

# Documentation for Submission 4

## Description of the implementation

The renderer final render consists of the following passes:

### 1) Prepass - Raster

**output** - Depth 32bits - Render Resolution

**output** - Compressed world space normals 16bits - Render Resolution

#### **Description:**

- All of the scene geometry is first rendered in a prepass. This is done for two reasons. First to reduce the pressure on the main pass which can be quite high since we use alpha discard geometry. Second because we need depth and normals for the screen space ambient occlusion. We utilize two separate pipelines for this, one with alpha discard and one without. The reason for this is to avoid not using early Z tests (as a result of using discard in fragment shader) on fully opaque geometry. Additionally it also reduces the bandwidth as for fully opaque objects we no longer need to sample the albedo.

### 2) Screen Space Ambient occlusion - Compute

**input** - Depth

**input** - Compressed world space normals

**output** - SSAO texture 16bits - Render Resolution

#### **Description:**

- Screen space ambient occlusion is calculated using 24 samples. Random 4x4 texture tiled on top of the screen is used to offset the tangent space and thus avoid banding. The samples are weighed so that most are distributed around closer to the original position. Additionally the kernel is generated with offset Z coordinates (in Tangent space) to avoid self shadowing. Groupshared memory was utilized to prefetch depth samples with a distance based heuristic. Each thread group fetches a 32 x 32 pixels wide tile of depth and uses it as a cache to avoid duplicate depth texture fetches for neighboring pixels. A depth based heuristic decides if this cache should be used on a per thread group basis. This is because the closer the depth the more spread the samples are and the less chance of hitting the cache they have.

### 3) Min Max depth passes (2) - Compute

**input** - Depth

**output** - Min/Max values stored in the depth buffer

#### **Description:**

- A two pass filter is used to compute the depth min and max values. In each pass a thread fetches 4 samples (in the first pass a 2x2 tile in the depth texture, in the second pass 4 values from GPU buffer). It then computes the min/max locally before using subgroup operations to compute min/max per warp. A single thread from each warp then writes the min/max bounds into group shared memory. Finally a single warp fetches the values from the shared memory and computes min/max using subgroup ops. A single thread

from this warp (and thus also from this thread group) writes the value into the global GPU Buffer. This buffer is then used by the second pass to finish the min/max pass.

4) Shadowmap matrix compute pass - Compute

**input** - Min/Max depth

**output** - Per shadow cascade data (ProjectionView matrix + cascade span)

**Description:**

- Each thread samples the min/max buffer produced by the previous pass. It uses these values to calculate the split between the two cascades. We use interpolation between linear and logarithmic distribution. After the split is calculated, the split portion of the frustum is projected into the light viewspace and its min/max bounds are calculated. From these bounds a per cascade view and projection matrices are derived and stored in a GPU buffer.

5) Shadowmap passes(2) - Raster

**input** - Per shadow cascade data

**output** - Shadow map cascades (Depth) 32bits -  $2048 * 2048 * 2(\text{cascades})$

**Description:**

- This pass uses the Matrices produced by the previous pass to draw the two shadowmap cascades. Similarly to the prepass we use two separate pipelines to draw the alpha discard geometry and the normal, opaque geometry. The shadowmap is drawn into a single  $4096 * 2048$  texture to allow for a single renderpass. The draw area is set by a dynamic viewport.

6) ESM blur passes (2) - Compute

**input** - Shadow map cascades

**output** - ESM shadow maps 16 bits -  $2048 * 2048 * 2(\text{cascades})$

**Description:**

- This is a two pass separable gaussian blur kernel, that converts the depths stored in the shadowmap into the final values described by the ESM formula. The blur is 5 pixels wide and each pass again uses shared memory to avoid duplicate texel fetches. Each threadgroup (aligned in either vertical or horizontal strips) first fetches the entire sampled area into shared memory. This is then used as a cache to compute the vertical/horizontal blur.

7) Main Draw Pass - Raster

**input** - SSAO texture

**input** - Compressed world space normals

**input** - ESM shadow maps

**input** - Depth Texture

**output** - Offscreen texture 64bits(RGBA 16bit SFLOAT) - Render Resolution

**output** - Motion vector texture 32bits (RG 16bit SFLOAT) - Render Resolution

**Description:**

- The main pass outputs the color and motion vectors. As opposed to the shadow pass and prepass we no longer need two pipelines. This is because we use the depth texture produced by the prepass and set our depth test as DEPTH\_EQUAL. All the fragments that would have been discarded by the alpha threshold have already been discarded in the prepass and thus will be rejected by the depth test. The motion vectors are computed using the previous frame and current view projection matrices without the jitter applied.

#### 8) Fog Pass - Compute

**input** - Depth texture

**input** - Offscreen texture

**output** - Offscreen texture

**Description:**

- The fog pass samples the depth texture to get the depth in the scene. It uses a simple analytical height based formula to calculate fog color. This fog is then blended on top of the offscreen texture color in the shader itself.

#### 9) FSR upscale Pass

**input** - Offscreen texture

**input** - Motion vector texture

**output** - Upscaled texture - 64bits (RGBA 16bit SFLOAT) - Display Resolution

**Description:**

- We utilize FSR2 which is openly available to download and use. We have integrated it into the Vulkan backend so the usage is very easy. The motion vectors, depth and offscreen textures are passed along with the target texture into which FSR2 upscales.

#### 10) Blit into swapchain

**input** - Upscaled texture

**output** - Swapchain image

**Description:**

- As the final step the upscaled texture is blitted into the swapchain texture and presented to the screen.

## Additional libraries

fastgltf - <https://github.com/spnda/fastgltf>

Vulkan Memory Allocator -

<https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator>

fmt - <https://github.com/fmtlib/fmt>

freeimage - <https://freeimage.sourceforge.io>

fsr2.2 - <https://github.com/GPUOpen-LibrariesAndSDKs/FidelityFX-SDK>

## Effect Artifacts

Most of the artifacts are caused by shadows. Because the shadow map matrices are fitted to the frustum, as the frustum moves, the shadow maps are repositioned, which causes noticeable shimmer. Additionally small amounts of shimmering on the leaves can also be observable. This is either caused by the shadow map imprecision or by a bug in the implementation.

The artifacts of SSAO can clearly be seen when coming closer to a heavily occluded area. Mainly the pattern of the banding prevention kernel can be visible. This can be prevented by using depth aware blur. This was unfortunately not done due to time constraints.

FSR artifacts can be mainly seen in the disocclusion convergence. Once an area previously blocked by a tree trunk for example becomes visible, a slight shimmer can be seen before the temporal upscaling catches up.

## Where to find effects in the scene

Most effects can best be observed in the mode without albedo applied (Described below in shortcuts). The normal maps are present on pretty much every single piece of geometry so it is very easy to notice their absence when switching them off.

The ambient occlusion is only applied to the secondary sources of light, so it might be easy to miss it with albedo enabled. It is mainly observable in the crowns of the trees and near the crevices the rocks create when meeting ground. To better observe the ambient occlusion it is recommended to force it on even for primary light contribution (Described below in shortcuts). Ambient occlusion will best be visible when disabling the shadows and forcing it on for the primary light sources.

Shadows are clearly visible underneath the forested part of the scene. When disabled the ambient occlusion will also not be applied in the previously shadowed areas. (as everything is now illuminated by the primary light source).

The fog is observable between the distant trees. The height dependance is mainly observable when the camera is closer to the ground. The tree trunks in the distance will be covered by fog but the leaf crown will have less fog applied.

The effects of FSR can mainly be seen by disabling it. Although we implemented mip maps (for albedo textures) there is still some shimmer introduced by the instability of frustum fitted shadow maps. FSR helps alleviate this issue. The performance of the entire application is also increased when FSR is enabled as all resolution dependent passes and resources will be cheaper.

## Controls

### **0** : Disable/Enable albedo

- When disabled albedo will be overridden to (1, 1, 1) so only the lighting contributions are rendered

### **1** : Disable/Enable shadows

- Shadows will not be applied by main pass (note that this only disables the sampling of the shadows, but the shadows themselves are still drawn, thus no perf increase...)

### **2** : Disable/Enable Ambient occlusion

- When disabled ambient occlusion will be set to (1, 1, 1) (similarly to shadows, this is does not effect in any perf increase)

### **LEFT SHIFT + 2** : Force Ambient occlusion

- By default the ambient occlusion is only applied to secondary light sources (ambient light in the shadows and point lights placed throughout the scene). We allow the override of this and force ambient occlusion to also contribute to the light by the main light source (the moon).

**3** : Disable/Enable Normal mapping

- Per fragment modification of normals using normal maps in prepass will no longer be applied. Interpolated geometry normals will be output instead.

**4** : Disable/Enable Fog

- The fog pass will only output the original color in the offscreen buffer

**5 - 9** : FSR Quality settings

- **5** - FSR Scaling factor 1.0 (Display resolution = Render resolution)
- **6** - Scaling factor 1.5 (Display resolution = 1.5 \* Render resolution)
- **7** - Scaling factor 1.75
- **8** - Scaling factor 2.0 (**DEFAULT**)
- **9** - Scaling factor 3.0

**MINUS** : Disable/Enable FSR

- When FSR is disabled the offscreen is put into native resolution (Display resolution = Render resolution) and the FSR upscale pass is skipped. Instead the offscreen texture is directly blitted onto the swapchain. When the FSR is enabled after disabling it stays scaling factor 1.0 until changed by one of the above keybinds.

**M** : Enable/Disable manual camera control

***THE FOLLOWING CONTROLS ONLY DESCRIBE MANUAL CAMERA. As the movement of the main camera is now automatic, these controls are not available, unless explicitly enabling manual camera movement.***

**Escape** : Enable/Disable free fly camera mode to be able to move

**W** : Move forward

**A** : Move left

**S** : Move backward

**D** : Move right

**Space** : Go up

**Left Control** : Go down

**Left Alt** : Move slower

**Left Shift** : Move faster

**C** : Zoom in while pressed

## Note

As it is right now all the paths to the resources (compiled shaders and gltf model) are hardcoded. The packaged executable runs fine, but when building the application from source the working directory has to be set to the root of the repository. The assets have to be added to the correct path : at the root of the repository.

I hope it is okay that I included FSR in the demo. I made an option to disable it, in case there is an issue with this. The application should hopefully still meet the frame time requirements (I could only measure on GTX 1080 where I got frametimes between 7-8ms, hopefully 1060 is not twice as slow).