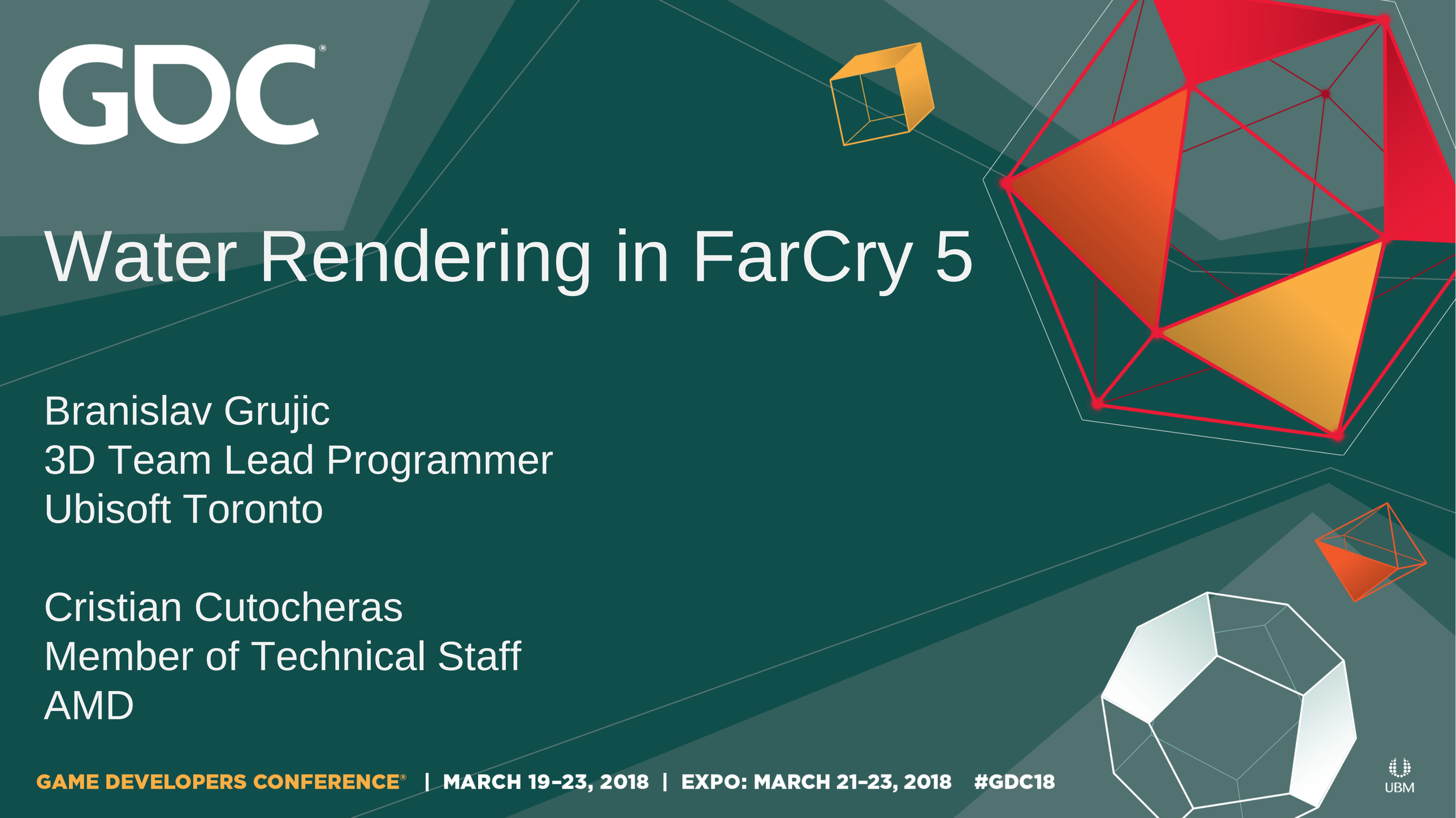


Water Rendering in FarCry 5

Branislav Grujic
3D Team Lead Programmer
Ubisoft Toronto

Cristian Cutocheras
Member of Technical Staff
AMD

GAME DEVELOPERS CONFERENCE® | MARCH 19–23, 2018 | EXPO: MARCH 21–23, 2018 #GDC18

Agenda

- Introduction
- History of water in previous FarCry games
- Montana Overview
- Engine, Tools & Rendering Goals
- Single Frame Rendering
- Optimizing with Half Precision Math
- Problems Encountered, Debugging and Future


N
0
2
0

Montana

Tech Overview

- Engine
  - Data Generation and Streaming
  - Water Queries API
- Tools
  - Artist driven tools
- Rendering
  - Single frame walkthrough

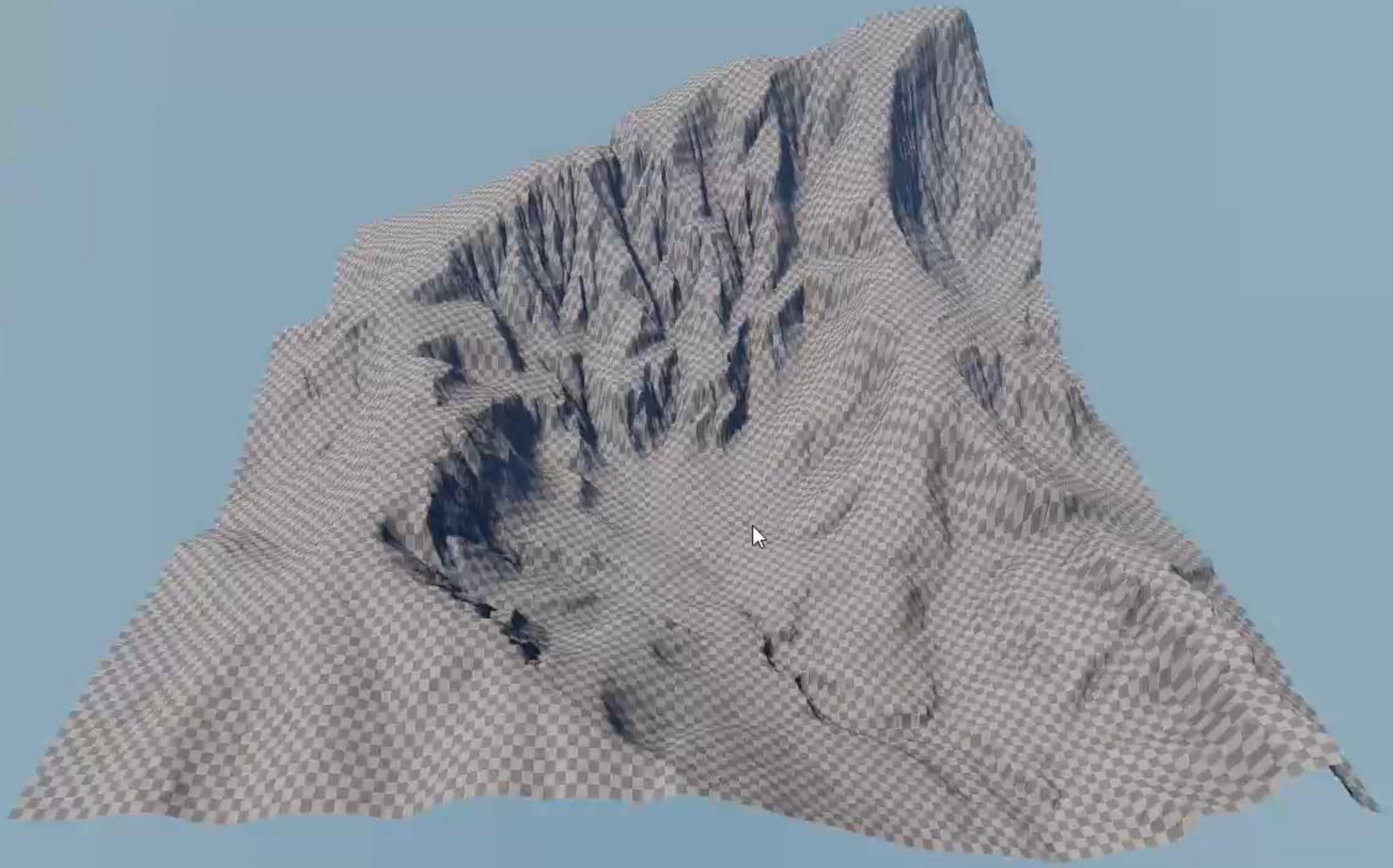
Engine

- Simple API
  - Single Function
- Fast Water Queries
  - Water Quad tree using bitfield (Water Planes + Ocean)
  - Baked Water height map streamed in (Lakes/Rivers/Waterfalls)
- Flow & Physics
  - Water flow map streamed (CPU)
- Material Access
  - Baked material map (CPU)

```
namespace WaterHelpers
{
    GRAPHICSRENDERER_DLL ndFloat GetGlobalWaterLevel(ndVec3 const& pos, ndBool precise)
    {
        //NOMAD_PROFILE(WaterHelpers::GetGlobalWaterLevel);
        ndFloat waterLevel = -10000.0f;

        {
            waterLevel = C3DEngine::GetInstance()->GetWaterLevel(pos).m_waterLevel;
        }

        return waterLevel;
    }

    ndFloat GetGlobalWaterLevelRender(ndVec3 const& pos)
    {
        //NOMAD_PROFILE(WaterHelpers::GetGlobalWaterLevel);
        ndFloat waterLevel = -10000.0f;

        {
            waterLevel = C3DEngine::GetInstance()->GetWaterLevel(pos, true).m_waterLevel;
        }

        return waterLevel;
    }

    GRAPHICSRENDERER_DLL void GetWaterFlowDirection(ndVec3 const& position, ndVec2& flowDir)
    {
        //NOMAD_PROFILE(WaterHelpers::GetWaterFlowDirection);
        CWaterManager::GetInstanceRead()->GetWaterFlowDirection(position, flowDir);
    }
}
```

Tools

- Easy to Use
- Fast Iteration
- Procedural Generation

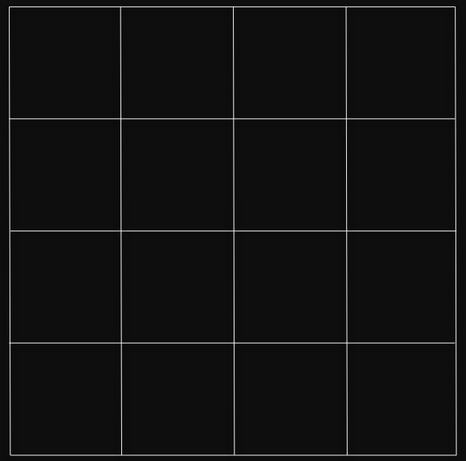


Rendering

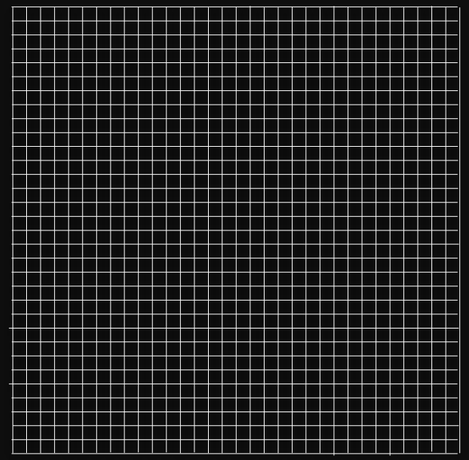
- Screen Space Tessellation
- Per pixel material with blending
- Compute Driven (async ♡)
- Flow Maps with foam


FARCRY
ARCADE

Idea




FBM
FBMTexture
FBM_LittleWaves_array.png
BC7_MaskMap No MipMap Array
FBMTexture Index
0
Amplitude
0.08
Roughness
0.3
Speed
0.3
Scale
0.37
Noise Strength
0
Noise Tiling
0.01



Materials

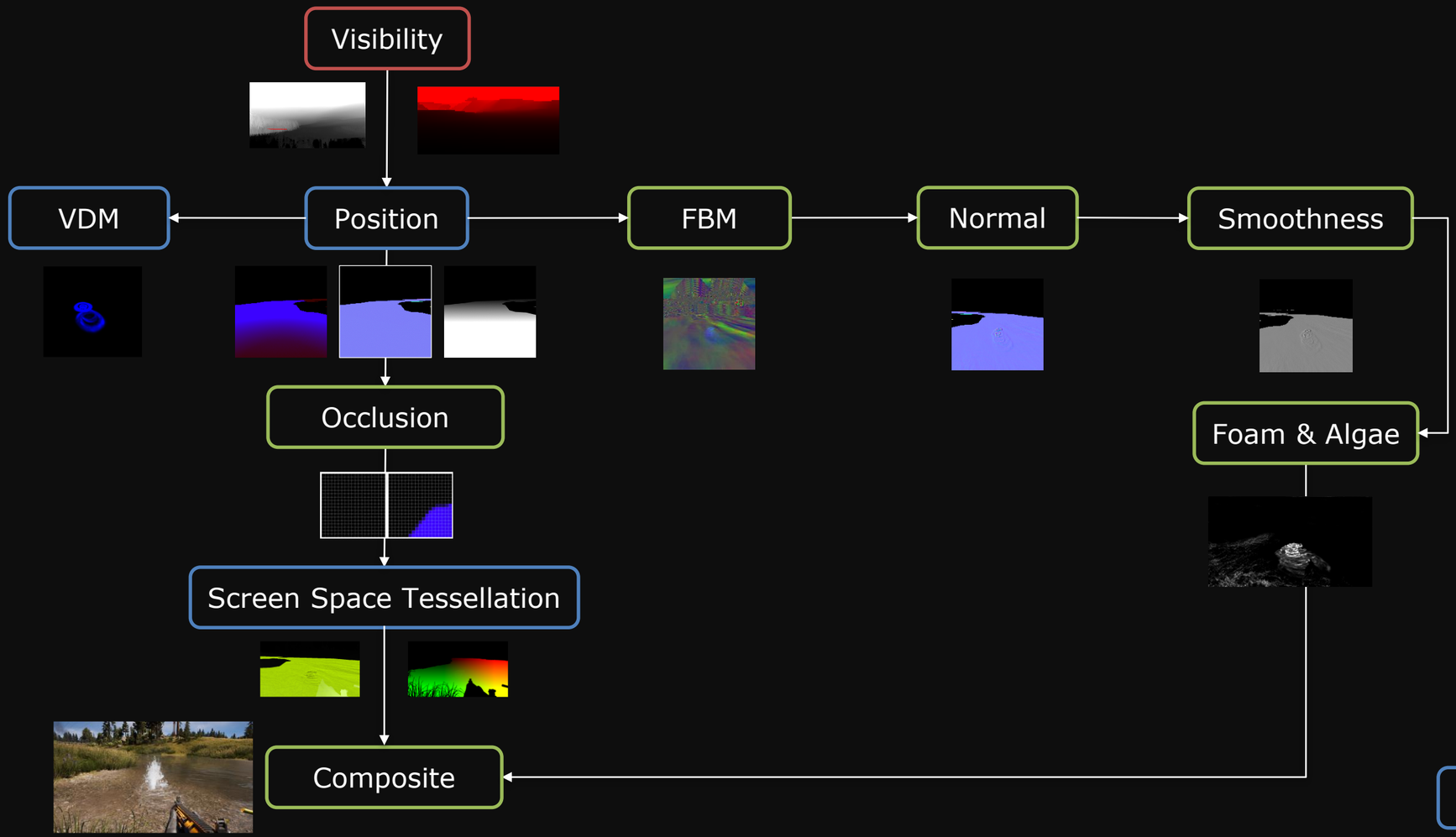

```
struct WaterMaterialData
{
    float2 baseTiling;
    float2 waterDistortion;
    float4 baseColor;
    float4 caustics;
    float4 lightBeamAttenuation;
    float4 fbmData;
    float2 fbmData2;
    float2 flowmapPhase;
    float4 lightIrradianceRatio;
    float4 foamParameter;
    float4 shorelineFoam;
    float4 textureIndex;
    float4 algaeData;
};
```

Materials


```
struct WaterMaterialData
{
    float2 baseTiling;
    float2 waterDistortion;
    float4 baseColor;
    float4 caustics;
    float4 lightBeamAttenuation;
    float4 fbmData;
    float2 fbmData2;
    float2 flowmapPhase;
    float4 lightIrradianceRatio;
    float4 foamParameter;
    float4 shorelineFoam;
    float4 textureIndex;
    float4 algaeData;
};
```

Material Structure Buffer

```
[numthreads(1, 1, 1)]
void ComputeShaderFunc(uint3 dispatchThreadId : SV_DispatchThreadID)
{
    WaterMaterialData data;
    data.baseTiling          = BaseTiling.xy;
    data.baseColor           = float4(BaseColor.xyz, AlgaeNormalStrength);
    data.lightBeamAttenuation = LightBeamAttenuation;
    data.fbmData             = float4(fbmAmplitude, fbmRoughness, fbmSpeed, fbmScale);
    data.fbmData2            = float2(fbmNoiseStrength, fbmNoiseTiling);
    data.flowmapPhase        = float2(FlowmapSpeedScale, FlowMapEnabled);
    data.lightIrradianceRatio = float4(LightIrradianceRatio.xyz, UnderWaterDepthScale);
    data.foamParameter       = foamParameter;
    data.waterDistortion     = float2(WaterDistortion, UnderWaterDistortion);
    data.caustics            = float4(CausticsScale, CausticsIntensity, Extinctions.x, Extinctions.w);
    data.textureIndex        = float4((float)FBMTextureIndex, FlowmapStretchReduction, SunShadowScale, 0);
    data.algaeData           = float4(AlgaeTiling, AlgaeNoiseTiling, AlgaeIntensity, AlgaeShorelineFalloff);
    data.shorelineFoam       = float4(ShorelineFoamIntensity, ShorelineFoamFalloff, FoamNoiseTiling, 0);

    WaterMaterialBuffer[MaterialStructBufIndex] = data;
}
```


Visibility
VDM
Position
FBM
Normal
Smoothness
Occlusion
Foam & Algae
Screen Space Tessellation
Composite
PS
CS
PS+CS

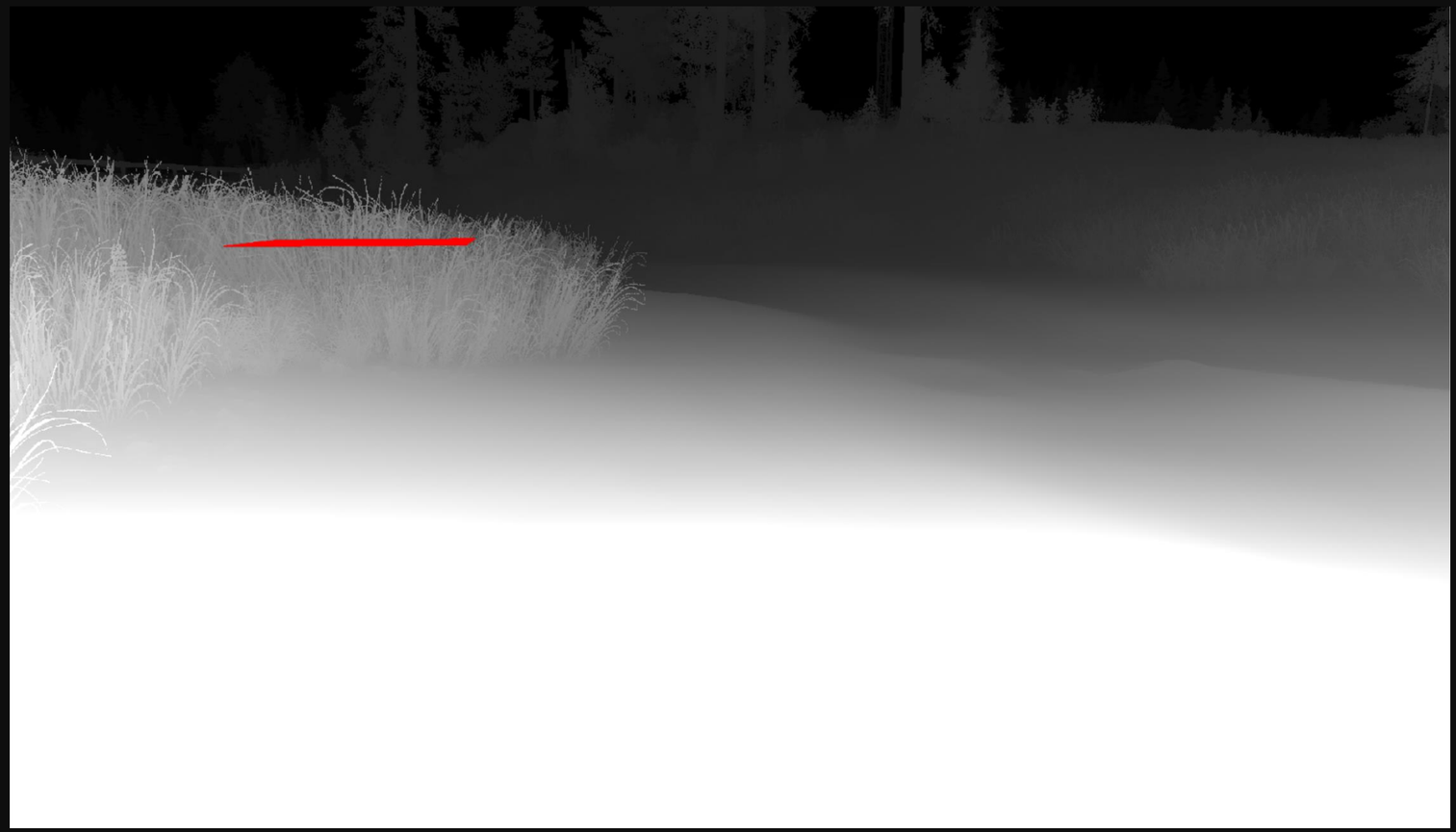
PS

CS

PS+CS

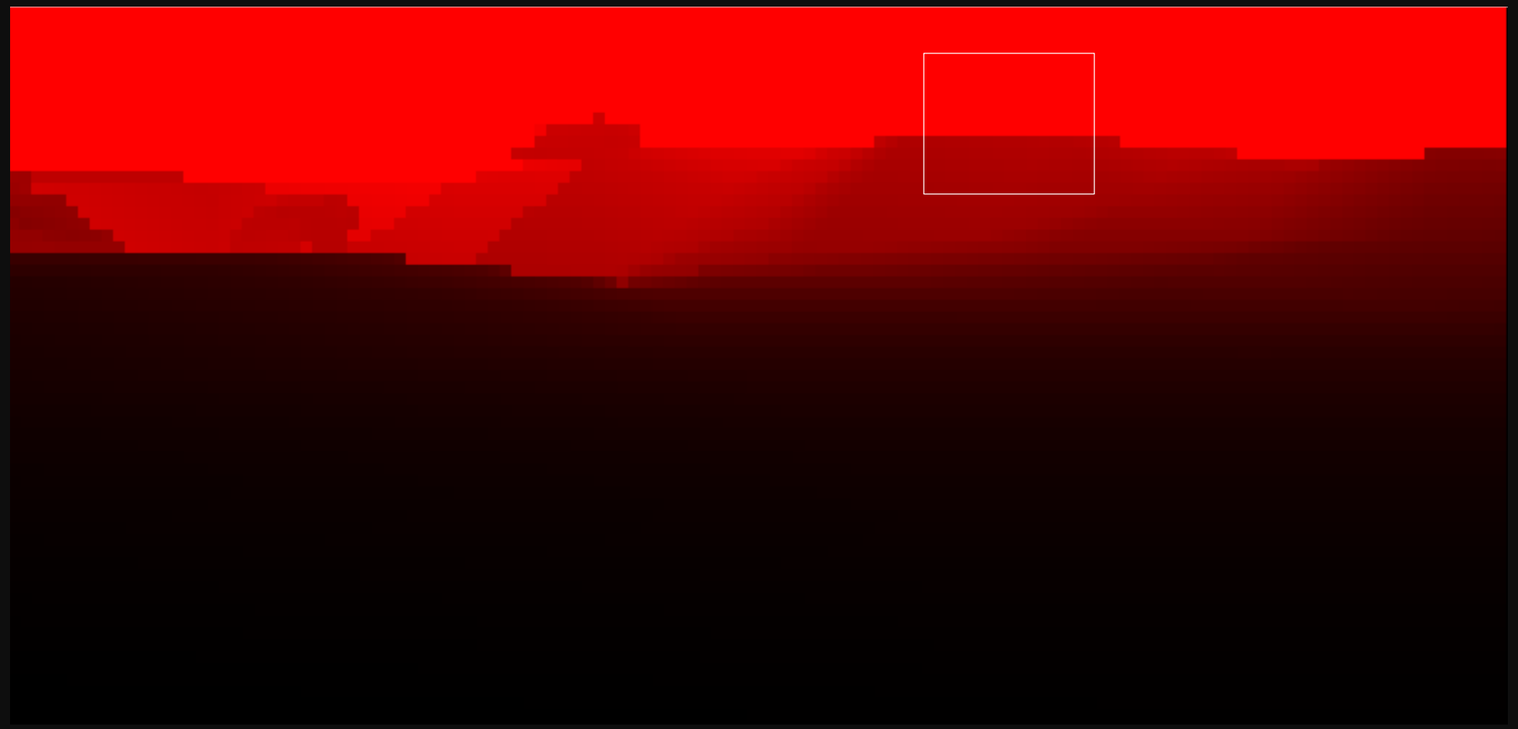
Visibility

- Water near player
  - Occlusion Queries
  - Render AABB in place of water mesh
  - Conditional Rendering Approach
  - Stores Query per mesh instance



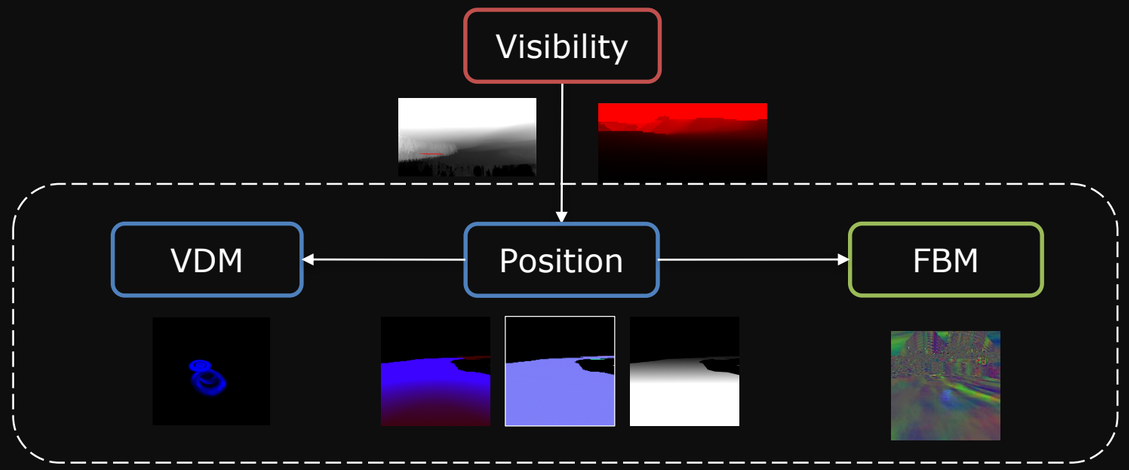
Visibility

- Water Vista
  - Flat Water (simple)
  - Height Map Water (test height map)
  - Per sector occlusion
  - AABB test against occlusion buffer
  - Builds indirect draw arguments buffer




N
BTM BOOST
Game paused!

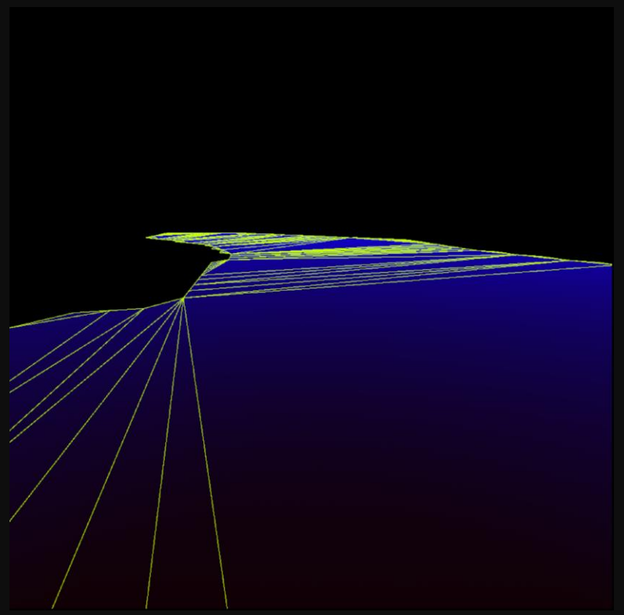

N
BTM BOOST
Game paused!



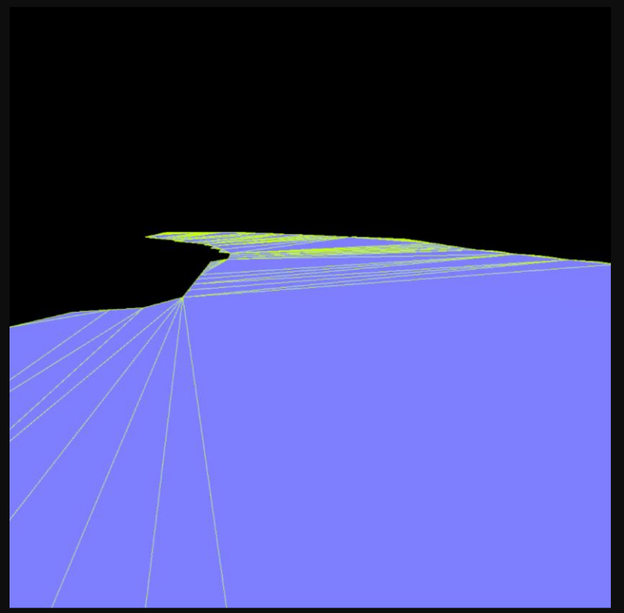
PS CS PS+CS

Position

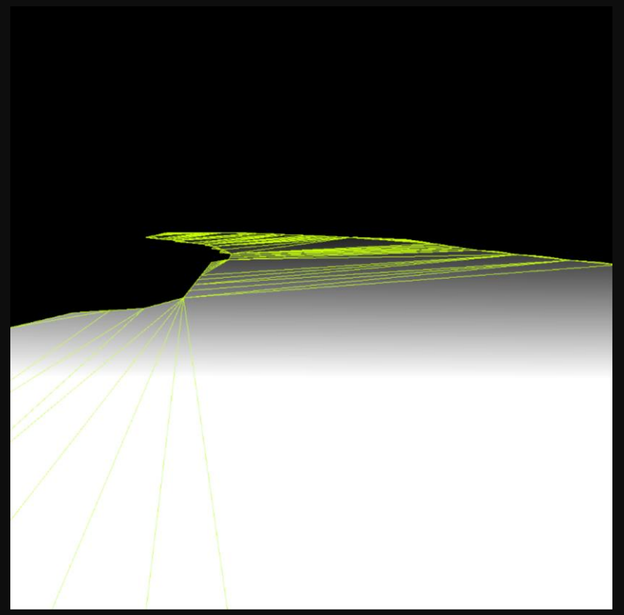
Data



Mesh Normal

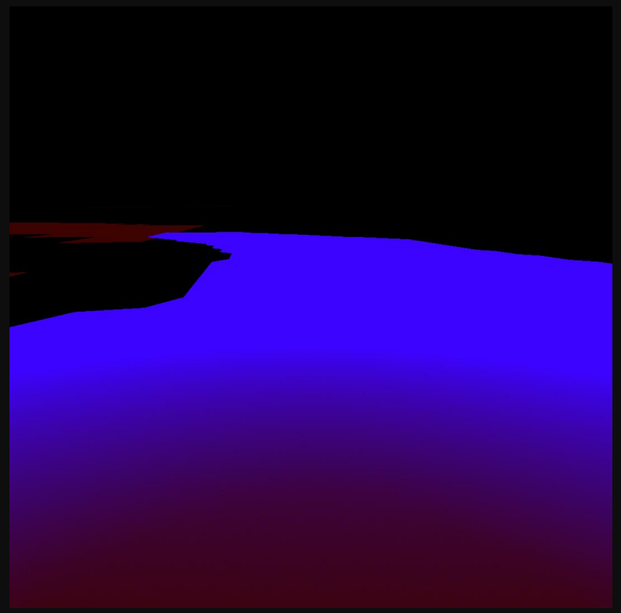


Depth

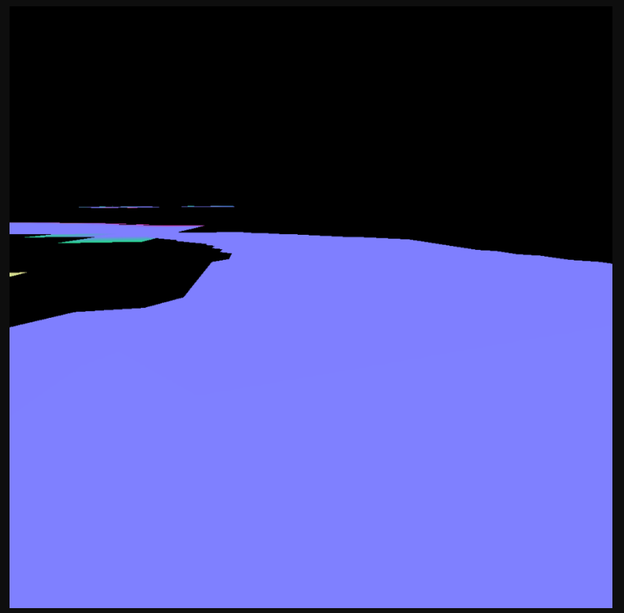


Position

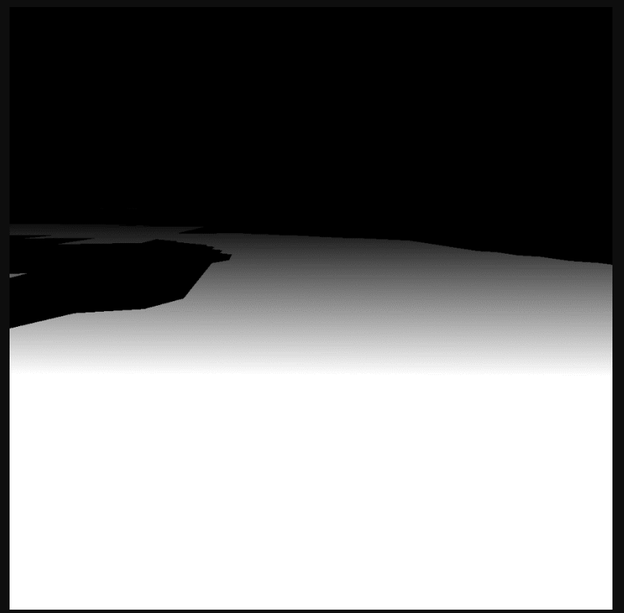
Data



Mesh Normal



Depth



Position

```
output.color0 = PackWaterData(shaderID, MaterialStructBufIndex, PackMipLevel(algaeMipLevel), PackMipLevel(foamMipLevel));
output.color1.xyz = PackNormal(input.normalContext.WSNormal.xyz);
```

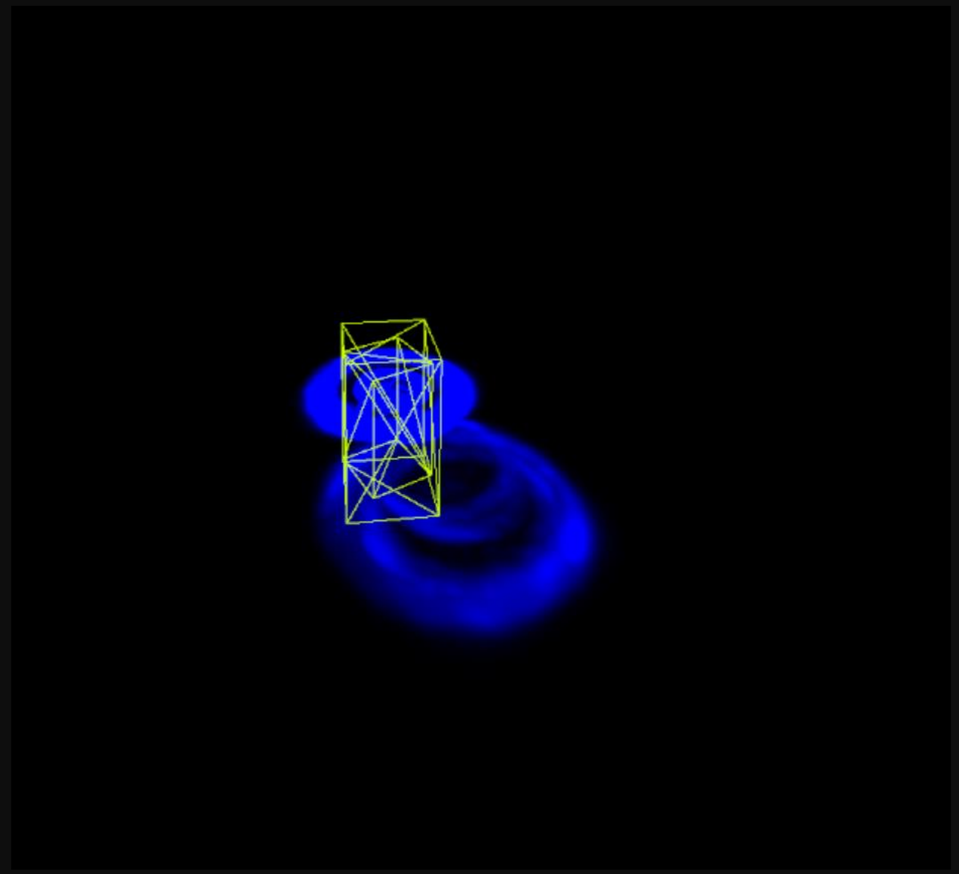
```
struct WaterMaterialData
{
    float2 baseTiling;
    float2 waterDistortion;
    float4 baseColor;
    float4 caustics;
    float4 lightBeamAttenuation;
    float4 fbmData;
    float2 fbmData2;
    float2 flowmapPhase;
    float4 lightIrradianceRatio;
    float4 foamParameter;
    float4 shorelineFoam;
    float4 textureIndex;
    float4 algaeData;
};
```

```
struct WaterMaterialData
{
    float2 baseTiling;
    float2 waterDistortion;
    float4 baseColor;
    float4 caustics;
    float4 lightBeamAttenuation;
    float4 fbmData;
    float2 fbmData2;
    float2 flowmapPhase;
    float4 lightIrradianceRatio;
    float4 foamParameter;
    float4 shorelineFoam;
    float4 textureIndex;
    float4 algaeData;
};
```

```
struct WaterMaterialData
{
    float2 baseTiling;
    float2 waterDistortion;
    float4 baseColor;
    float4 caustics;
    float4 lightBeamAttenuation;
    float4 fbmData;
    float2 fbmData2;
    float2 flowmapPhase;
    float4 lightIrradianceRatio;
    float4 foamParameter;
    float4 shorelineFoam;
    float4 textureIndex;
    float4 algaeData;
};
```

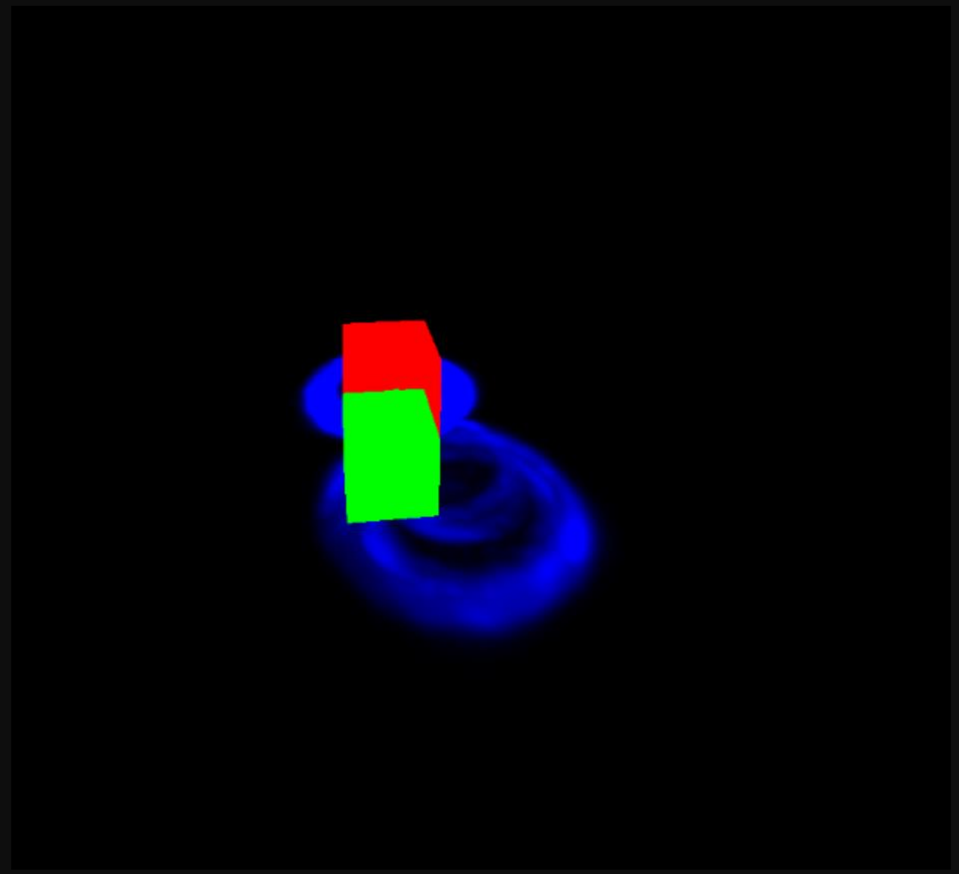
VDM

- Particle Editor Extension
  - Artist friendly workflow
- Projected Box Decal
  - PositionFromDepth to project onto water
  - Invert to Object Space for applying uv's
  - Clip Off Screen Pixels
  - Sample Displacement Texture
  - Animation Lerp
  - **Fade displacement towards edge of box**
  - Max Alpha Blend



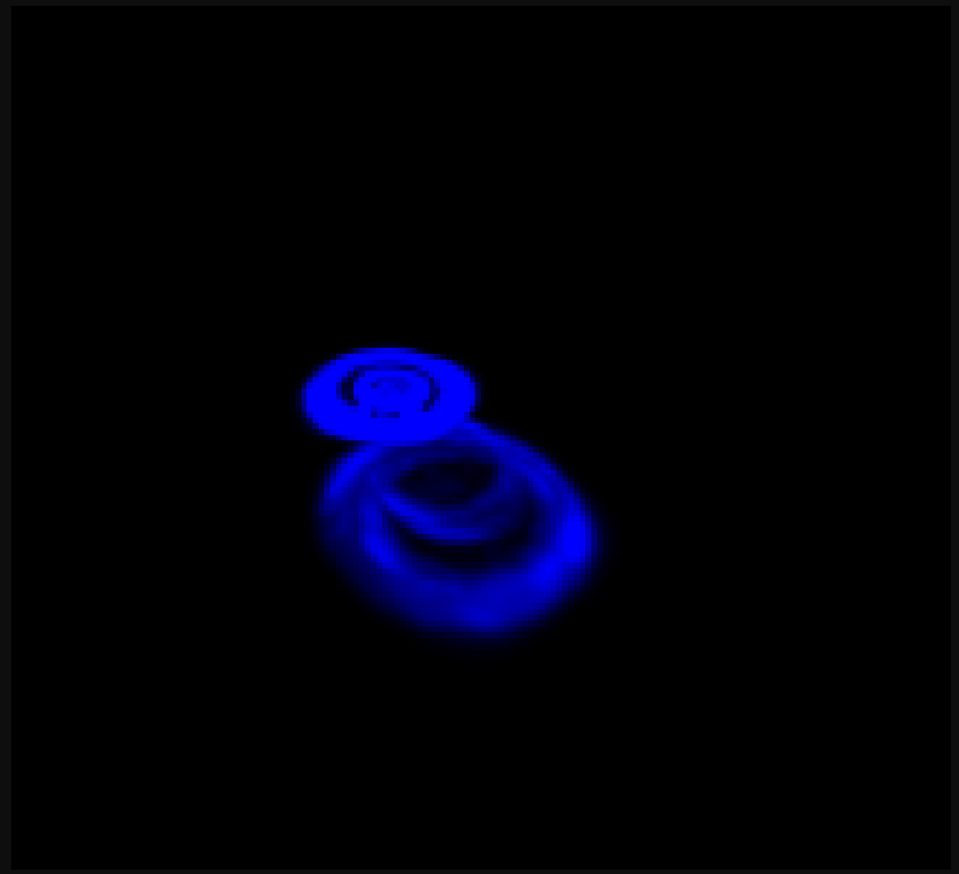
VDM

- Particle Editor Extension
  - Artist friendly workflow
- Projected Box Decal
  - PositionFromDepth to project onto water
  - Invert to Object Space for applying uv’s
  - Clip Off Screen Pixels
  - Sample Displacement Texture
  - Animation Lerp
  - **Fade displacement towards edge of box**
  - Max Alpha Blend



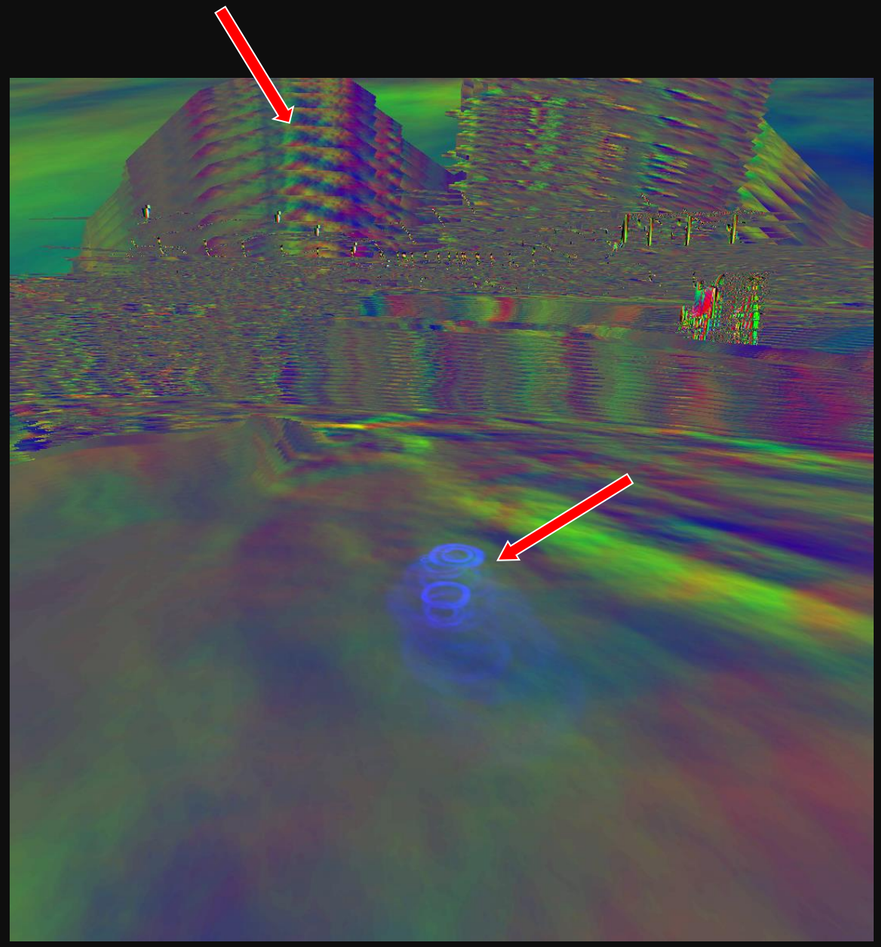
VDM

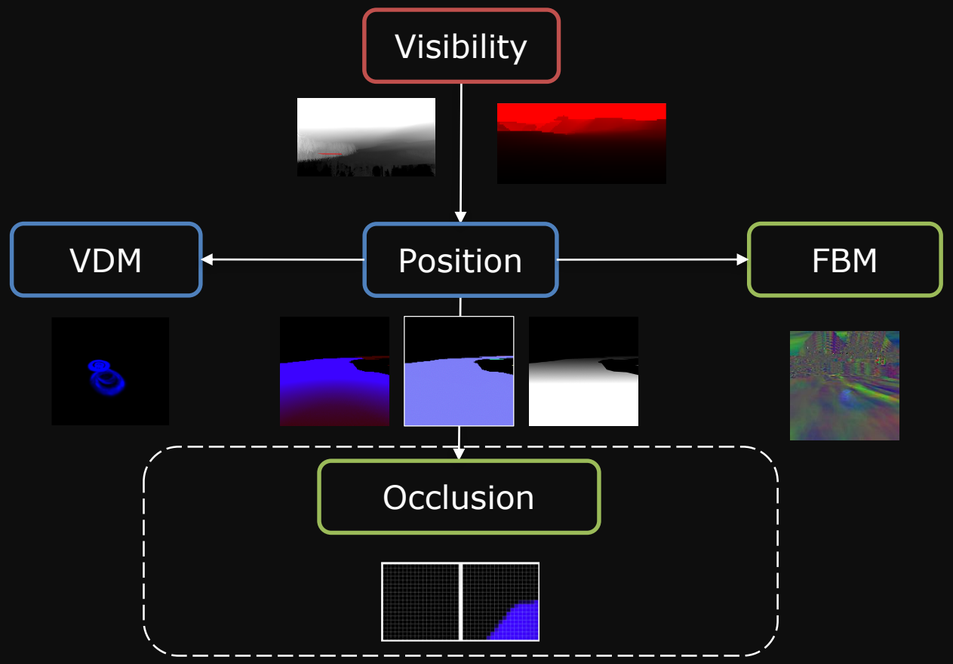
- Particle Editor Extension
  - Artist friendly workflow
- Projected Box Decal
  - PositionFromDepth to project onto water
  - Invert to Object Space for applying uv’s
  - Clip Off Screen Pixels
  - Sample Displacement Texture
  - Animation Lerp
  - **Fade displacement towards edge of box**
  - Max Alpha Blend



FBM

- Generate displacement from noise
  - 9 iterations per pixel
  - Each iteration adds more frequency as you double the uv scale
  - LOD distance based (min 3 - max 9)
  - Combines the vector displacement

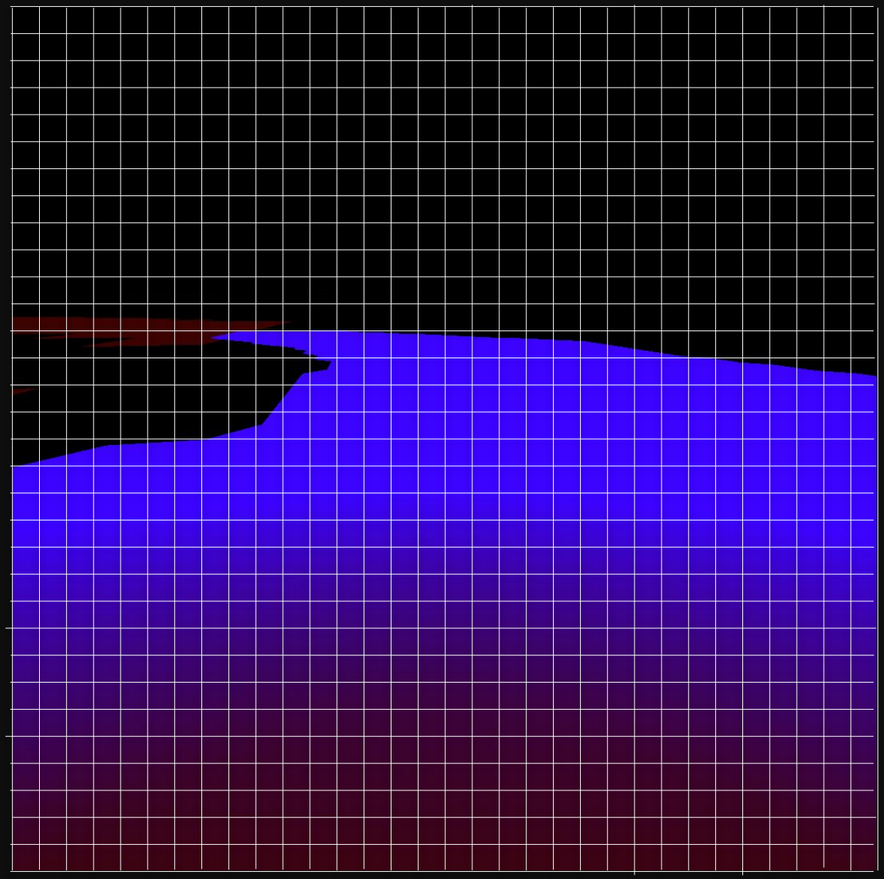




PS CS PS+CS

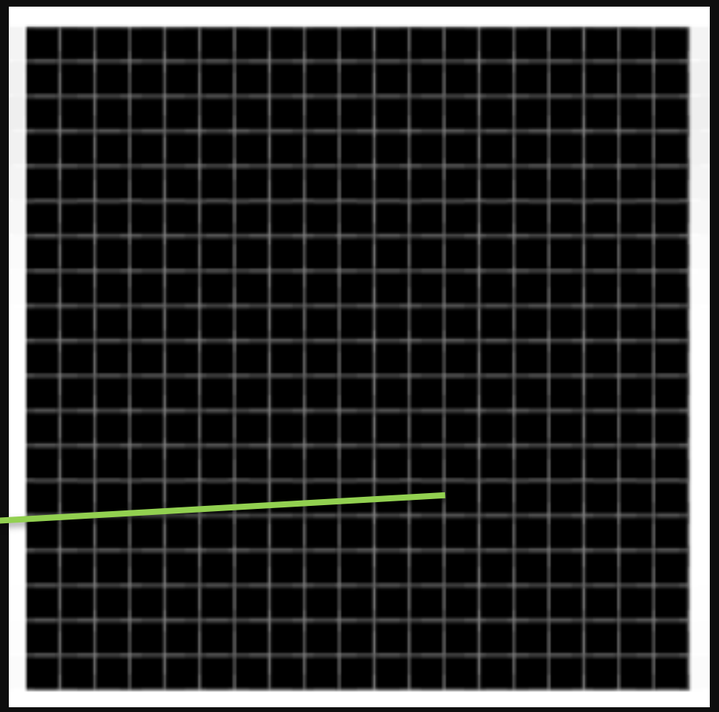
Occlusion

- Divide Screen into 32x32 Tiles
  - Check if tile has water
  - Per tile pixel count



Occlusion

- Count Pixel's
  - Compute TileID from threadID
  - Groupshared memory for intermittent values
  - Store into structure buffer
  - WaveActiveBallot (DX12*)

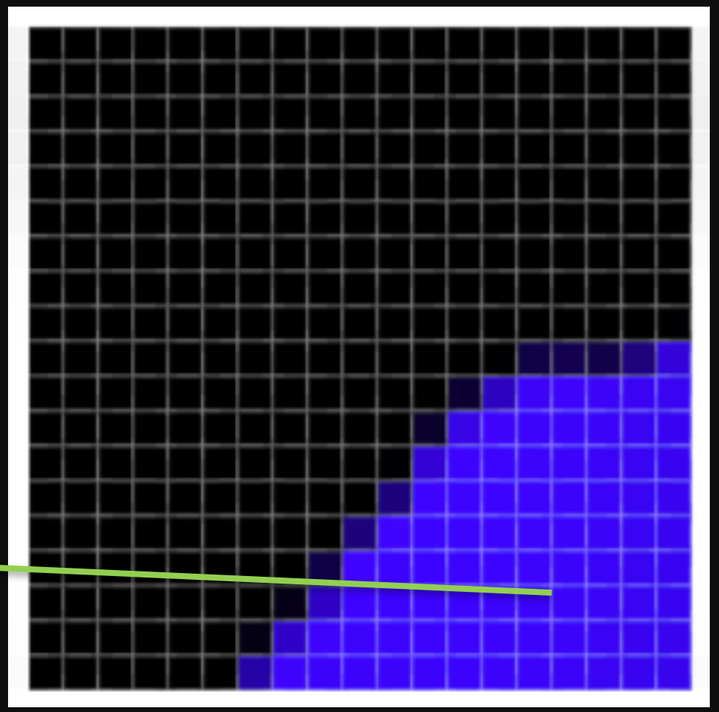


```
InterlockedAdd( GroupPixelCount, ValidShader(waterData) );
```

```
if(groupThreadID.x == 0 && groupThreadID.y == 0)
{
    InterlockedAdd( WaterTileOcclusionBuffer[groupID.z], GroupPixelCount );
}
```

Occlusion

- Count Pixel's
  - Compute TileID from threadID
  - Groupshared memory for intermittent values
  - Store into structure buffer
  - WaveActiveBallot (DX12*)

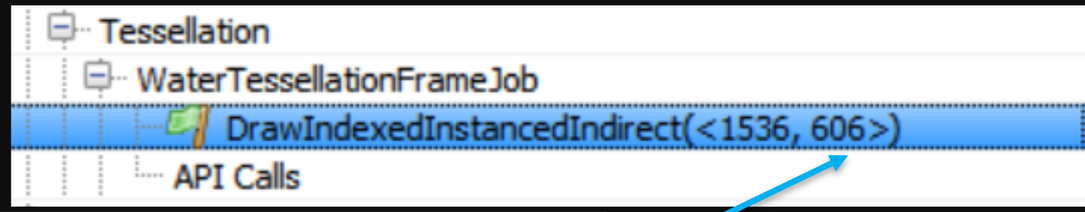


```
InterlockedAdd( GroupPixelCount, ValidShader(waterData) );
```

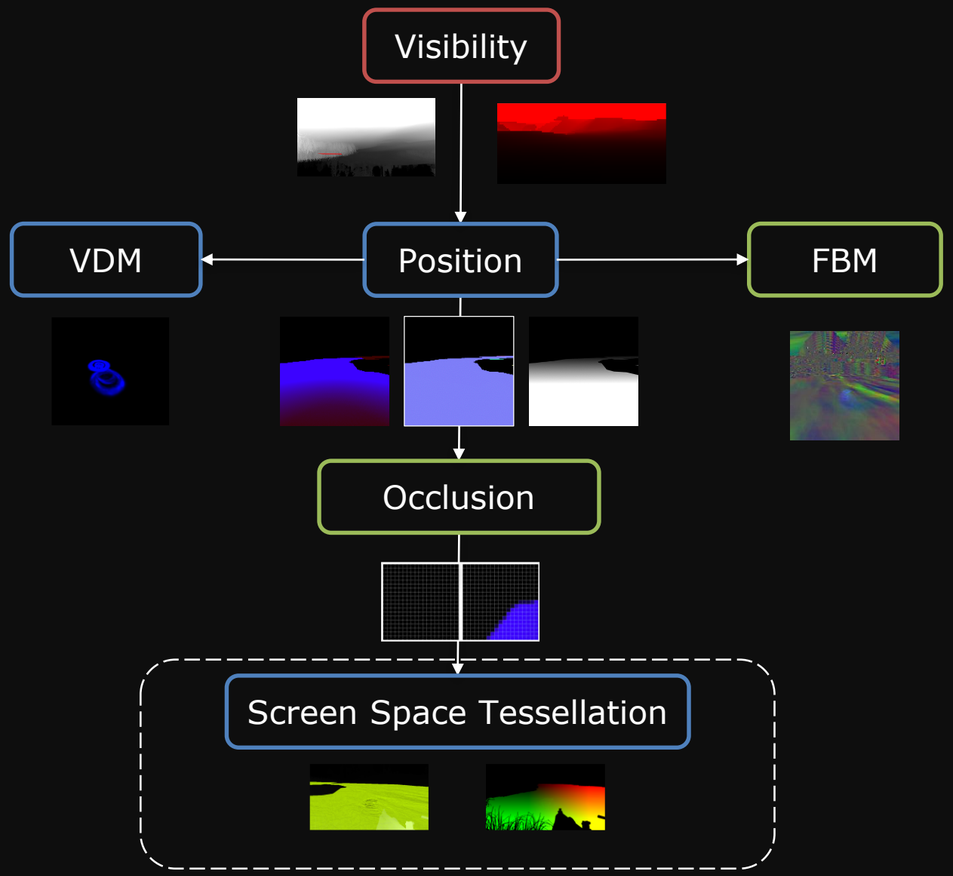
```
if(groupThreadID.x == 0 && groupThreadID.y == 0)
{
    InterlockedAdd( WaterTileOcclusionBuffer[groupID.z], GroupPixelCount );
}
```

Occlusion

- Generate IndirectDrawArgs buffer



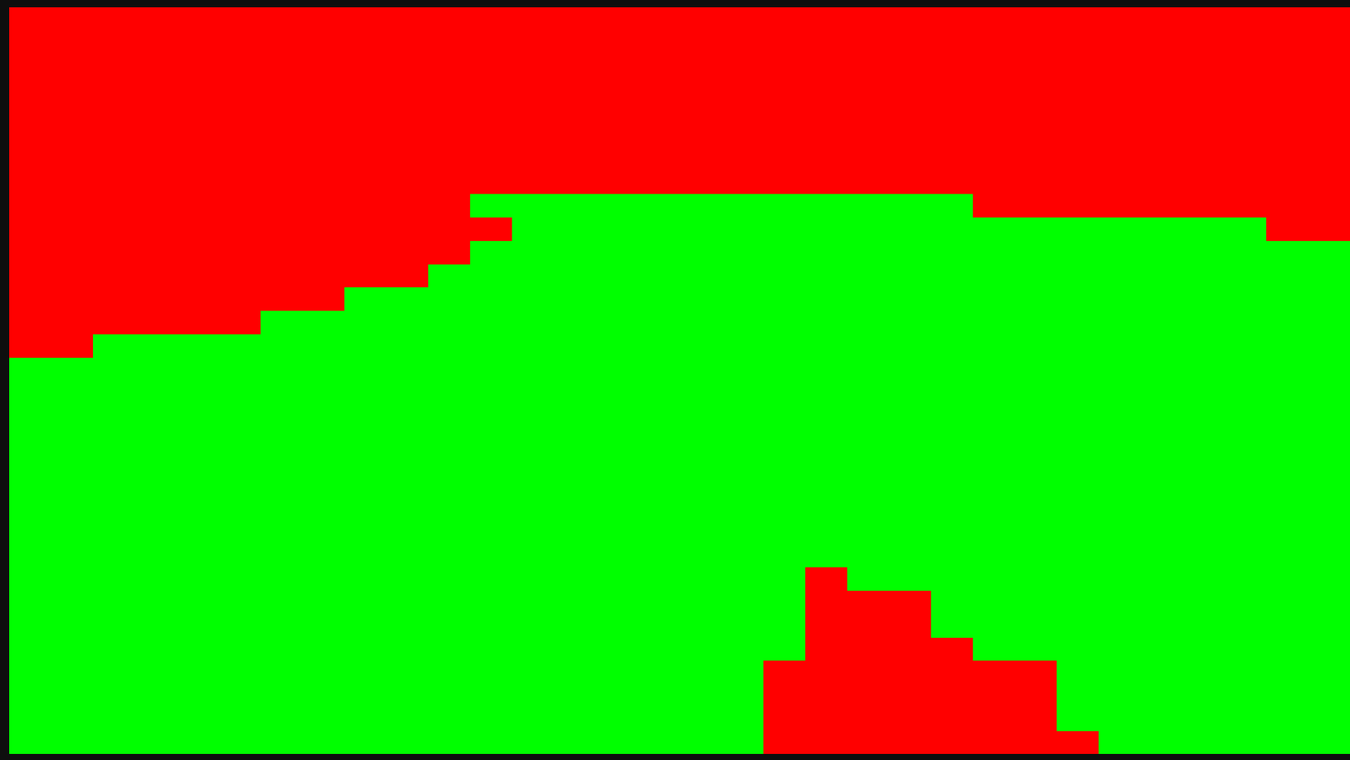

```
int numPixels = WaterTileOcclusionBuffer[dispatchThreadId.x];

// if the specific tile has any pixels written to it, than we must add that instance to the queue
if (numPixels > 0)
{
    //write the corresponding indirect draw arguments
    //increase instance count by 1
    uint indexToStore = 0;
    InterlockedAdd(WaterIndirectDrawArgs[1], 1, indexToStore);

    // copy the data from the right buffer
    WaterTileDataOutput[indexToStore] = dispatchThreadId.x;
}
```


Visibility
VDM
Position
FBM
Occlusion
Screen Space Tessellation

PS CS PS+CS

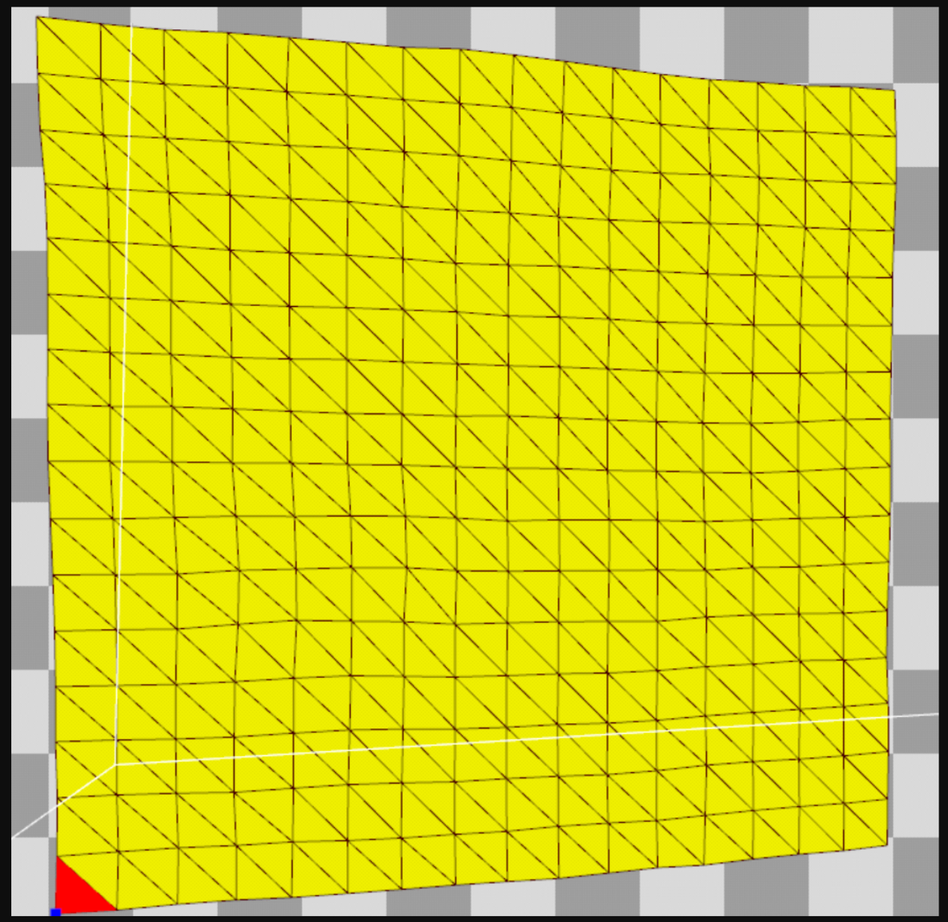
Tessellation

- Render tessellated mesh per tile
  - Mesh Vertex density is buffer resolution / 32
    - 512 / 32 = 16x16 quads
  - DrawIndexedInstancedIndirect
  - **Constant density tessellation**



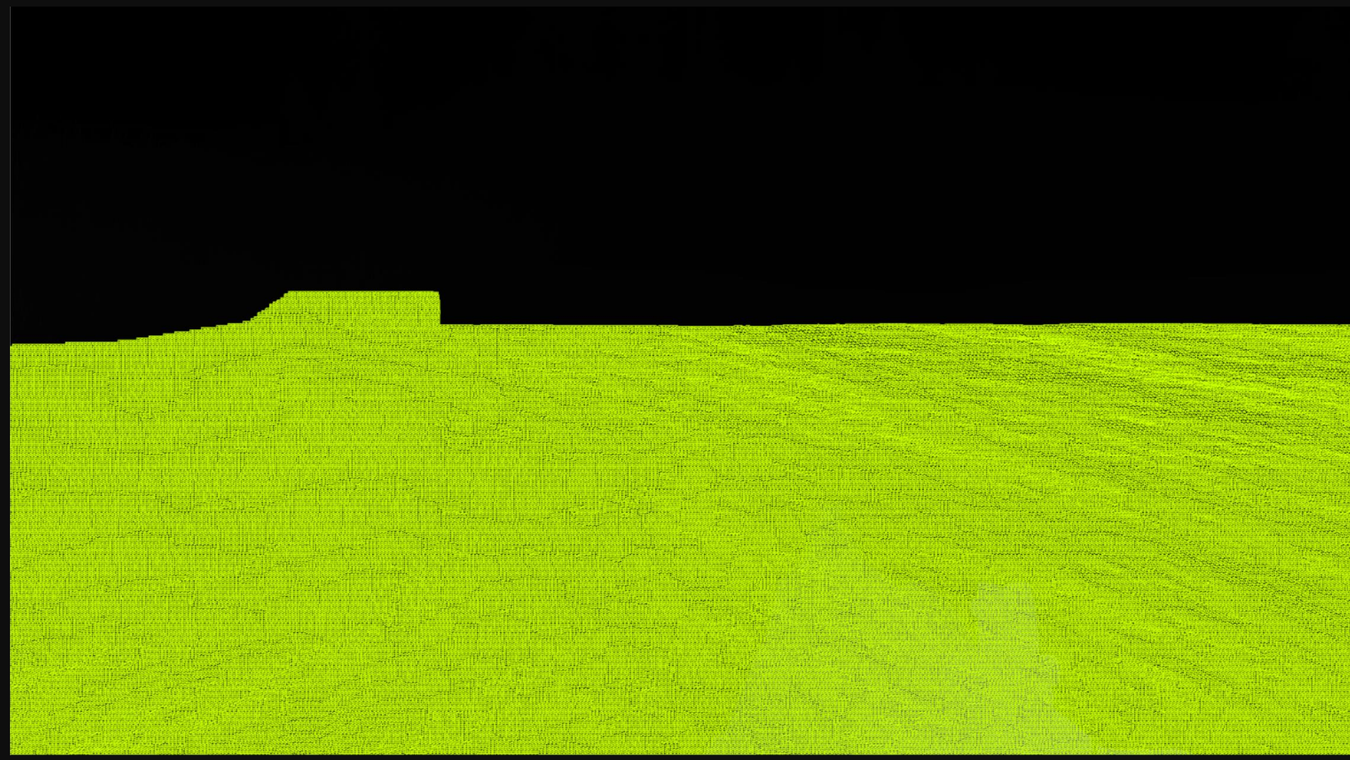
Tessellation

- Render tessellated mesh per tile
  - Mesh Vertex density is buffer resolution / 32
    - 512 / 32 = 16x16 quads
  - DrawIndexedInstancedIndirect
  - **Constant density tessellation**



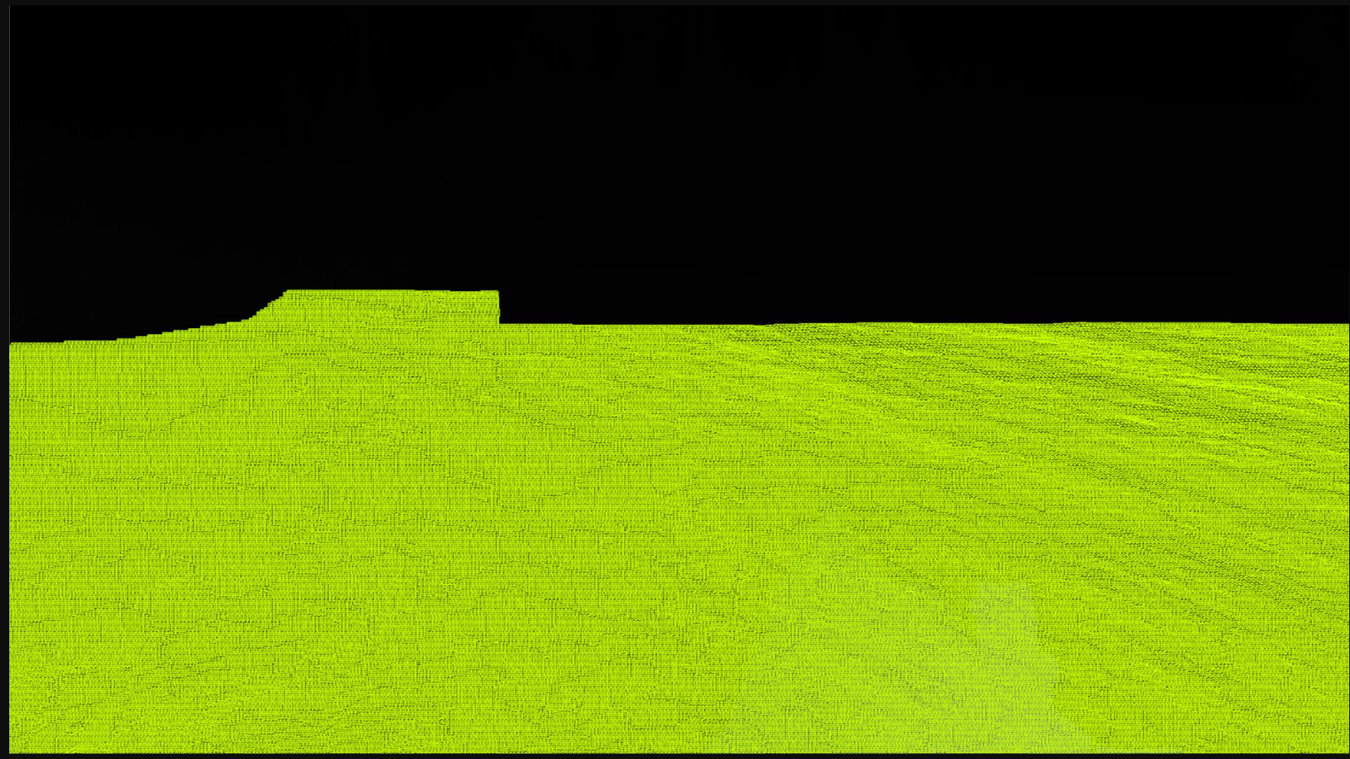
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)



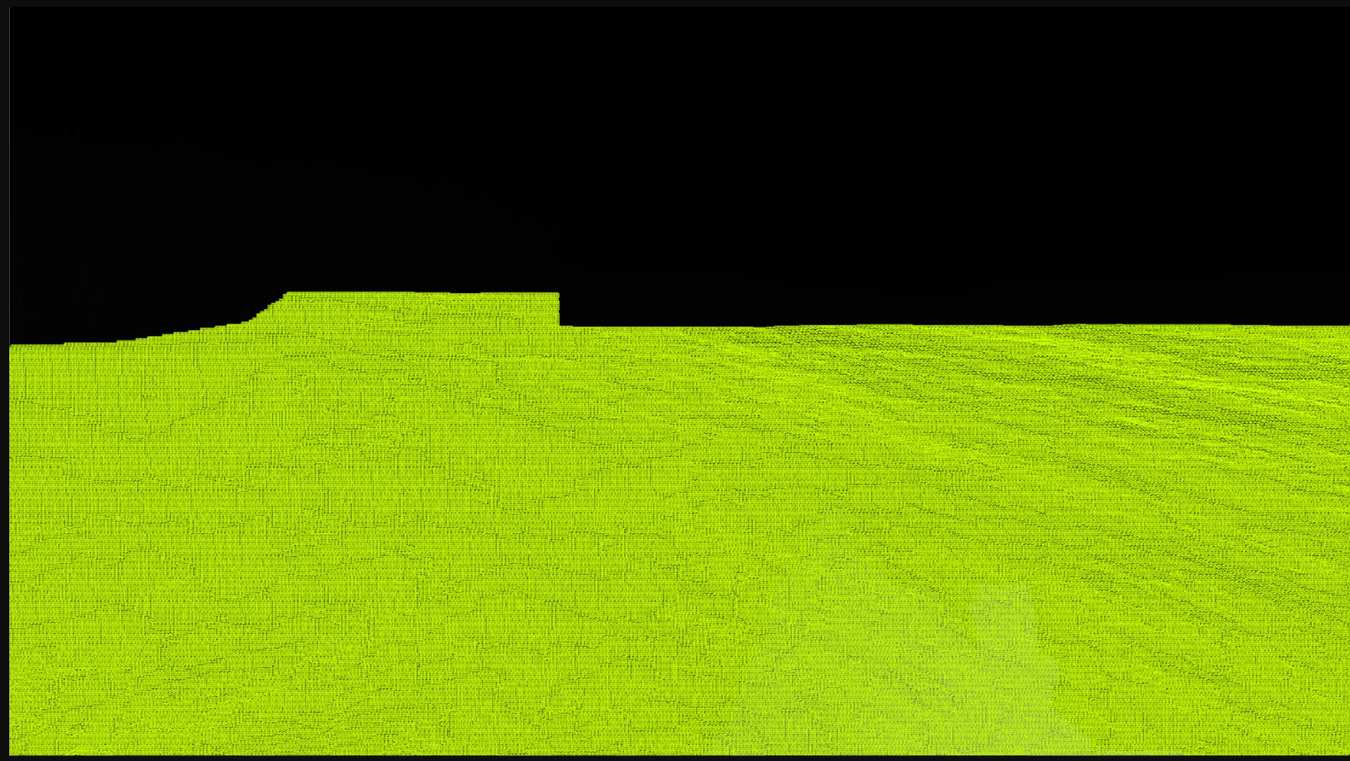
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)



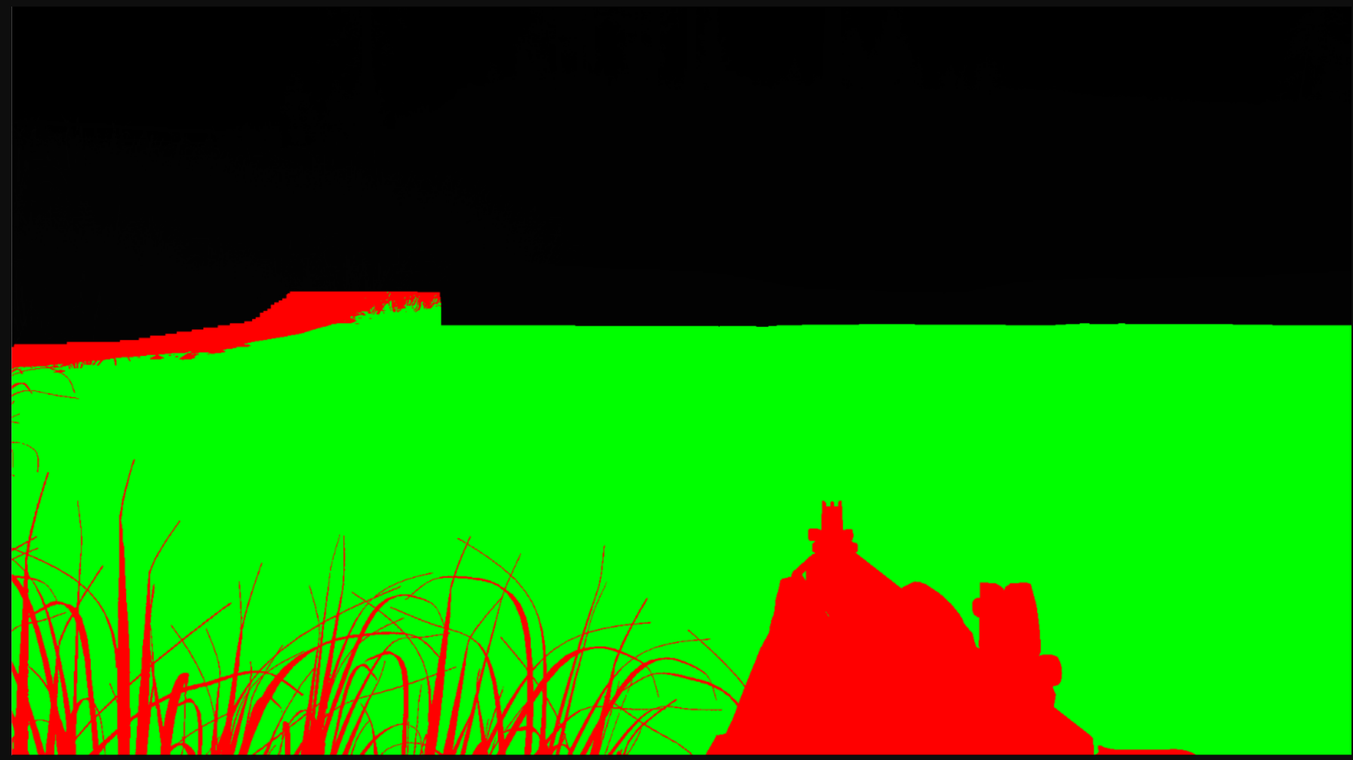
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)



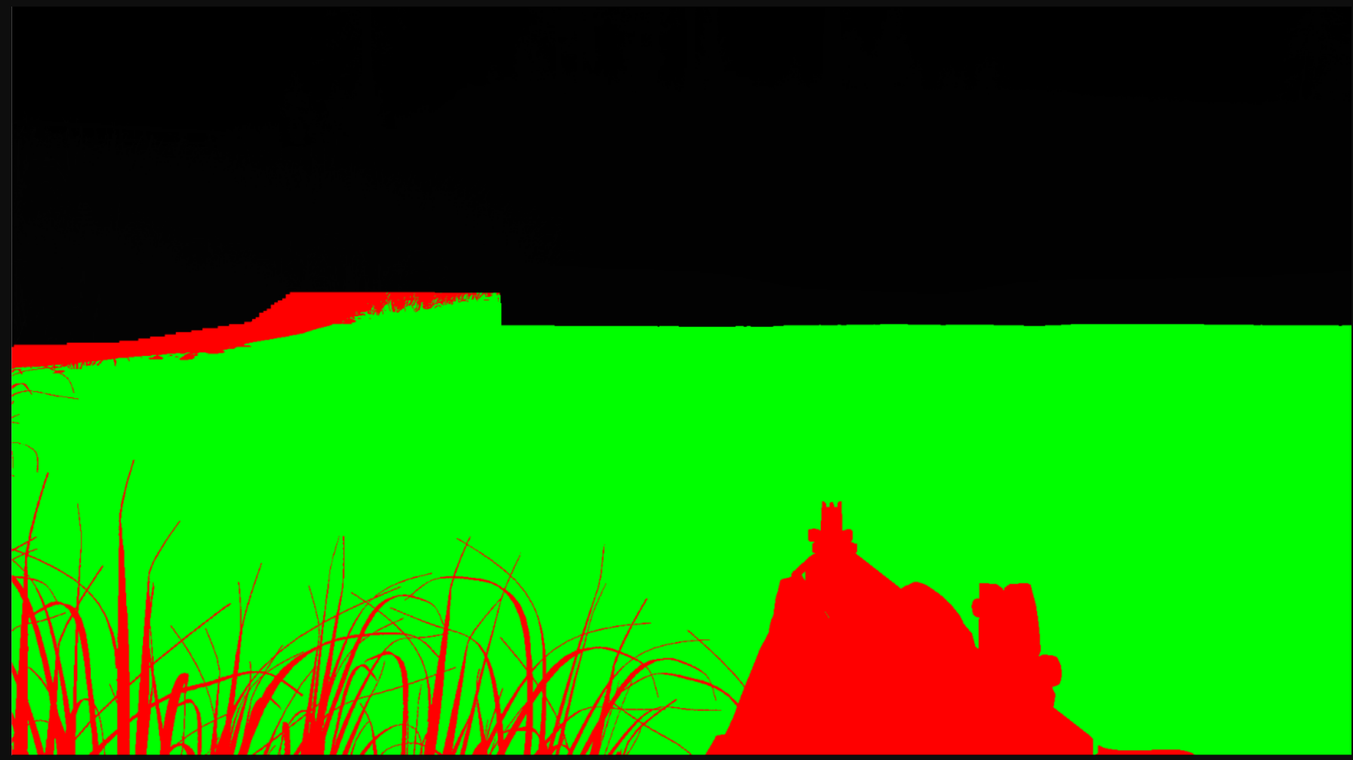
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)



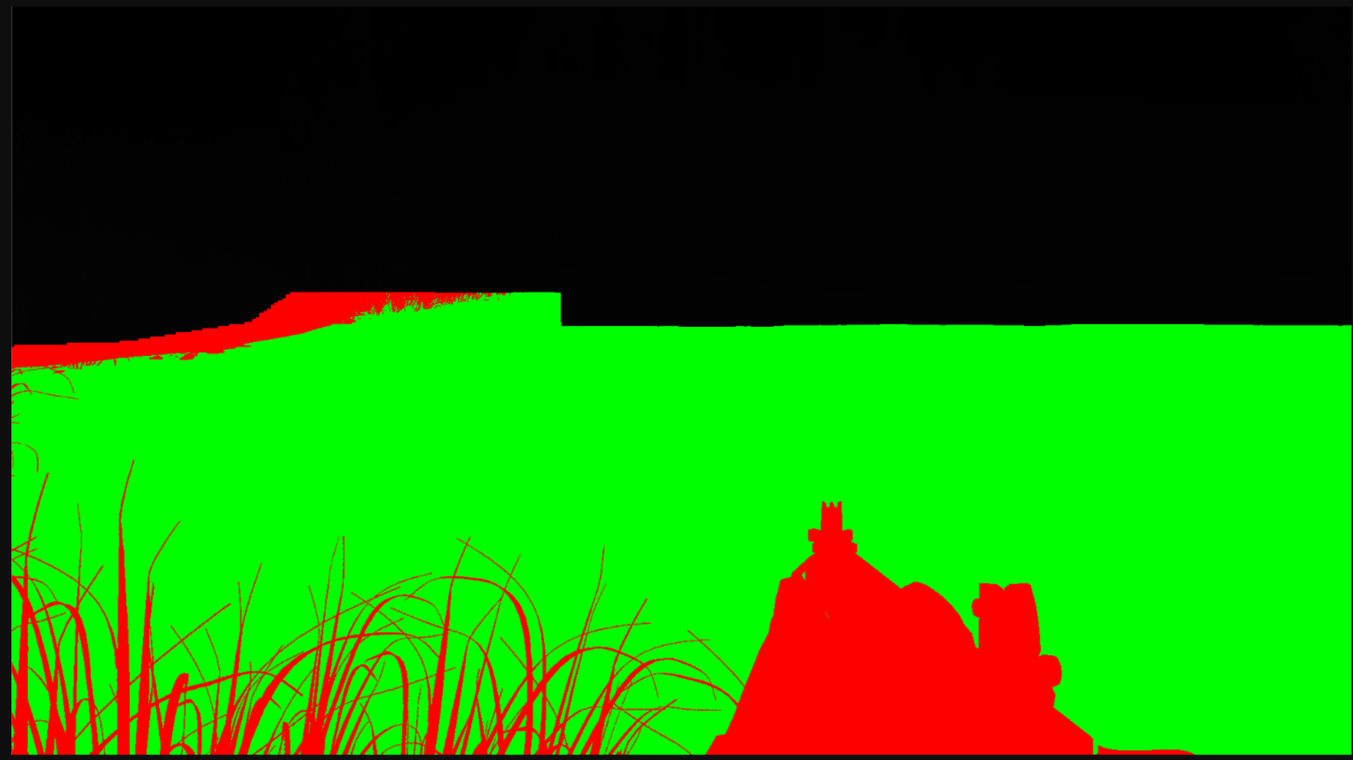
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)



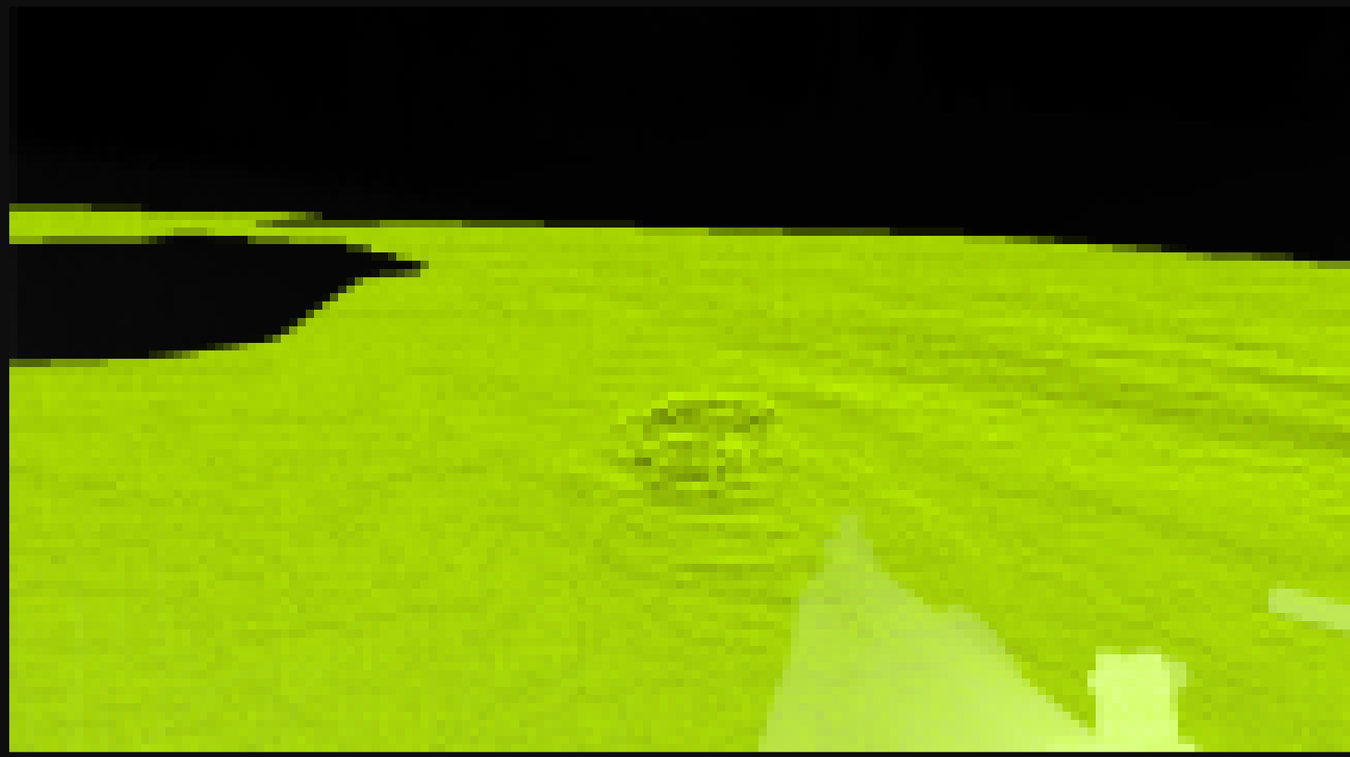
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)



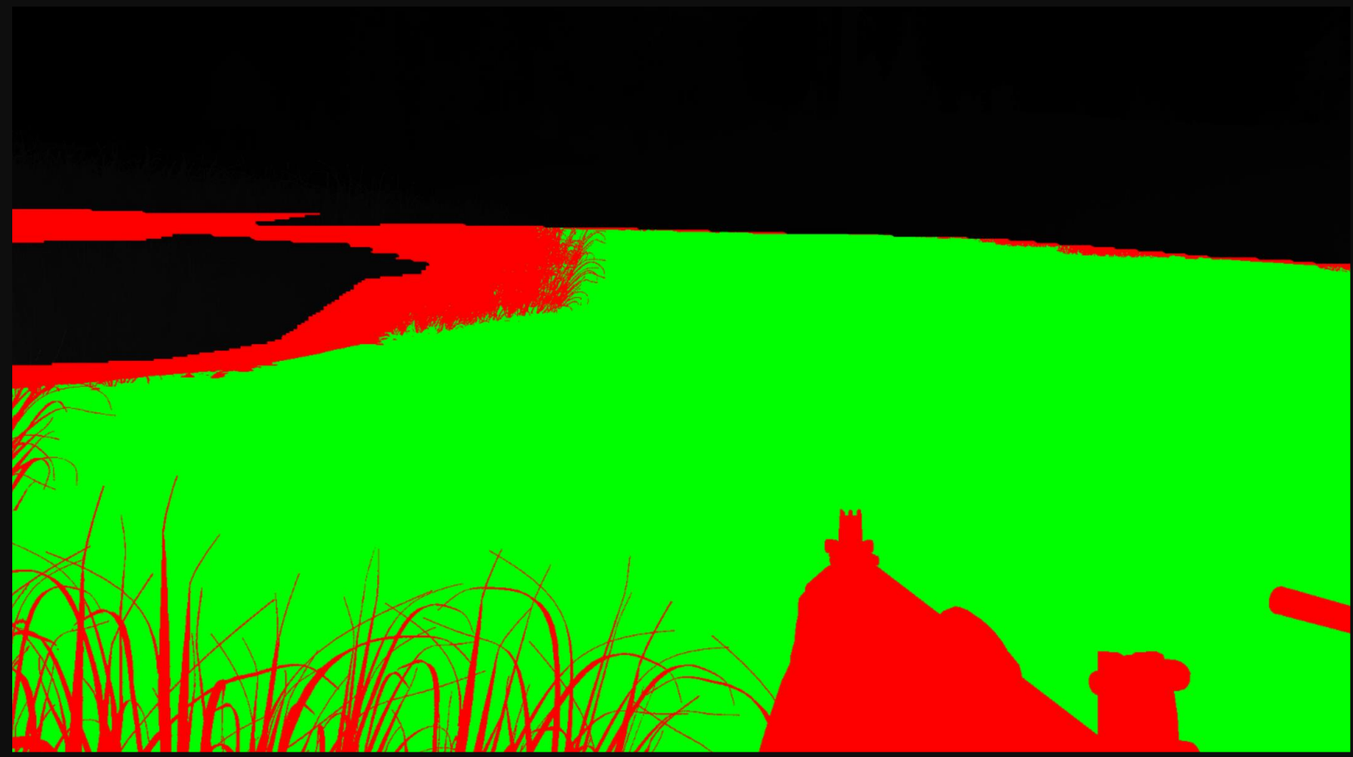
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)



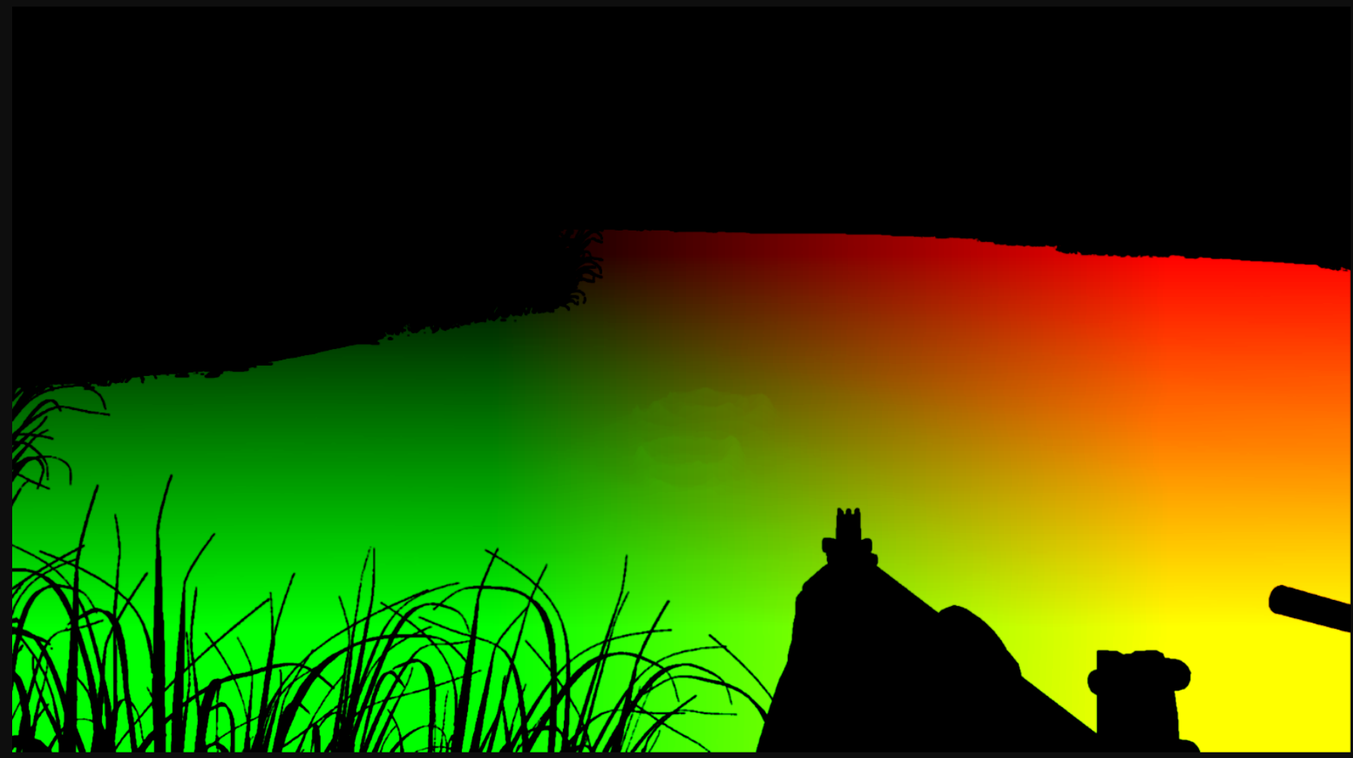
Tessellation

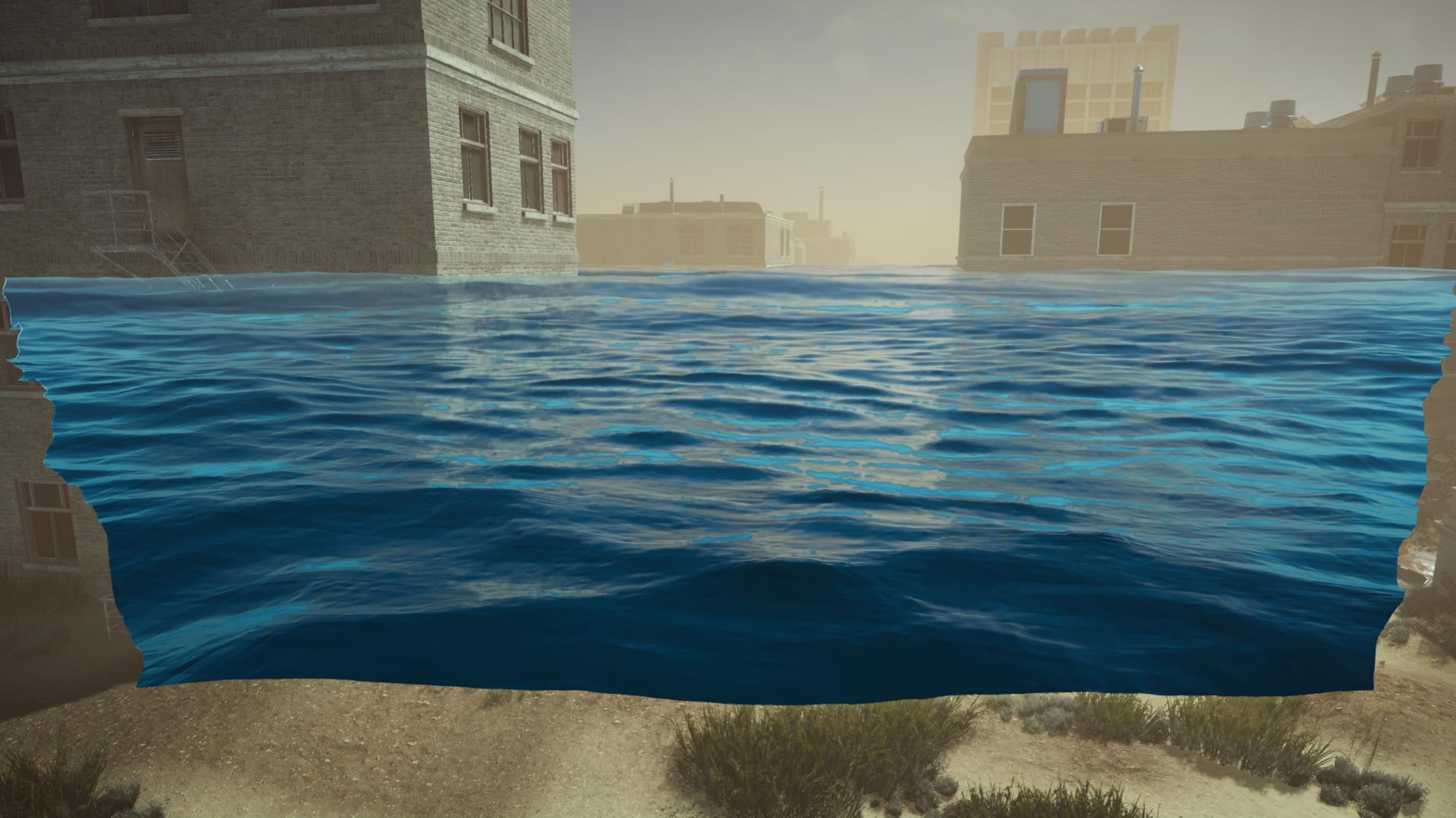
- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)

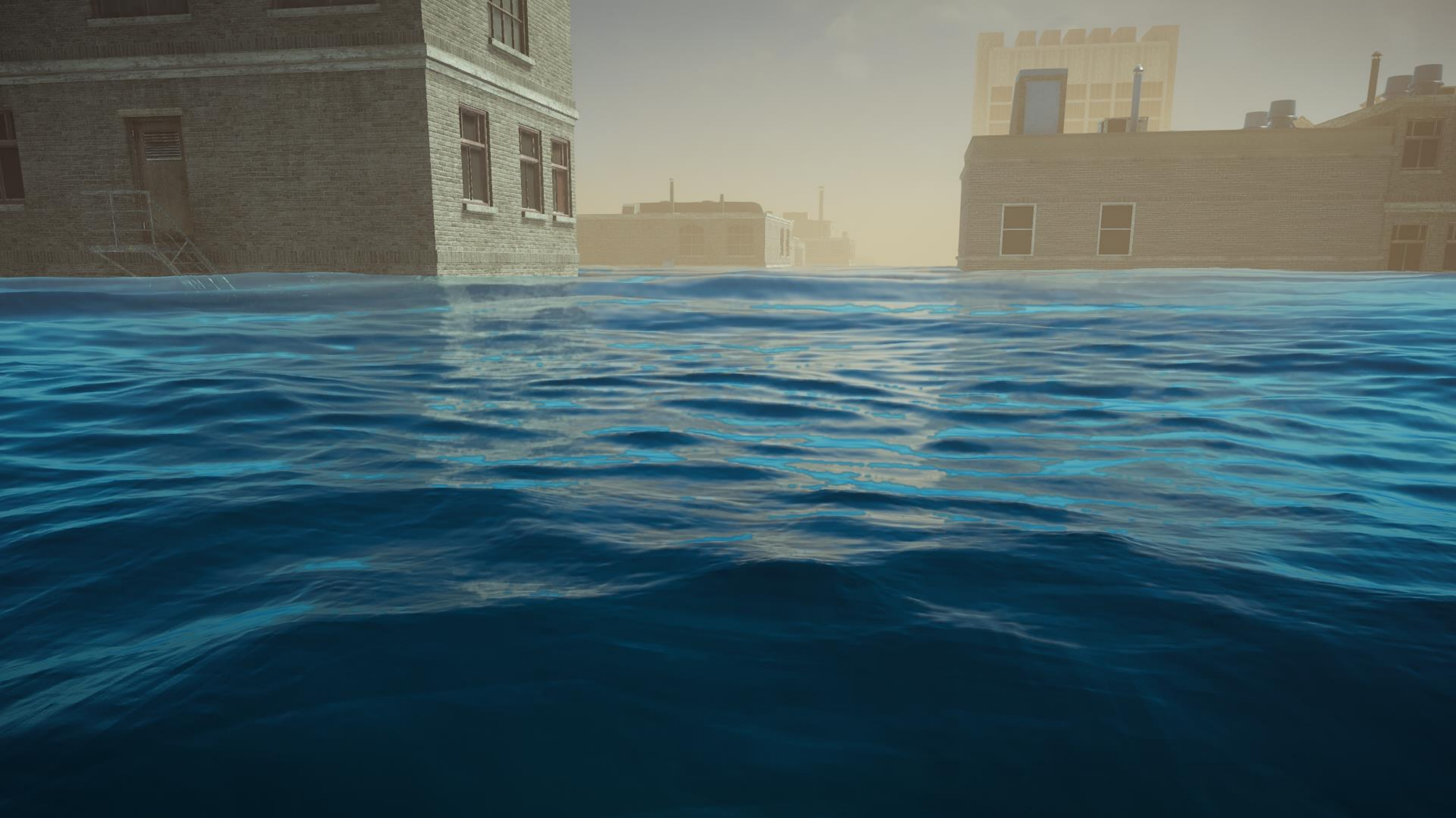


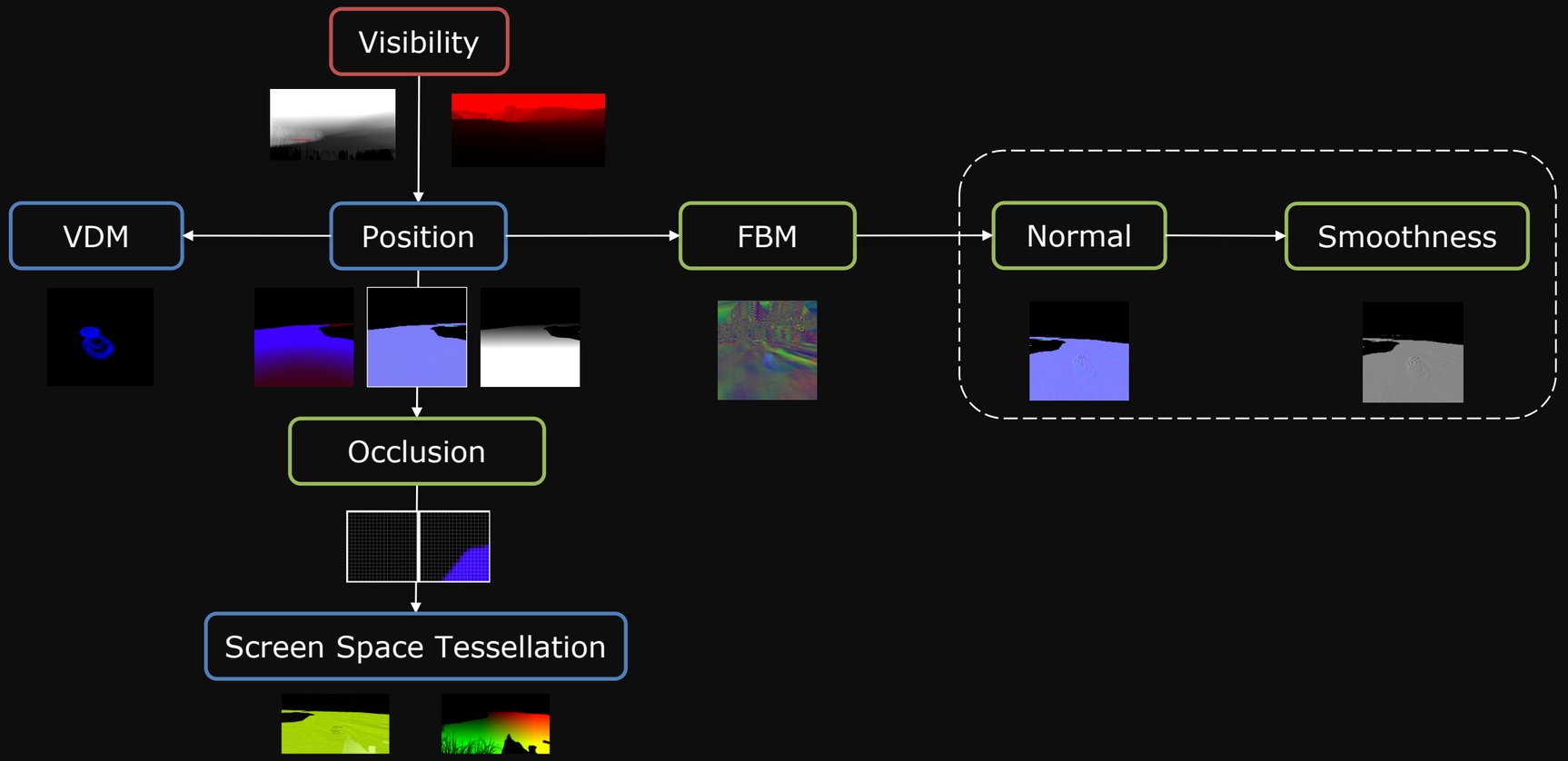
Tessellation

- For each vertex
  - Sample depth, compute position (> fov)
  - Sample displacement (FBM + splash)
  - Clip invalid vertices with a NaN (/ 0)
  - Project into screen space and write uv’s
  - Depth test against scene buffer (== fov)





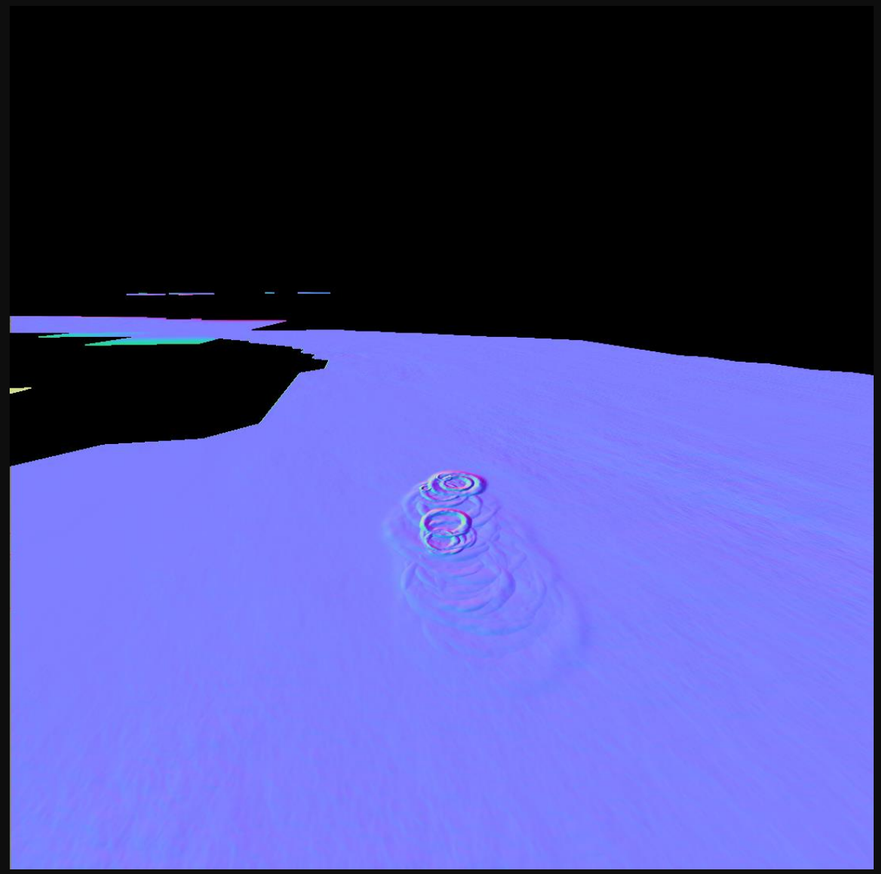



Visibility
VDM
Position
FBM
Normal
Smoothness
Occlusion
Screen Space Tessellation

PS CS PS+CS

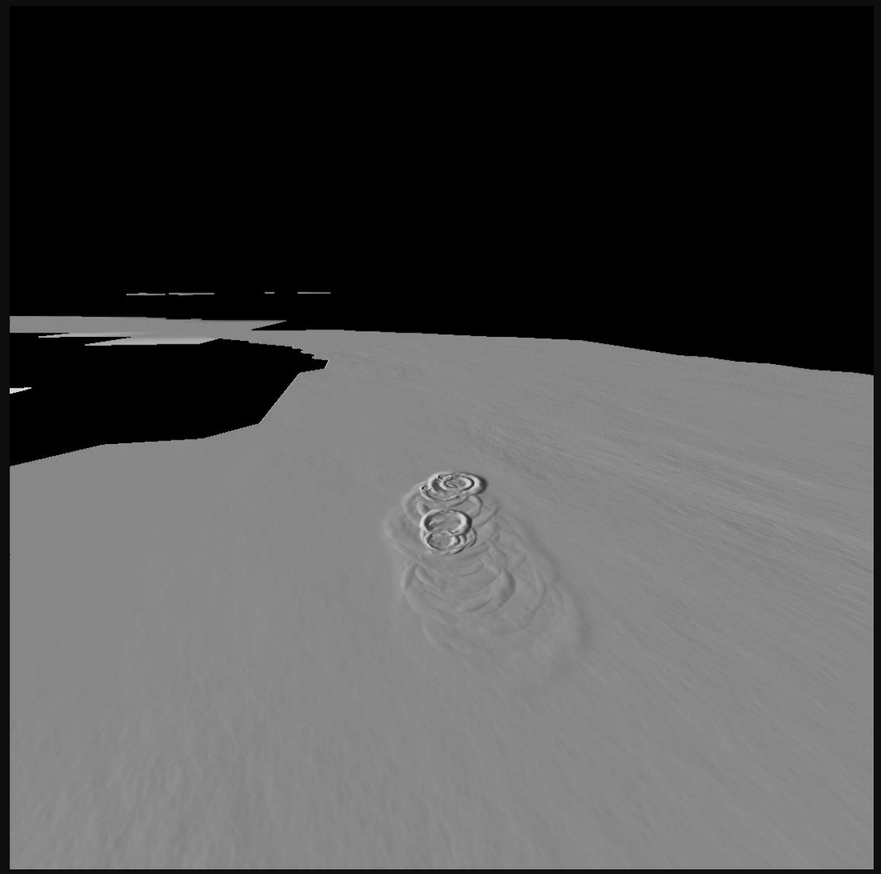
Normal

- Generate screen space normal map
  - Handles mesh, splash displacement and fbm
  - 4 Position from depth samples, cross product
  - Increase sampling distance based on distance to water
  - Blend with mesh normal to remove discontinuities



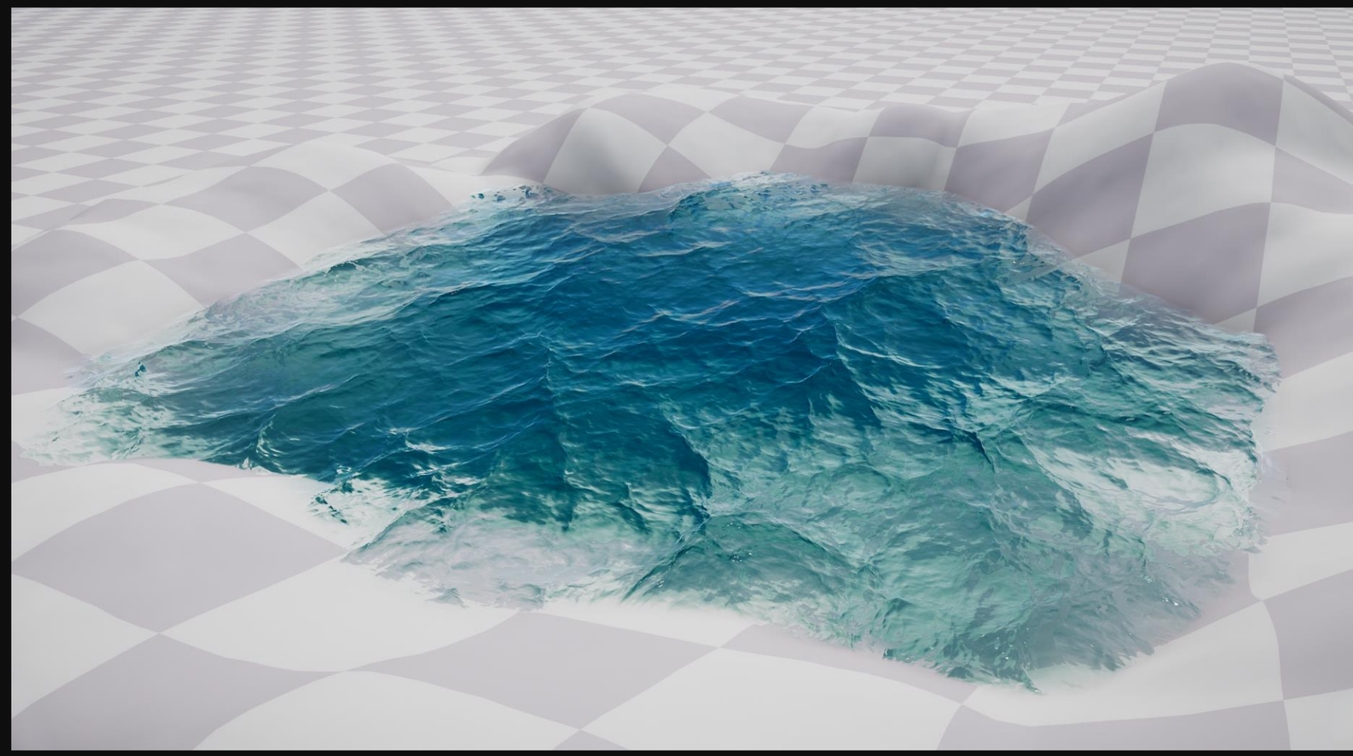
Smoothness

- Generate screen space smoothness
  - Very important for filtering lighting
  - Variance based
  - Compute Gaussian normal
  - Solve for smoothness



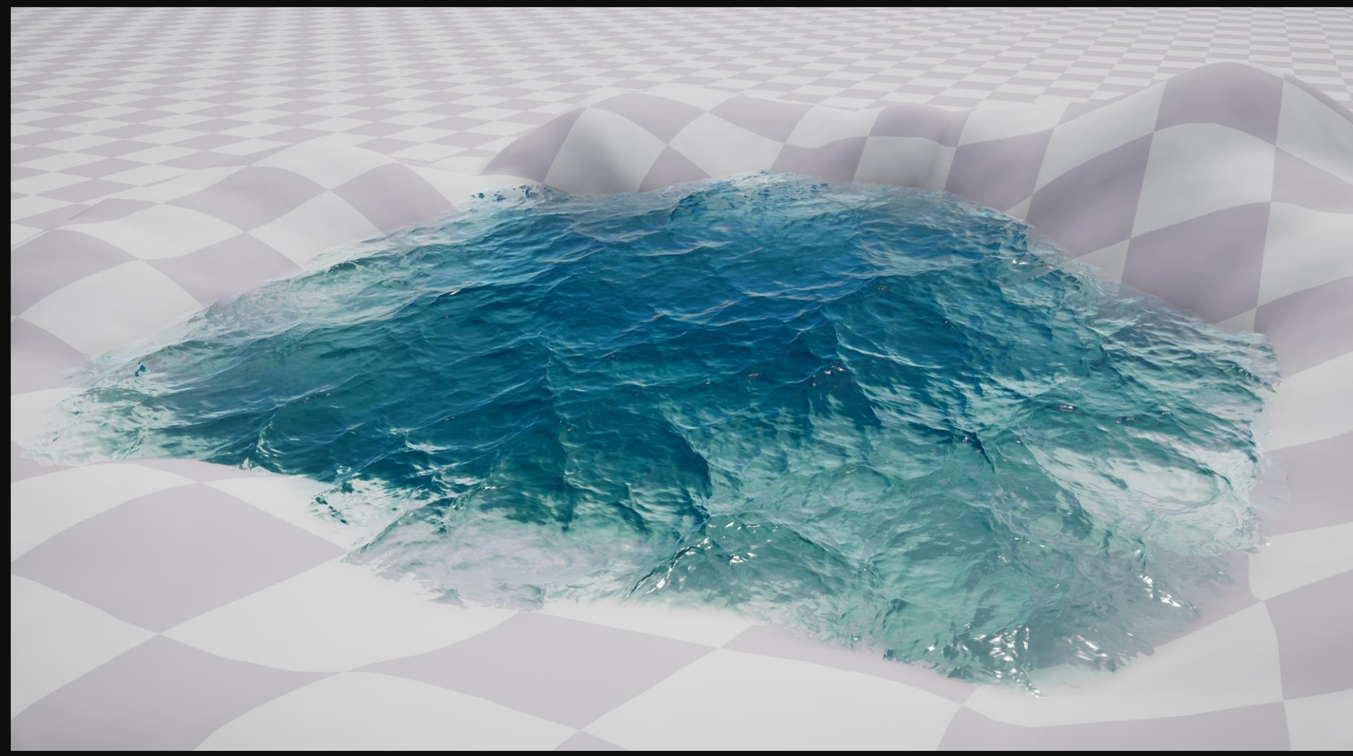
Smoothness

- Generate screen space smoothness
  - Very important for filtering lighting
  - Variance based
  - Compute Gaussian normal
  - Solve for smoothness



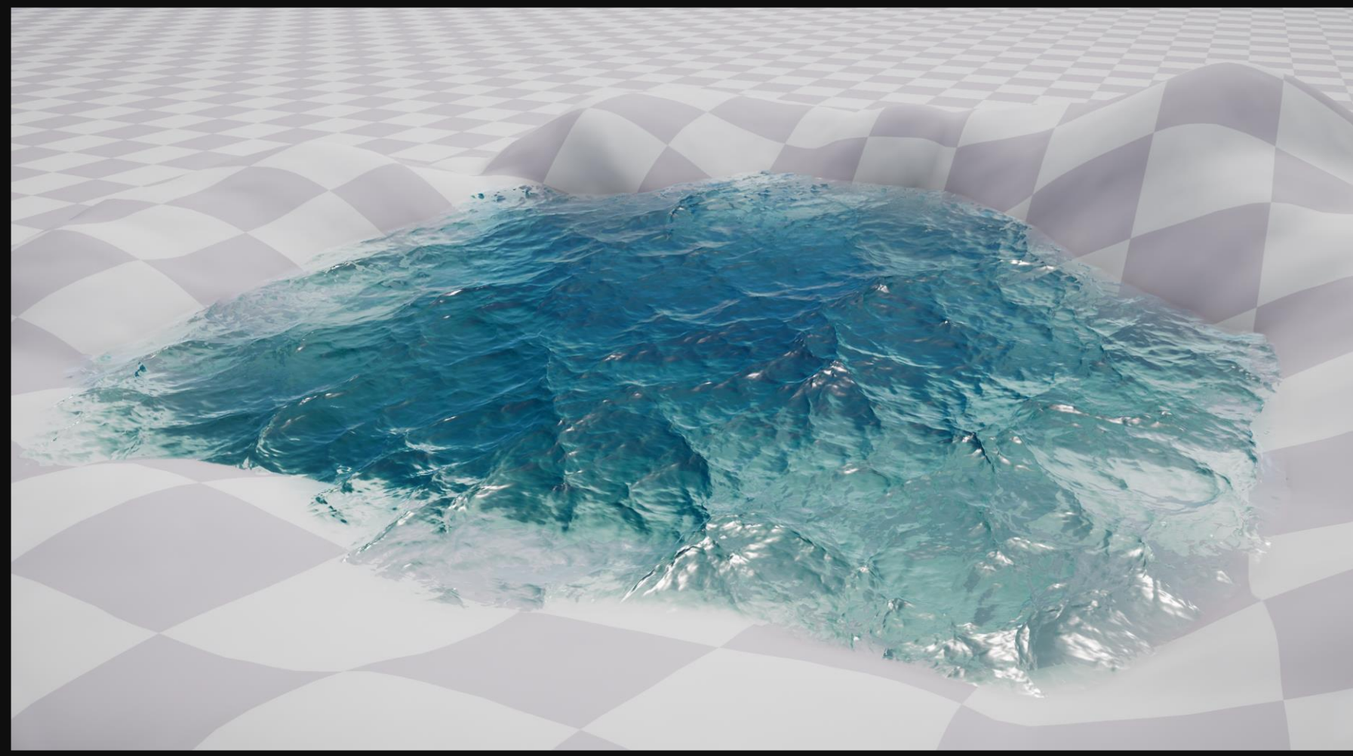
Smoothness

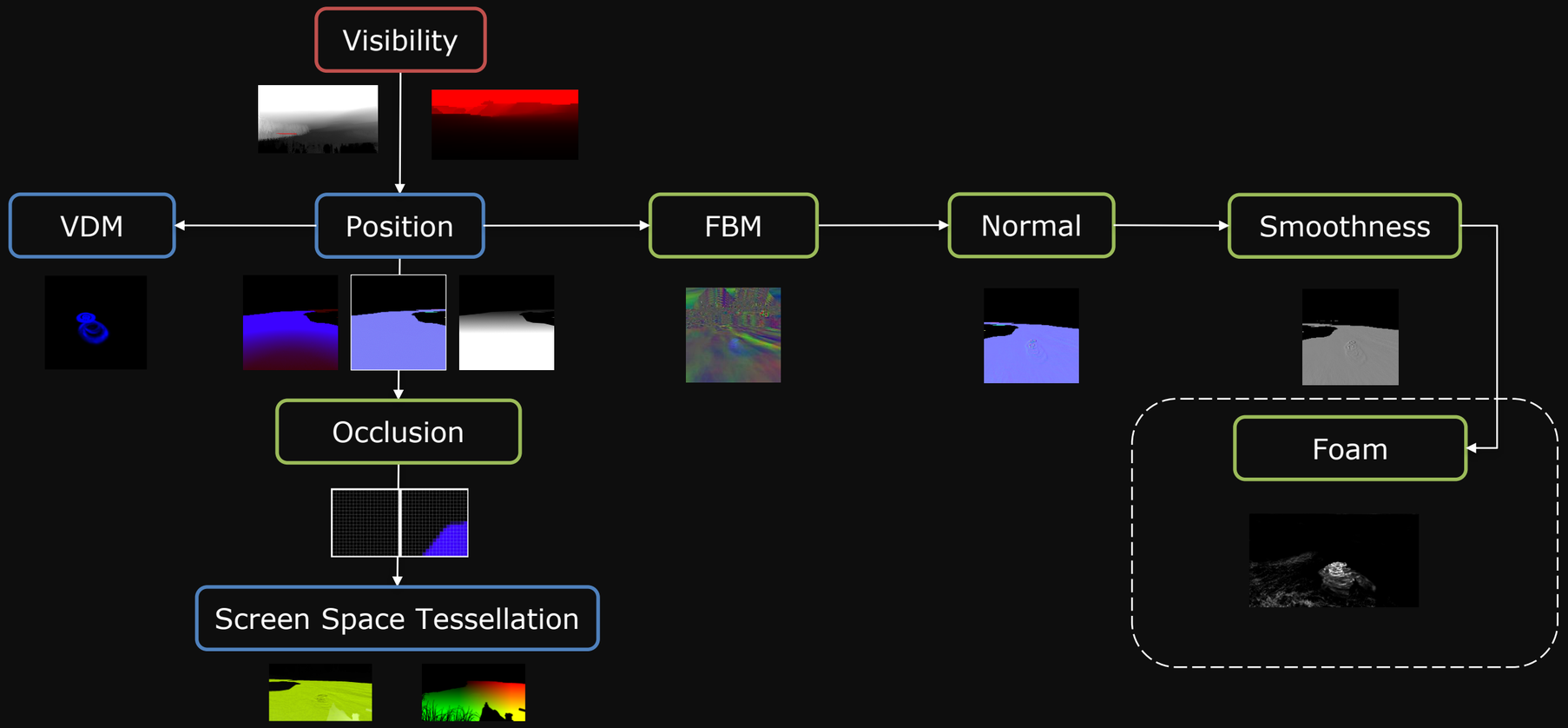
- Generate screen space smoothness
  - Very important for filtering lighting
  - Variance based
  - Compute Gaussian normal
  - Solve for smoothness



Smoothness

- Generate screen space smoothness
  - Very important for filtering lighting
  - Variance based
  - Compute Gaussian normal
  - Solve for smoothness

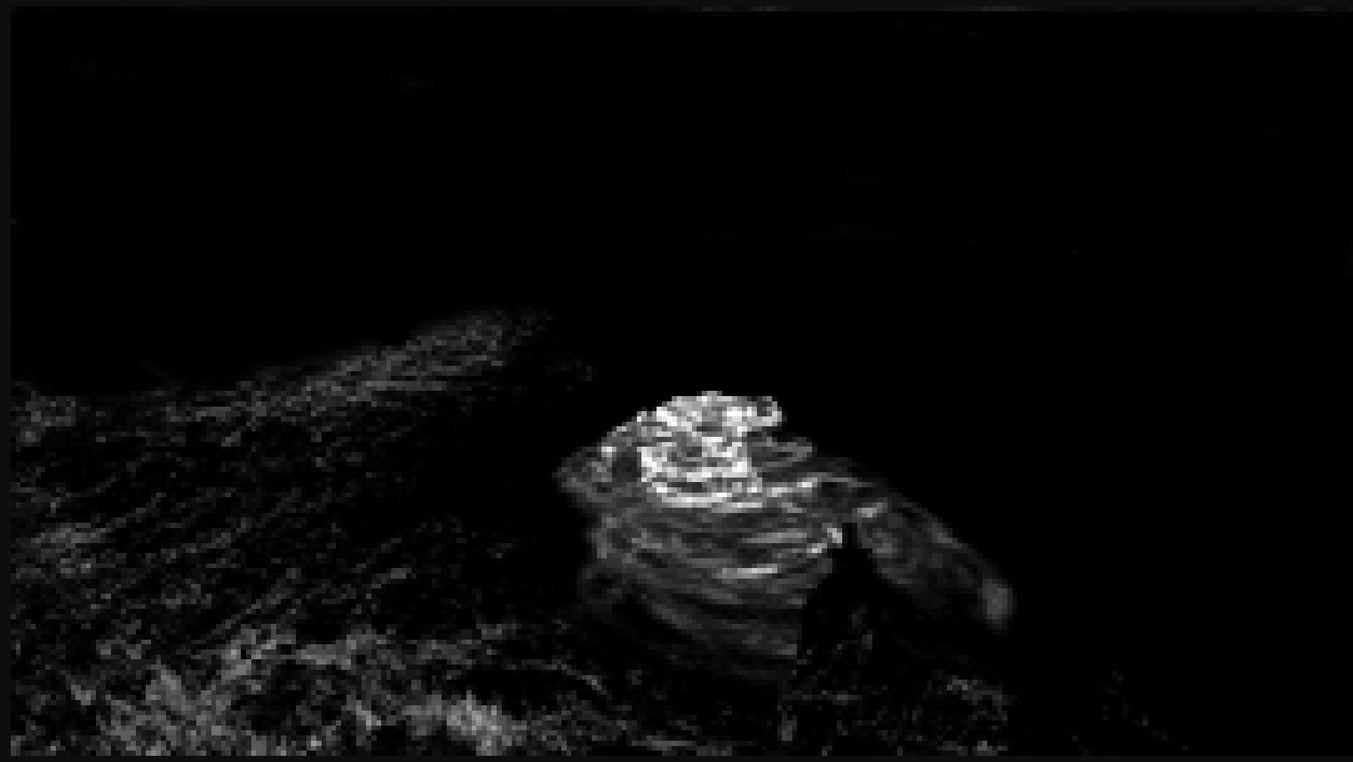



Visibility
VDM
Position
FBM
Normal
Smoothness
Occlusion
Foam
Screen Space Tessellation

PS CS PS+CS

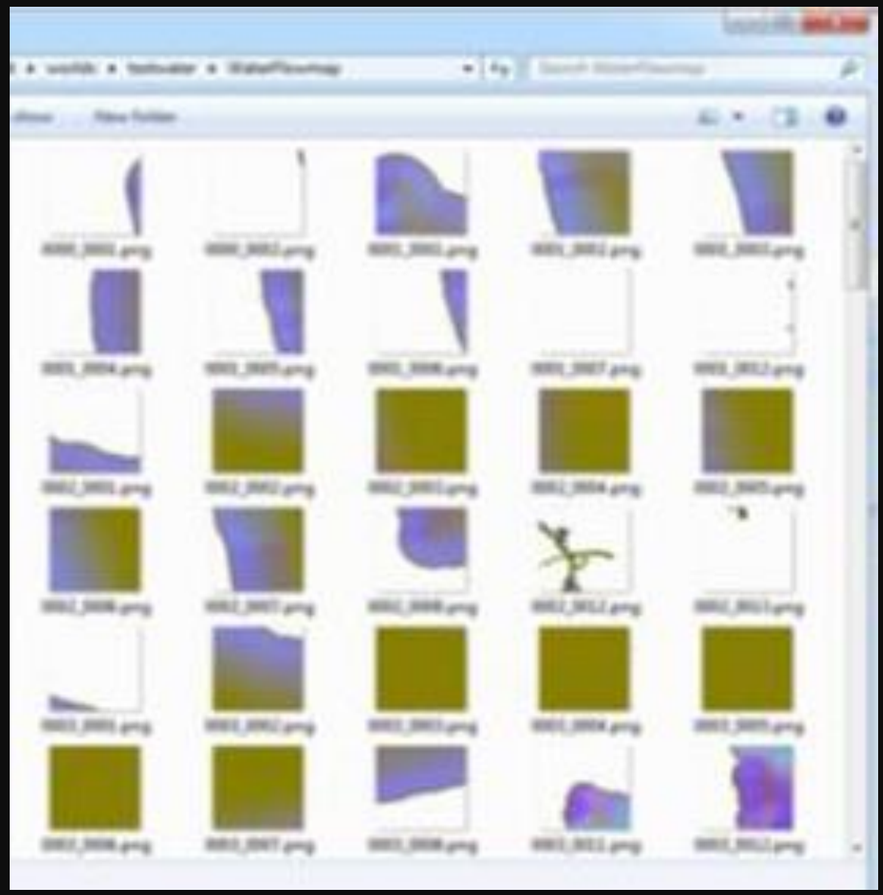
Foam

- Noise Texture
  - Foam color modulated by a noise texture
- Flow Map
  - Sampled using two offset phases
- SDF controls where foam appears
  - Rocks & Shorelines
- Blends displacement foam
  - Max blend



Flow Map

- Auto Generated
  - Based on terrain and water level



Flow Map

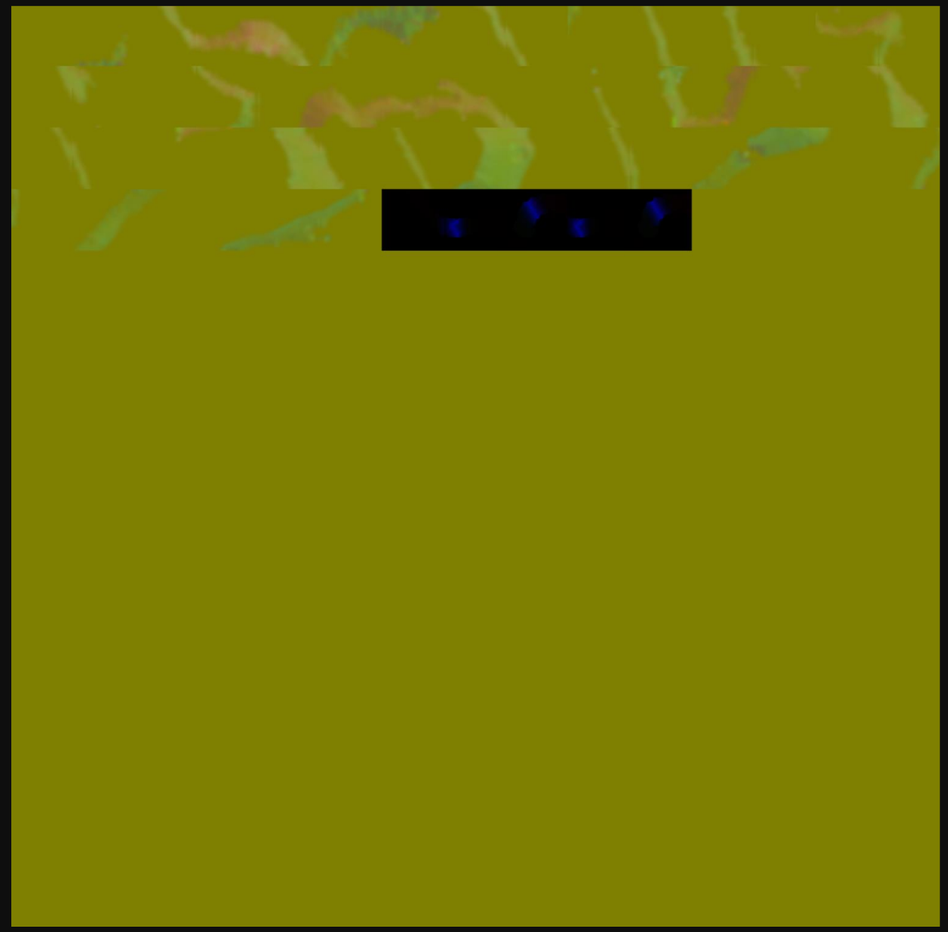
- Auto Generated
  - Based on terrain and water level

Flow Map

- Auto Generated
  - Based on terrain and water level
- Spline and Flood Fill Based
  - SDF guides flood fill algorithm

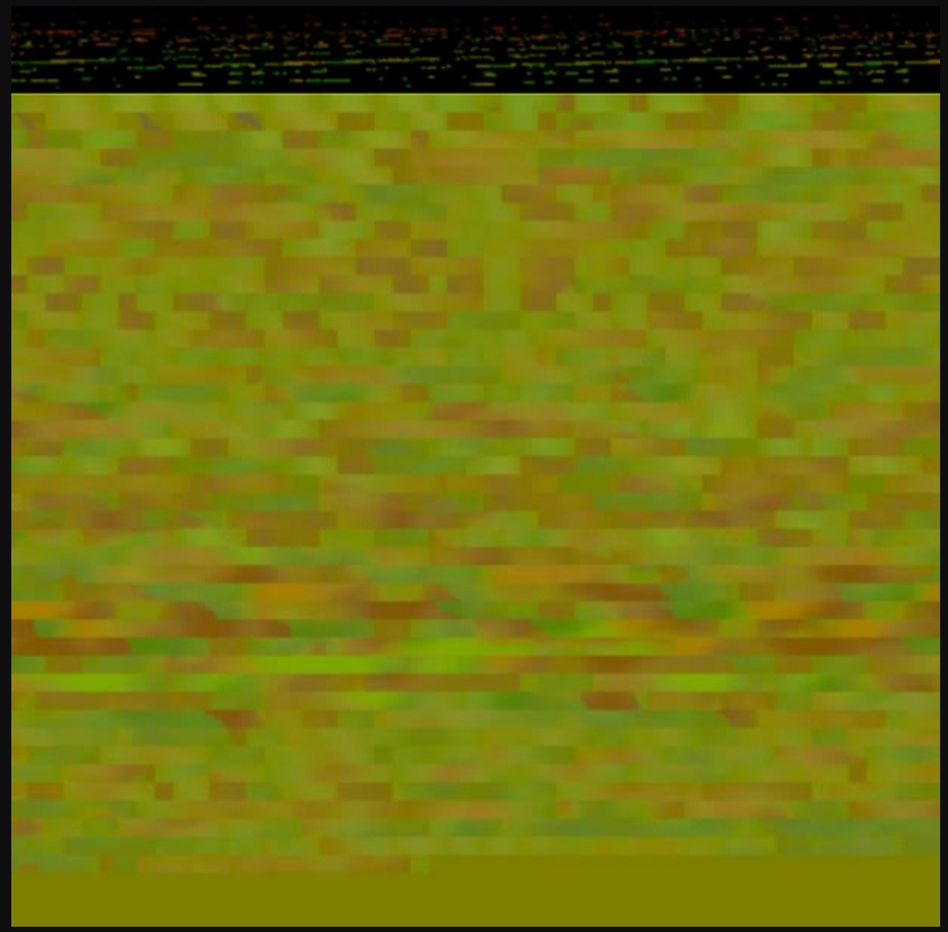
Flow Map

- Auto Generated
  - Based on terrain and water level
- Spline and Flood Fill Based
  - SDF guides flood fill algorithm
- Creates a flow map texture atlas
  - High resolution close to player
  - World flow map (vista + world)



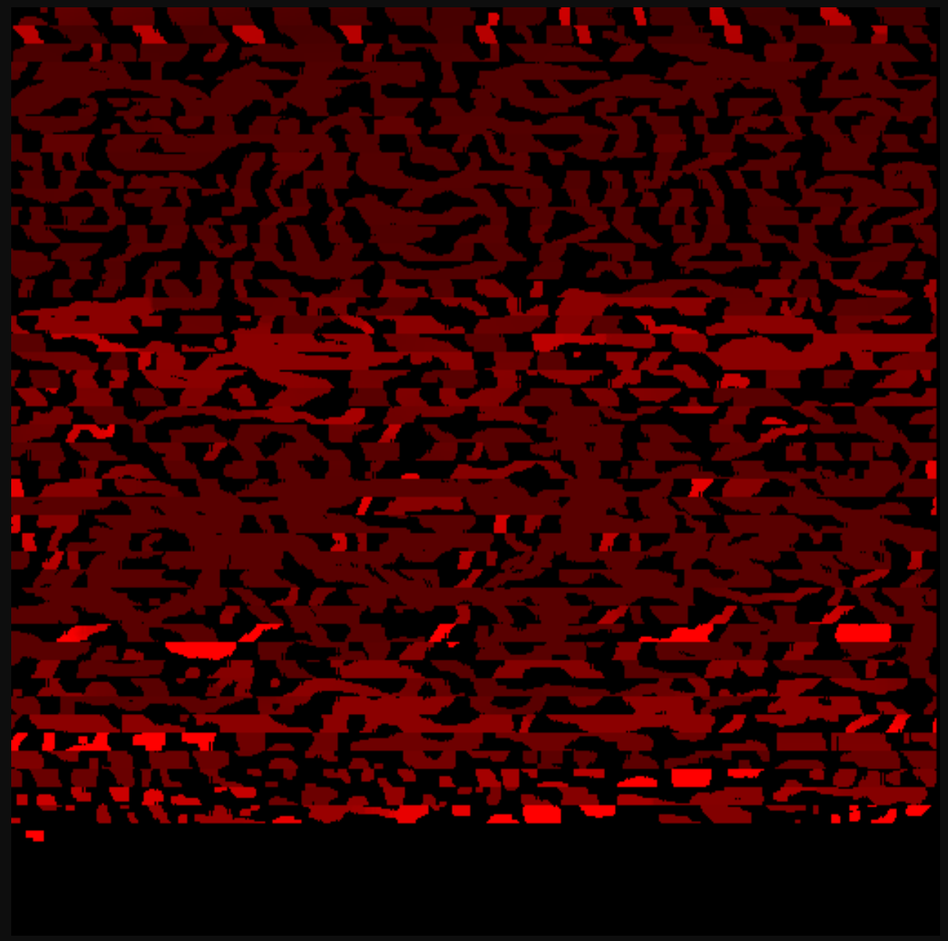
Flow Map

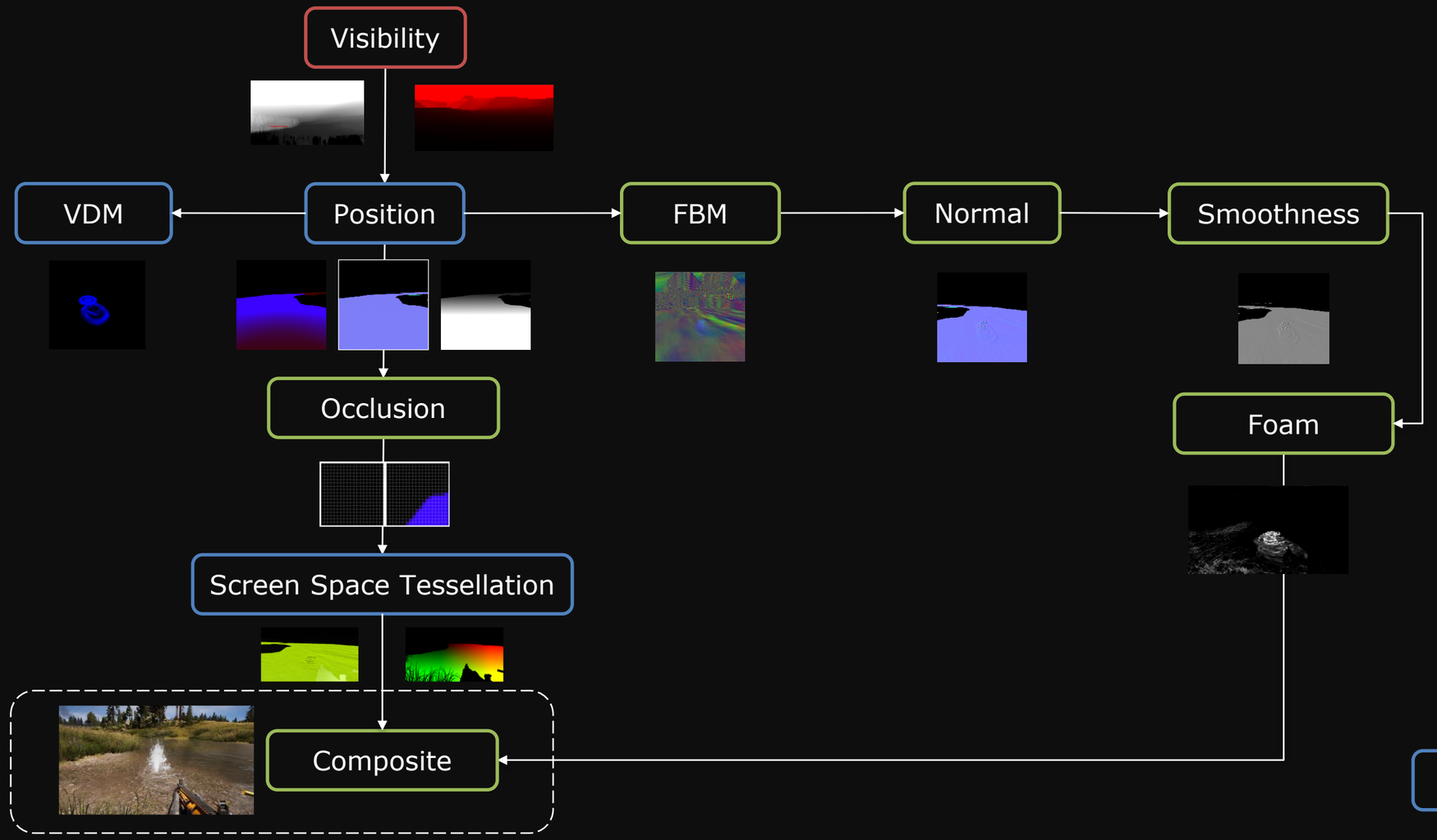
- Auto Generated
  - Based on terrain and water level
- Spline and Flood Fill Based
  - SDF guides flood fill algorithm
- Creates a flow map texture atlas
  - High resolution close to player
  - World flow map (vista + world)



Flow Map

- Auto Generated
  - Based on terrain and water level
- Spline and Flood Fill Based
  - SDF guides flood fill algorithm
- Creates a flow map texture atlas
  - High resolution close to player
  - World flow map (vista + world)
- World Height Map
  - 8 meter per pixel




Visibility
VDM
Position
FBM
Normal
Smoothness
Occlusion
Foam
Screen Space Tessellation
Composite
PS
CS
PS+CS

Composite

- Lights the water surface
  - Sample material buffer
  - Depth w + wo water
- Tiled z-binned lighting
  - Indirect (Ambient (GI) + Reflection (EnvMap + SSLR))
  - Directional (Sun)
  - PointAndSpotLighting
  - ExposureLighting

```
float2 vpos        = dispatchThreadId.xy;
uint materialID    = GetMaterialID(data);
uint mtlBlendId    = GetBlendMaterialIdx(data);
uint shaderID      = GetShaderID(data);
bool isFrontFace   = GetIsFrontFaceID(data);
float smoothness   = GetSmoothness(data);
```

```
WaterMaterialData materialData = InterpolateMaterialData(WaterMaterialBuffer[materialID],
                                                         WaterMaterialBuffer[mtlBlendId],
                                                         mtlBlendFact);
```

Composite

- Lights the water surface
  - Sample material buffer
  - Depth w + wo water
- Tiled z-binned lighting
  - Indirect (Ambient (GI) + Reflection (EnvMap + SSLR))
  - Directional (Sun)
  - PointAndSpotLighting
  - ExposureLighting

```
// Read Data From Material Structured Buffers
float4 baseColor            = materialData.baseColor;
float2 baseTiling           = materialData.baseTiling;
float4 lightBeamAttenuation = materialData.lightBeamAttenuation;
float  flowmapSpeedScale    = materialData.flowmapPhase.x;
float  enableFlowMap        = materialData.flowmapPhase.y;
float4 lightIrradianceRatio = materialData.lightIrradianceRatio;
float4 foamParameters       = materialData.foamParameter;
float2 waterDistortion      = materialData.waterDistortion;
float  underWaterDepthScale = materialData.lightIrradianceRatio.w;
float4 causticsData         = materialData.caustics;
float sunShadowScale        = lerp(materialData.textureIndex.z, 1.0f, isFrontFace);
float  useLightTransport    = 1.0f - step(lightBeamAttenuation.w, 0);
float cameraToPlaneDistance = length(positionWS - CameraPosition);
```

Composite

- Lights the water surface
  - Sample material buffer
  - Depth w + wo water
- Tiled z-binned lighting
  - Indirect (Ambient (GI) + Reflection (EnvMap + SSLR))
  - Directional (Sun)
  - PointAndSpotLighting
  - ExposureLighting

```
LightingOutput lightingOutput = (LightingOutput)0;
ComputeIndirectLighting(lightingInput, lightingOutput, waterMaterialProperties.smoothness);
float3 ambientLighting = lightingOutput.diffuse;
ComputeDirectionalLighting(lightingInput, lightingOutput);
ComputePointAndSpotLighting(lightingInput, lightingOutput, lightTileIndex, minLightIndex, maxLightIndex);
ComputeExposureLighting(lightingInput, lightingOutput, lightTileIndex, minLightIndex, maxLightIndex);
```

Composite

- Light Transport for Surface Color
- Foam, refraction and caustics
- **VGPR heavy pass**

```
name="ScatteringCoefficients0"  type="float4" defaultvalue="36,   4.3,  1.8,  0.037"
name="ScatteringCoefficients1"  type="float4" defaultvalue="40,   7.6,  6.2,  0.098"
name="ScatteringCoefficients2"  type="float4" defaultvalue="44.5, 11.6, 11.6, 0.158"
name="ScatteringCoefficients3"  type="float4" defaultvalue="45,   13,   17,   0.219"
name="ScatteringCoefficients4"  type="float4" defaultvalue="45.5, 16.5, 23,   0.420"
name="ScatteringCoefficients5"  type="float4" defaultvalue="46,   20,   29,   0.620"
name="ScatteringCoefficients6"  type="float4" defaultvalue="50,   25.5, 36,   0.820"
name="ScatteringCoefficients7"  type="float4" defaultvalue="52,   28.25,39.5, 1.021"
name="ScatteringCoefficients8"  type="float4" defaultvalue="54,   31,   43,   1.222"
name="ScatteringCoefficients9"  type="float4" defaultvalue="63,   49,   71,   1.422"
name="ScatteringCoefficients10" type="float4" defaultvalue="69.5, 63.5, 97,   1.623"
name="ScatteringCoefficients11" type="float4" defaultvalue="76,   78,   123,  1.824"
```

Composite

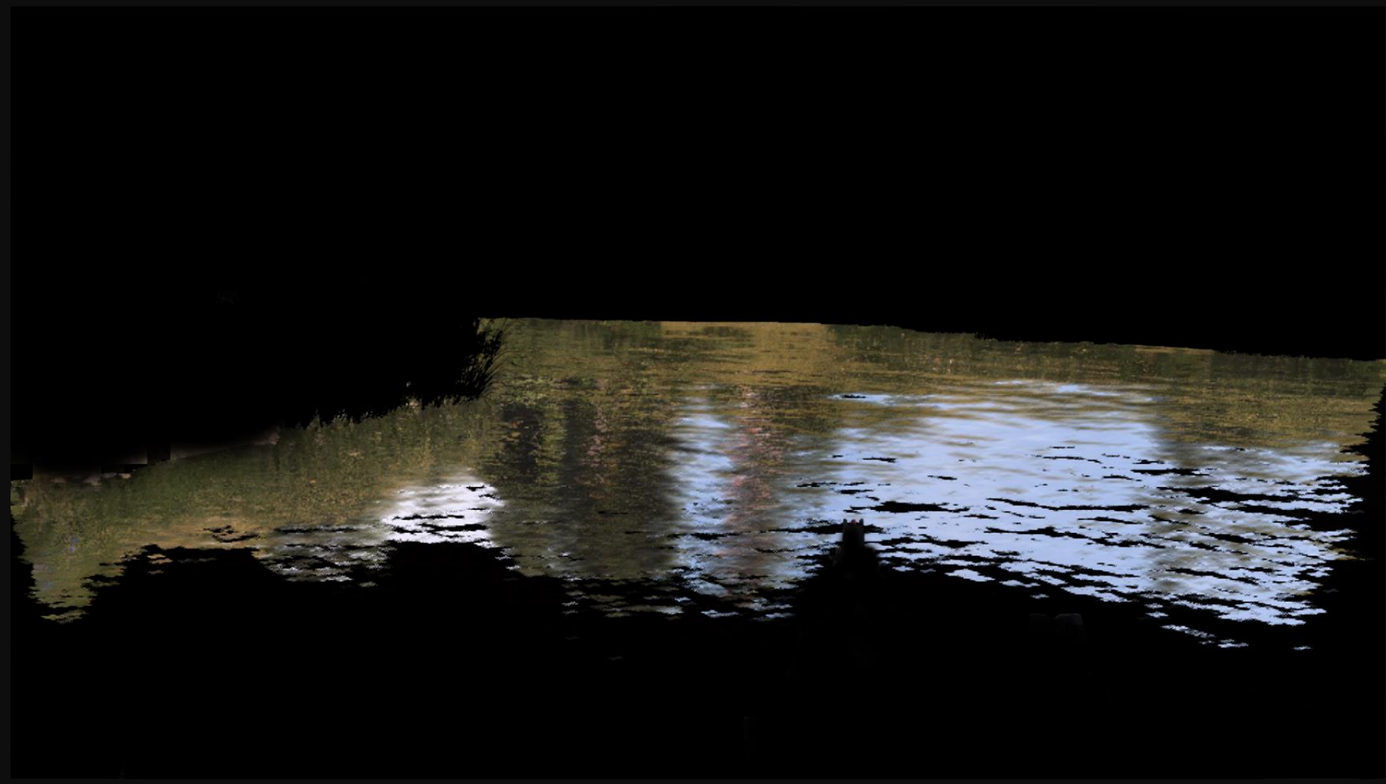
- Light Transport for Surface Color
- Foam, refraction and caustics
- **VGPR heavy pass**

Composite

- Light Transport for Surface Color
- Foam, refraction and caustics
- **VGPR heavy pass**

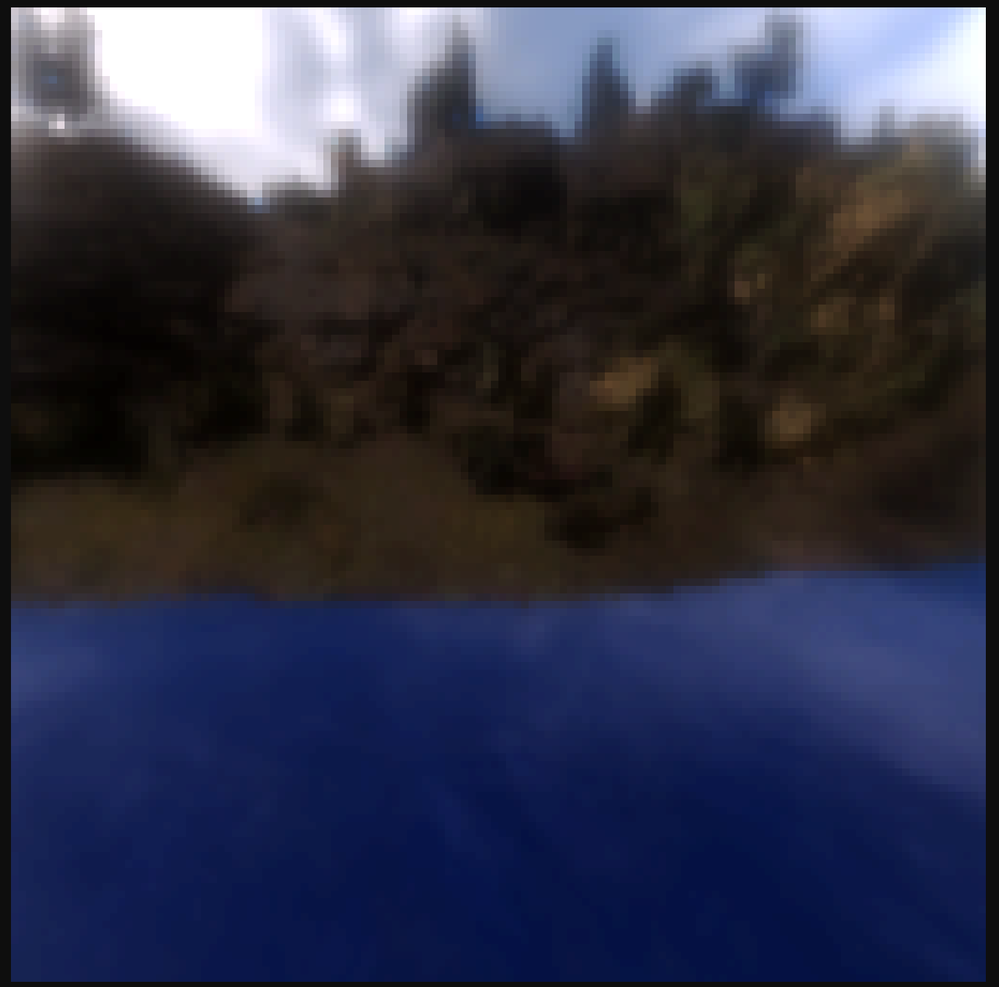
Composite

- Light Transport for Surface Color
- Foam, refraction and caustics
- **VGPR heavy pass**



Composite

- Light Transport for Surface Color
- Foam, refraction and caustics
- **VGPR heavy pass**



Material blending in half precision

- Water surface composite pass uses 9 VGPRs less after below optimization
- Single simple change, in yellow below

```
struct WaterMaterialData //HLSL declaration of water material structured buffer
{
    min16float2 baseTiling;
    min16float2 waterDistortion;
    min16float4 baseColor;
    min16float4 caustics;
    min16float4 lightBeamAttenuation;
    min16float4 fbmData;
    min16float2 fbmData2;

    //more parameters were actually converted for this buffer, but omitted here to save space
};

StructuredBuffer<WaterMaterialData> materialDataBuffer;
```

Minimum precision basics

- '**min16float**' is a HLSL basic type
  - Let the compilers know precision can be lowered
  - The actual precision stored in the buffer is still full
    - GPU is free to convert it down to 16-bit when sampling
  - Does not actually force precision to be low
  - GPU needs to support lower precision
- The 'half' HLSL basic type is full precision
- Counterparts exist for GLSL

Precision lowering cannot be automatic

- Full precision is usually required for:
  - Texture coordinates for sizes of 512 texels or higher
  - Normal vectors in Cartesian coordinates
  - Any other math that causes major loss of significance
    - Subtracting two nearly equal numbers
    - Dividing to a number close to 0
    - Iterative math that can accumulate into a large error

Register pressure is a common bottleneck

- Low register usage allows shaders to run more threads concurrently, in order to counter memory latency
  - Occupancy increases at discrete thresholds
  - Narrowly missing a threshold should be avoided
  - Optimizing lower occupancy shaders yields higher gains

Main source of register pressure

- Maximum number of simultaneously ‘live’ registers
  - Memory reads are performed early and cached
  - Loop unrolling
  - Large number of intermediary values

Register allocation overhead

- Caused by memory operation requirements
- High number of channels
  - Channels need to be in **consecutive** registers
  - Channels need to be **ordered properly** in the registers
  - E.g. `buffer_load_format_xyzw v[1:4], v0, s[4:7], 0`
- High number of texture dimensions
  - Also need to be in **consecutive** registers and **ordered properly**
  - 3D textures, texture arrays, cube maps, LOD index

Live register analysis

- A register is ‘live’ at a given location when the value it holds will be needed at a later execution time
- Live register analysis is needed to measure allocation overhead
- **Radeon™ GPU Analyzer** output will be shown next
  - Two different examples will be compared
  - The first example has no allocation overhead
  - Second example has slightly modified HLSL causing overhead
    - The 1 VGPR of overhead can further cause allocation fragmentation
    - The math cannot be done in place without additional ALU

```
1 |    5 | :^^^^ | buffer_load_format_xyzw v[1:4], v0, s[4:7], 0
2 |    5 | ::::: | s_waitcnt vmcnt(0)
3 |    5 | :xv:: | v_add_f32 v1, v1, v2
4 |    5 | ::^v: | v_add_f32 v2, v3, v1
5 |    5 | :::^v | v_add_f32 v3, v4, v2
6 |    4 | vvvv  | buffer_store_dwordx3 v[1:3], v0, s[8:11], 0
7 |    0 |       | s_endpgm

Maximum # VGPR used   5, # VGPR allocated:   5
```

- No allocation overhead
- 3 ALU operations
- The math is done in place

```
Buffer<float4> InputBuffer;
RWStructuredBuffer<float3> OutputBuffer;

[numthreads(64, 1, 1)]
void ComputeShaderFunc(uint threadId : SV_GroupThreadID)
{
	float4 XYZW = InputBuffer[threadId.x];
	OutputBuffer[threadId.x] = float3(  X+Y,  X+Y+Z,  X+Y+Z+W  );
}
```

```
1 |    5 | :^^^^   | buffer_load_format_xyzw v[1:4], v0, s[4:7], 0
2 |    5 | :::::   | s_waitcnt vmcnt(0)
3 |    5 | :::xv   | v_add_f32 v3, v3, v4
4 |    5 | ::v:^   | v_add_f32 v4, v2, v3
5 |    5 | :v ::^  | v_add_f32 v5, v1, v4
6 |    4 | v  vvv  | buffer_store_dwordx3 v[3:5], v0, s[8:11], 0
7 |    0 |         | s_endpgm

Maximum # VGPR used   5, # VGPR allocated:   6
```

- 1 VGPR allocation overhead
- 3 ALU operations
- Allocation fragmentation

```
Buffer<float4> InputBuffer;
RWStructuredBuffer<float3> OutputBuffer;

[numthreads(64, 1, 1)]
void ComputeShaderFunc(uint threadId : SV_GroupThreadID)
{
	float4 XYZW = InputBuffer[threadId.x];
	OutputBuffer[threadId.x] = float3(  W+Z,  W+Z+Y,  W+Z+Y+X  );
}
```

Half precision counters allocation overhead

- Half precision needs half the consecutive registers
  - min16float4 channels need just 2 registers
  - min16float4 can be more than twice better than float4
    - A lot less opportunity for allocation overhead

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

Problems

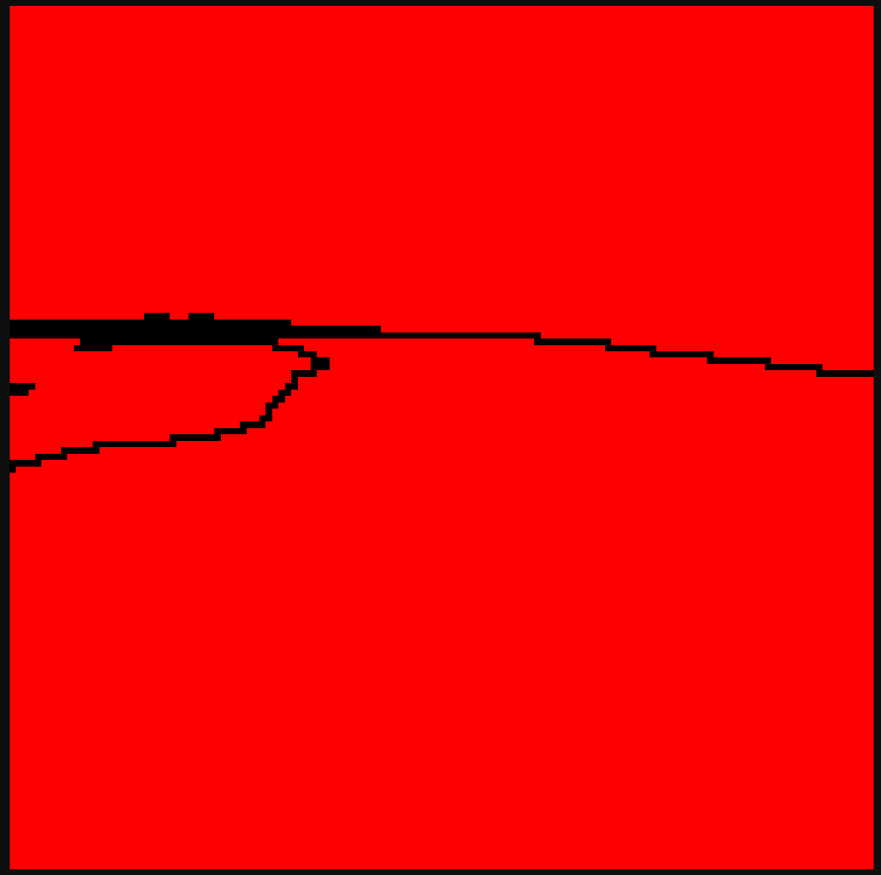
- Water writes depth
  - Bugs, bugs and more bugs
- Many Small Textures
  - Pack + Ping pong
- Screen Space Tessellation
  - VS Wave Launch Rate
  - Edge issues
- SSLR
- Render Order
  - Hard to move things around


BTM BOOST
Game paused!


BTM BOOST
Game paused!

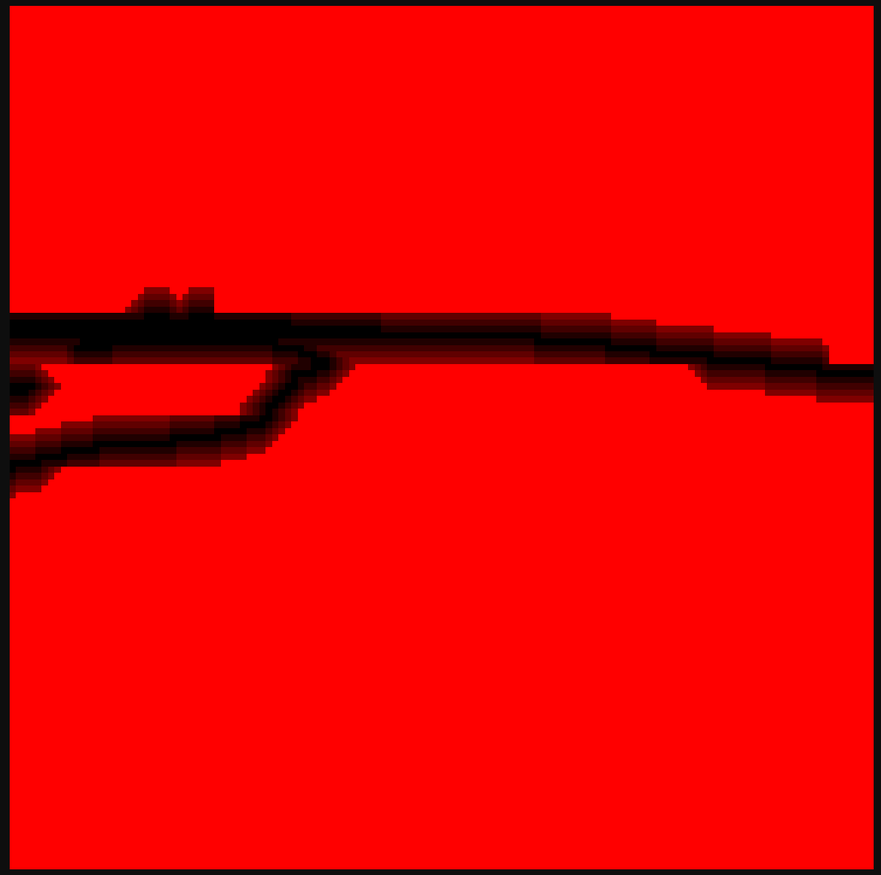
Dampen Edge

- Fix connecting water bodies
  - Dampen displacement between water bodies
  - Break connecting water bodies
- Edge Detect Pass
  - 8 samples surrounding each pixel
  - Large distance? Edge
- Down Sample Pass
  - 8x Down sample per pass using LDS
  - Each pass sample 4 points, write to single lds point
  - PC requires GroupMemoryBarrierWithGroupSync



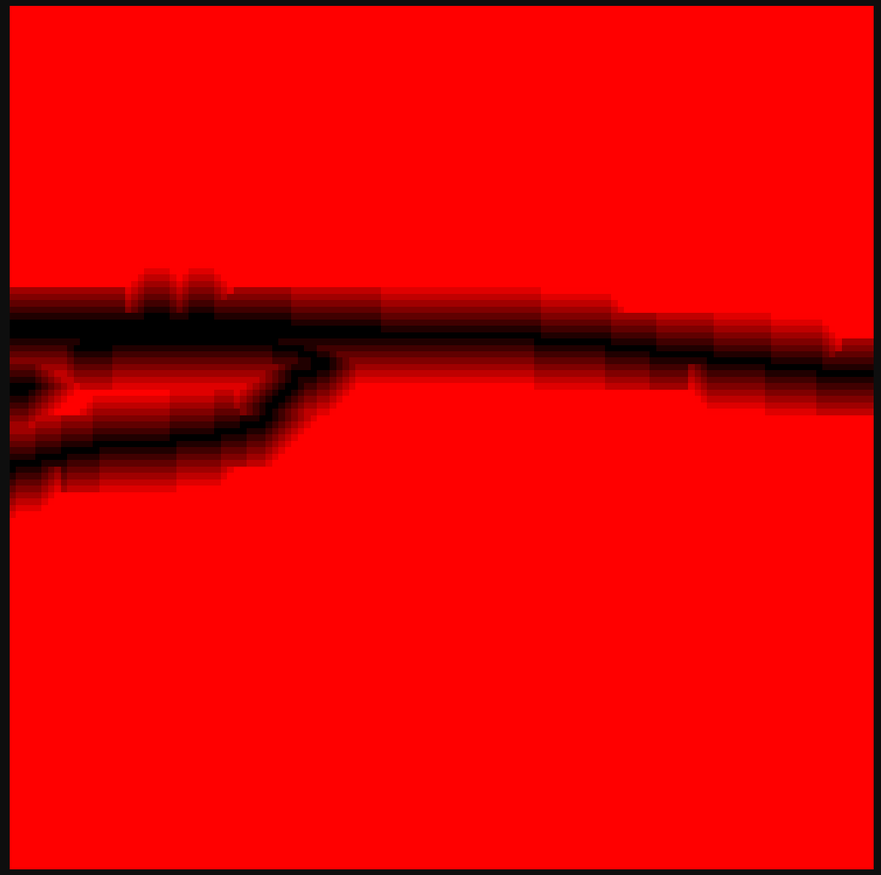
Dampen Edge

- Fix connecting water bodies
  - Dampen displacement between water bodies
  - Break connecting water bodies
- Edge Detect Pass
  - 8 samples surrounding each pixel
  - Large distance? Edge
- Down Sample Pass
  - 8x Down sample per pass using LDS
  - Each pass sample 4 points, write to single lds point
  - PC requires GroupMemoryBarrierWithGroupSync



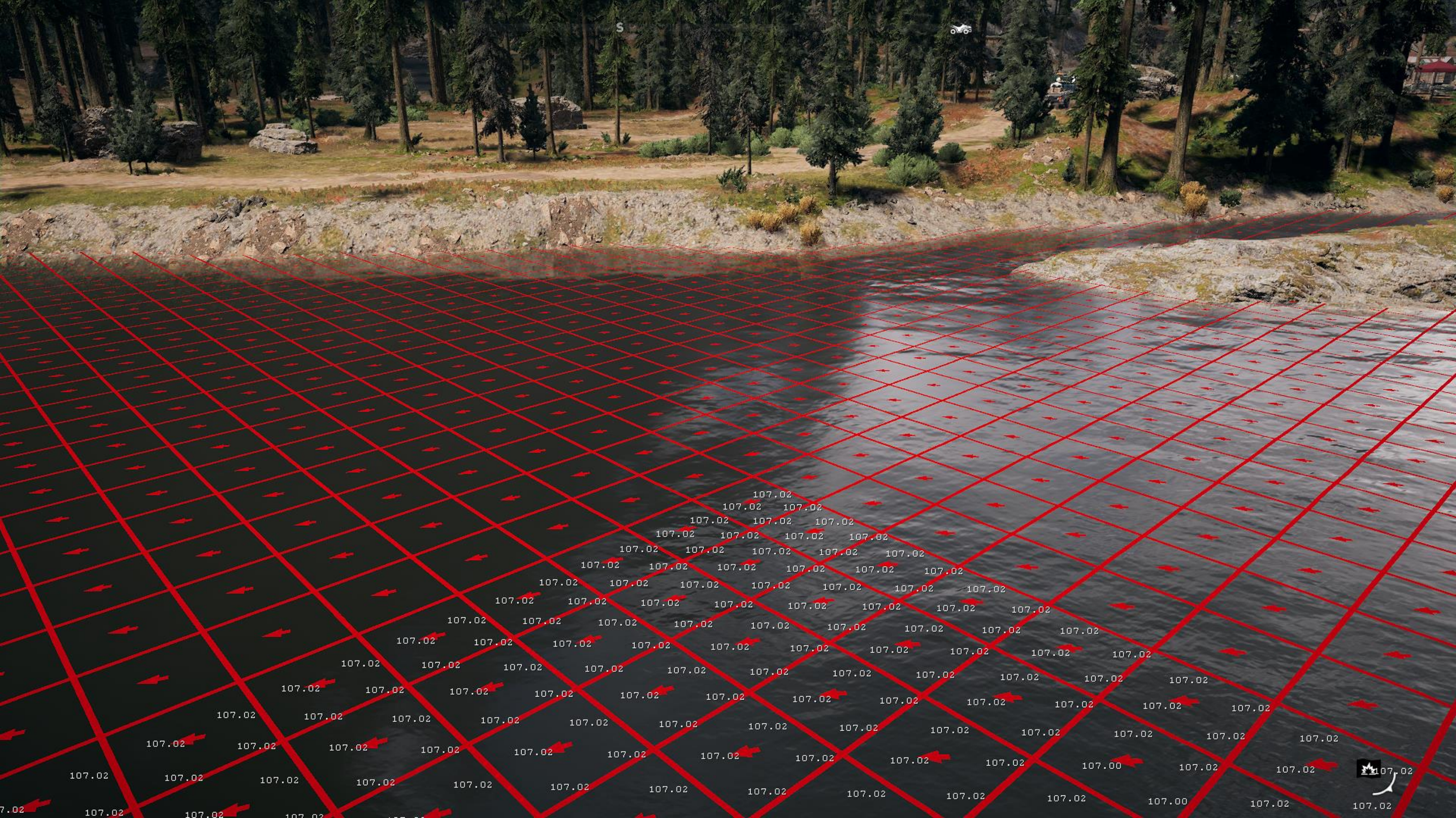
Dampen Edge

- Fix connecting water bodies
  - Dampen displacement between water bodies
  - Break connecting water bodies
- Edge Detect Pass
  - 8 samples surrounding each pixel
  - Large distance? Edge
- Down Sample Pass
  - 8x Down sample per pass using LDS
  - Each pass sample 4 points, write to single lds point
  - PC requires GroupMemoryBarrierWithGroupSync




Compiling shaders (1 remaining) -
ai_ignoreplayer is ON
OVERBUDGET: Anim 101.22% - (253.06MB/250.00MB)
S
Reel Line
Hook Fish


ai_ignoreplayer is ON
OVERBUDGET: Anim 100.43% - (251.07MB/250.00MB)
S
Reel Line
Hook Fish


S
107.02
107.00


Underwater Material:
Camera RiverSlow
Ghost Camera
Relax/Multi
Material List:
1:Water_RiverSlow
2:Water_RiverFast
Toggle Tandem Mode
5:Water_Wetlands
7:Water_Waterfalls_C3_sec_5975
N

Performance (ms)

Water Near Occlusion	0.03
Water Vista Occlusion	0.024
Position Pass	0.093
VDM Displacement	0.014
FBM Diplacement	0.022
Occlusion	0.064
Tessellation	0.22
Normal	0.085
Smoothness	0.047
Occlusion High Res	0.2
Foam	0.25
Composite	0.87
Total	1.919

Performance (ms)

Water Near Occlusion	0.03
Water Vista Occlusion	0.024
Position Pass	0.093
VDM Displacement	0.014
FBM Diplacement	0.022
Occlusion	0.064
Tessellation	0.22
Normal	0.085
Smoothness	0.047
Occlusion High Res	0.2
Foam	0.25
Composite	0.87
Total	1.919

async

-0.582

Water Near Occlusion	0.03
Water Vista Occlusion	0.024
Position Pass	0.093
VDM Displacement	0.014
FBM Diplacement	0.022
Occlusion	0.064
Tessellation	0.22
Normal	0.0
Smoothness	0.0
Occlusion High Res	0.0
Foam	0.0
Composite	0.87
	1.337

FarCry Talks

- Terrain Rendering in ‘FarCry 5’ – Jeremy Moore
  - Wednesday March 21 - Room 22 North Hall
  - 5pm – 6pm
- The Asset Build System of ‘FarCry 5’ – Remi Quenin
  - Wednesday March 21 - Room 2002 West Hall
  - 3 30pm – 4 30pm
- Procedural World Generation of ‘FarCry 5’ – Etienne Carrier
  - Thursday March 22 / Room 3007 West Hall
  - 11 30am – 12 30pm


Thank you!