illumination in big worlds.

It is also a technique that can be easily be combined with PSM and LSPSMS (Light Space Perspective Shadow Maps) to obtain better results. However, It is true that there are a bunch of things that can be done to optimized it: matrix multiplications and updating less frequently the fasthest shadow maps between others.

References

[1] Microsoft: Cascaded Shadow Maps,
Link

[2] Microsoft: Common Techniques to Improve Shadow Depth Maps,
Link

[3] Rouslan Dimitrov : Cascaded Shadow Maps,
Link

[4] Johan Medestrom: Cascaded Shadow Maps,
Link

[5] NVIDIA GPU Gems : Parallel-Split Shadow Maps on Programmable GPUs,
Link