Lukas Feller

- Graphics Programmer
- Last year Fog, this year VFX :)

Julien Koenen
Technical Director
@ Keen Games

Enshrouded VFX System

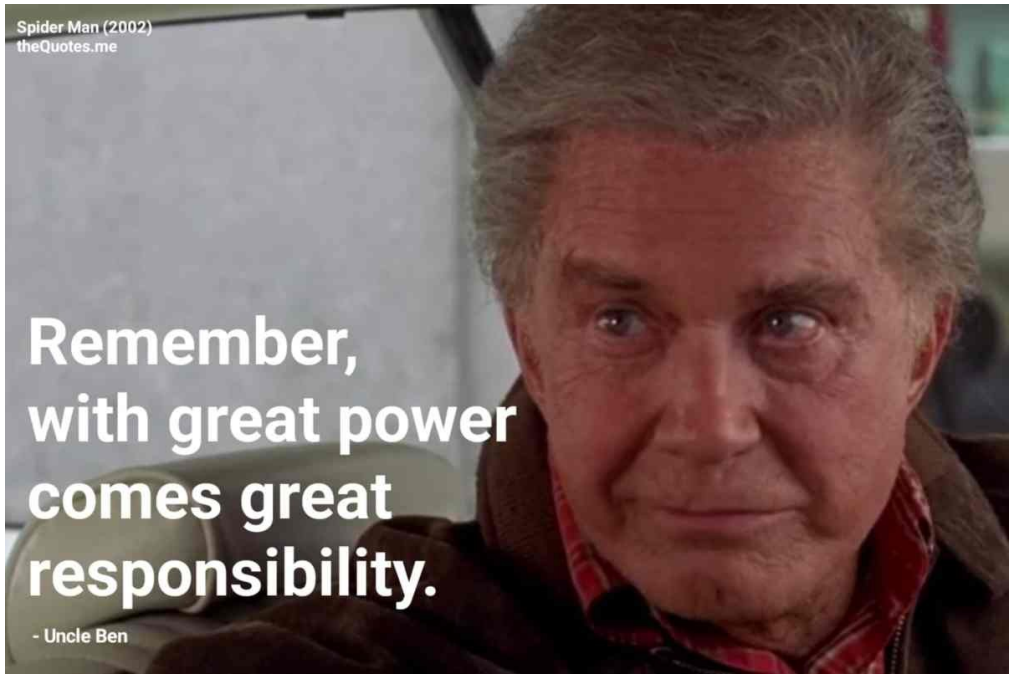
- Existing system showed its age
- Entirely new VFX system
- Designed in close collaboration with the VFX artists

Enshrouded VFX System

- Fully GPU driven
  - Natural choice nowadays
  - Enshrouded has a GPU driven renderer
  - Reuse data which is on the GPU already

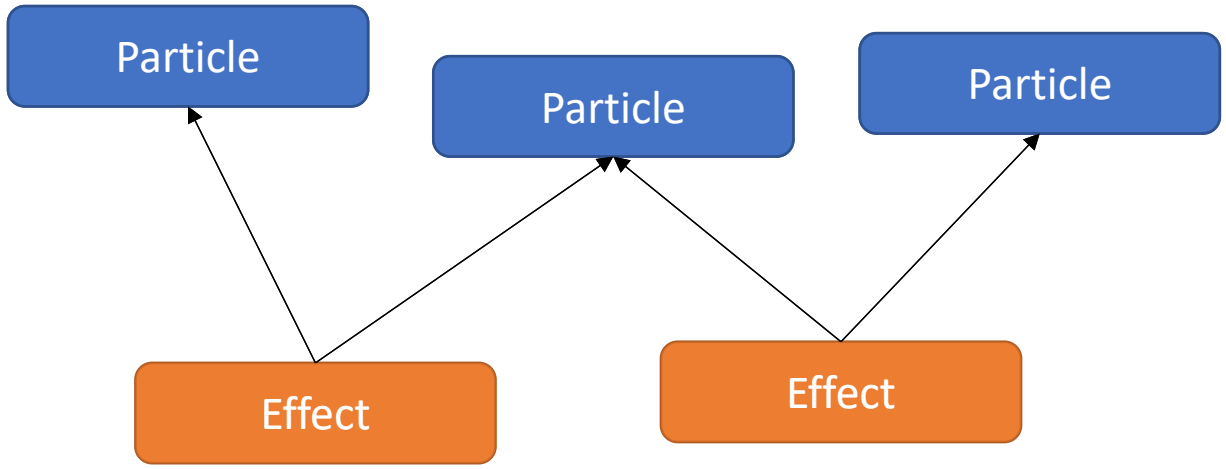
Enshrouded VFX System

- Standard HLSL for scripting
  - Custom scripting language was a maintenance & support burden
  - Well known
  - Lots of existing resources & tooling
  - Maximum freedom for the artist
- Freely import Textures & Meshes
  - From Houdini, EmberGen & others

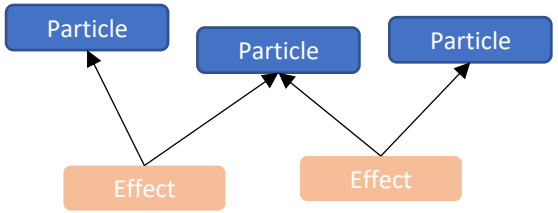


Overview

- Particles
  - Define behavior
- Effects
  - Combine particles
  - Parameterize



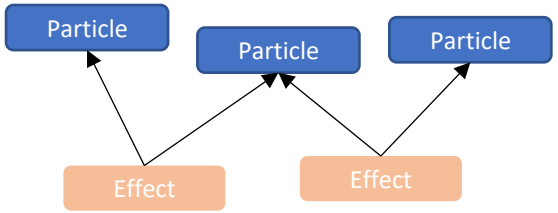
Particle

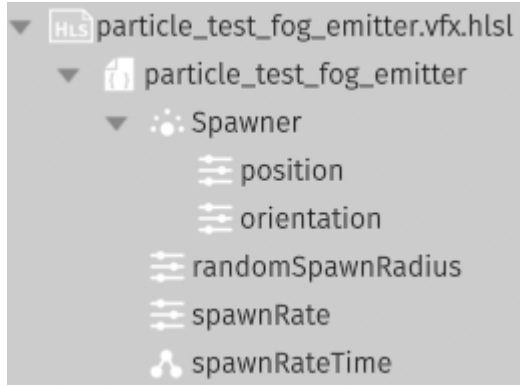


- State
  - Persistent
- Parameters
  - Evaluated every frame
- HLSL code
  - Single function which is executed once per frame per particle

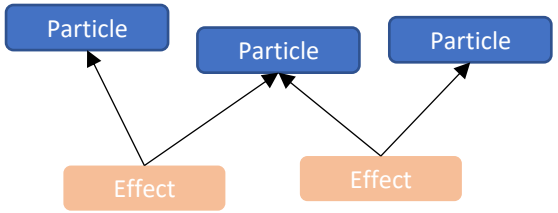
Particle Data

- State, Parameters
- Types:
  - {Float,Int,UInt,Bool}{,2,3,4}
  - Model, Mesh, Image, Sound
  - Emitter





Particle Code



- Single HLSL function
  - Called once per frame per particle
  - Extensive API to sample data and generate output
  - Return value: still alive?

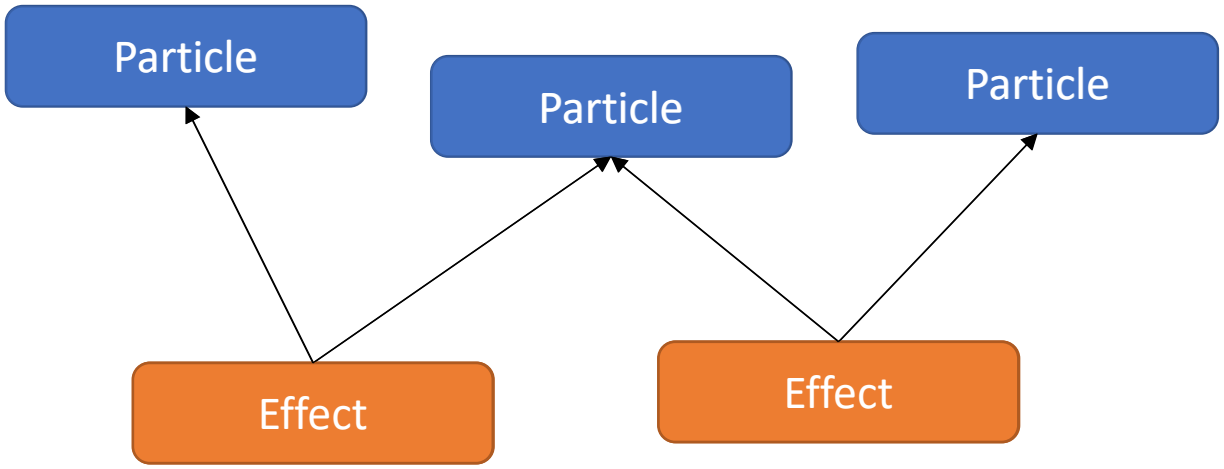
```
bool updateParticle(
        inout VfxParticleState state,
        VfxParticleParameters parameters,
        VfxEffectData effect,
        VfxEngineData engine,
        bool isNew
)
```

- Generated HLSL structs for state & parameters

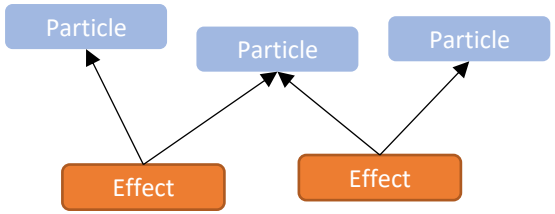
```
struct VfxParticleState
{
    float timer;
    float age;
    float ageNormalized;
    float alpha;
    float brightness;
    …
```

Overview

- Particles
  - Define behavior
- Effects
  - Combine particles
  - Parameterize

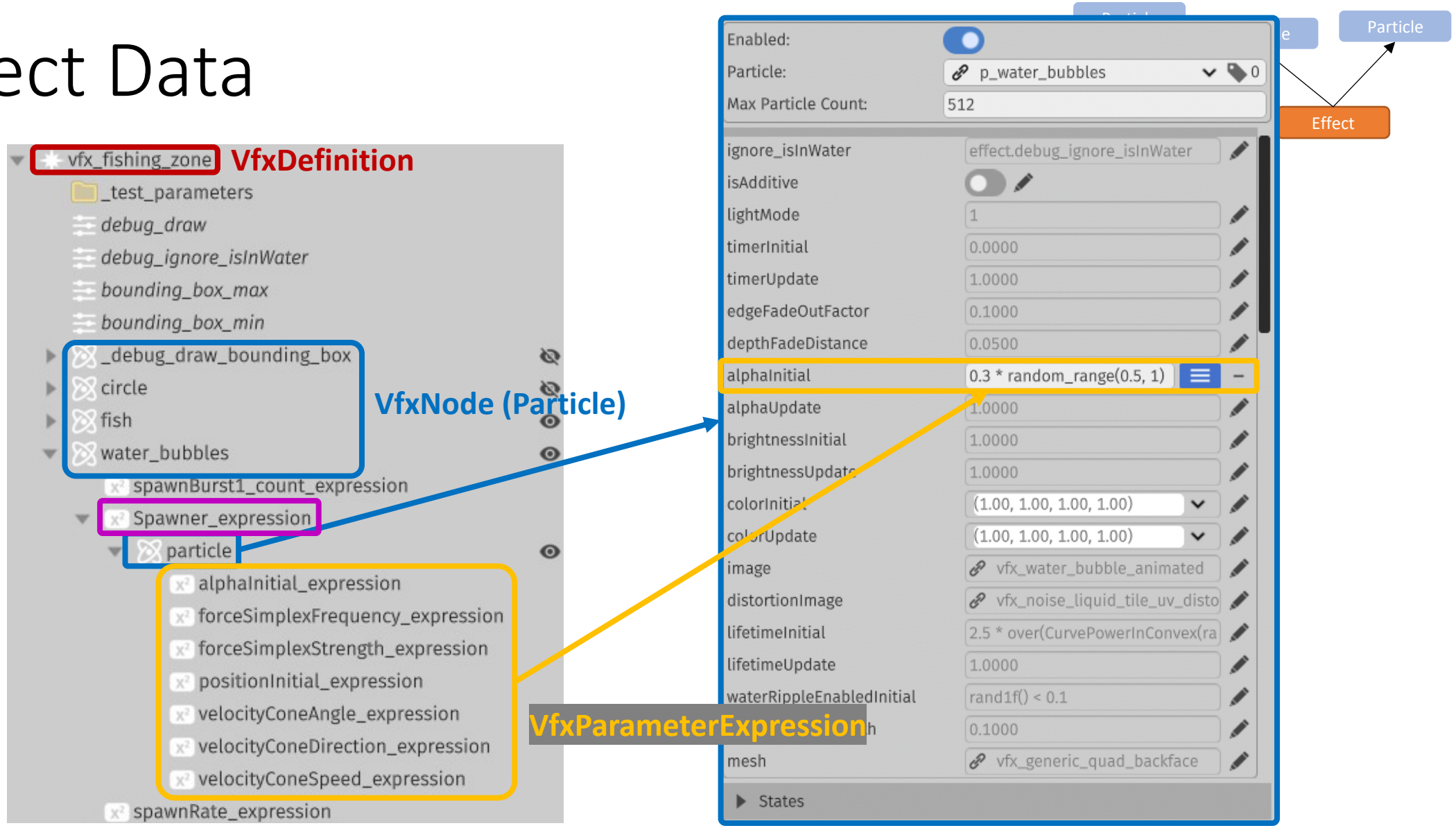


Effect Data

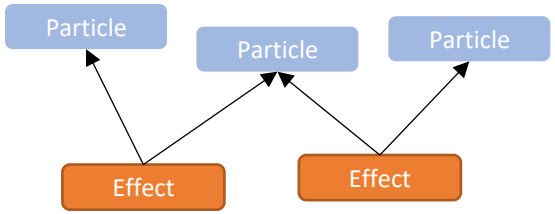


- Instantiated by the game for a VFX
- Hierarchy of particles
  - Parents can spawn children
  - Defines order of execution
- Define values or expressions for particle parameters

Effect Data

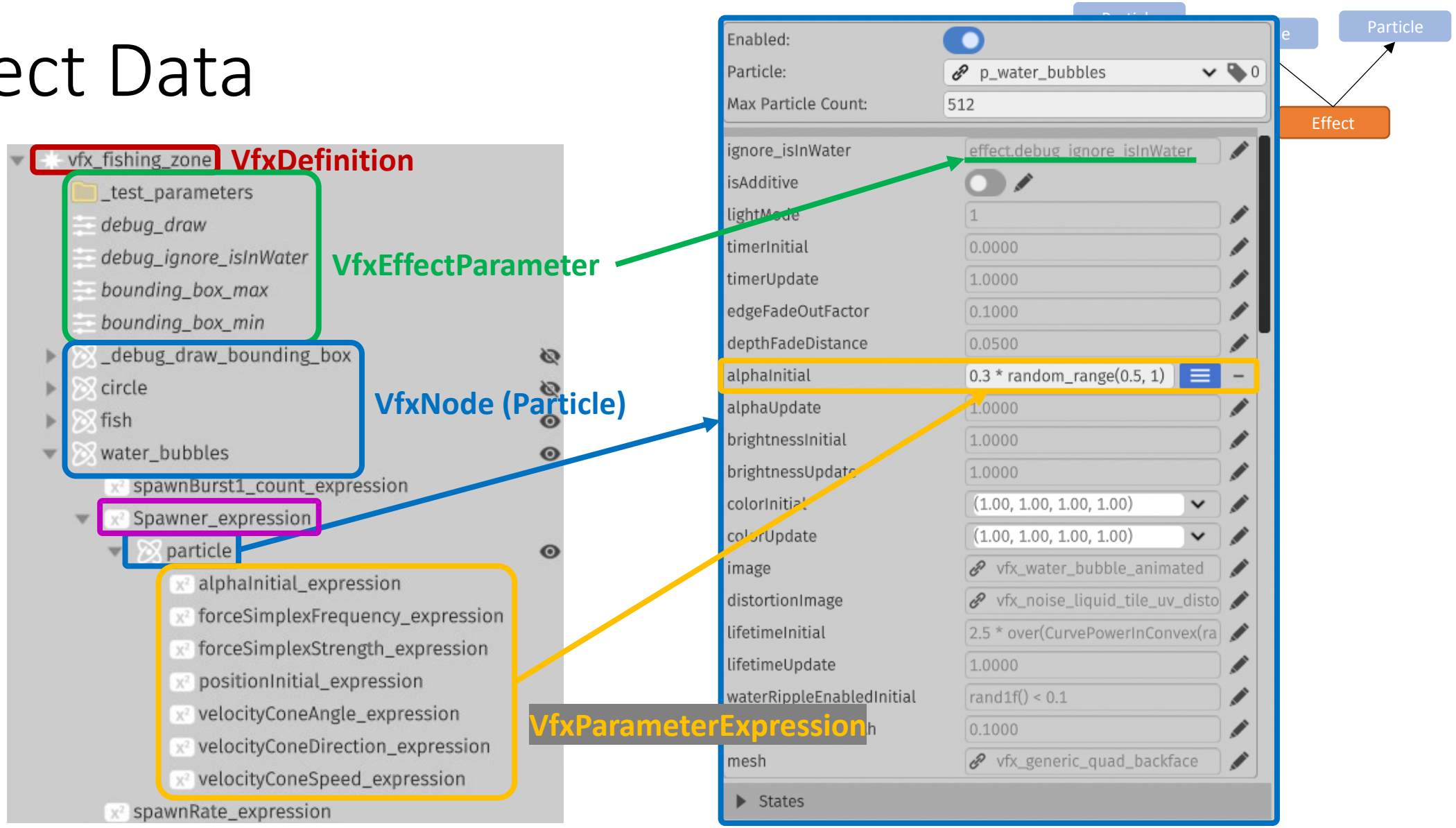

vfx_fishing_zone
VfxDefinition
_test_parameters
debug_draw
debug_ignore_isInWater
bounding_box_max
bounding_box_min
_debug_draw_bounding_box
circle
fish
water_bubbles
VfxNode (Particle)
spawnBurst1_count_expression
Spawner_expression
particle
alphaInitial_expression
forceSimplexFrequency_expression
forceSimplexStrength_expression
positionInitial_expression
velocityConeAngle_expression
velocityConeDirection_expression
velocityConeSpeed_expression
spawnRate_expression
VfxParameterExpression
Enabled:
Particle:
p_water_bubbles
Max Particle Count:
512
ignore_isInWater
effect.debug_ignore_isInWater
isAdditive
lightMode
1
timerInitial
0.0000
timerUpdate
1.0000
edgeFadeOutFactor
0.1000
depthFadeDistance
0.0500
alphaInitial
0.3 * random_range(0.5, 1)
alphaUpdate
1.0000
brightnessInitial
1.0000
(1.00, 1.00, 1.00, 1.00)
image
vfx_water_bubble_animated
distortionImage
vfx_noise_liquid_tile_uv_disto
lifetimeInitial
2.5 * over(CurvePowerInConvex(ra
lifetimeUpdate
1.0000
waterRippleEnabledInitial
rand1f() < 0.1
0.1000
mesh
vfx_generic_quad_backface
States
Particle
Effect

Effect Data

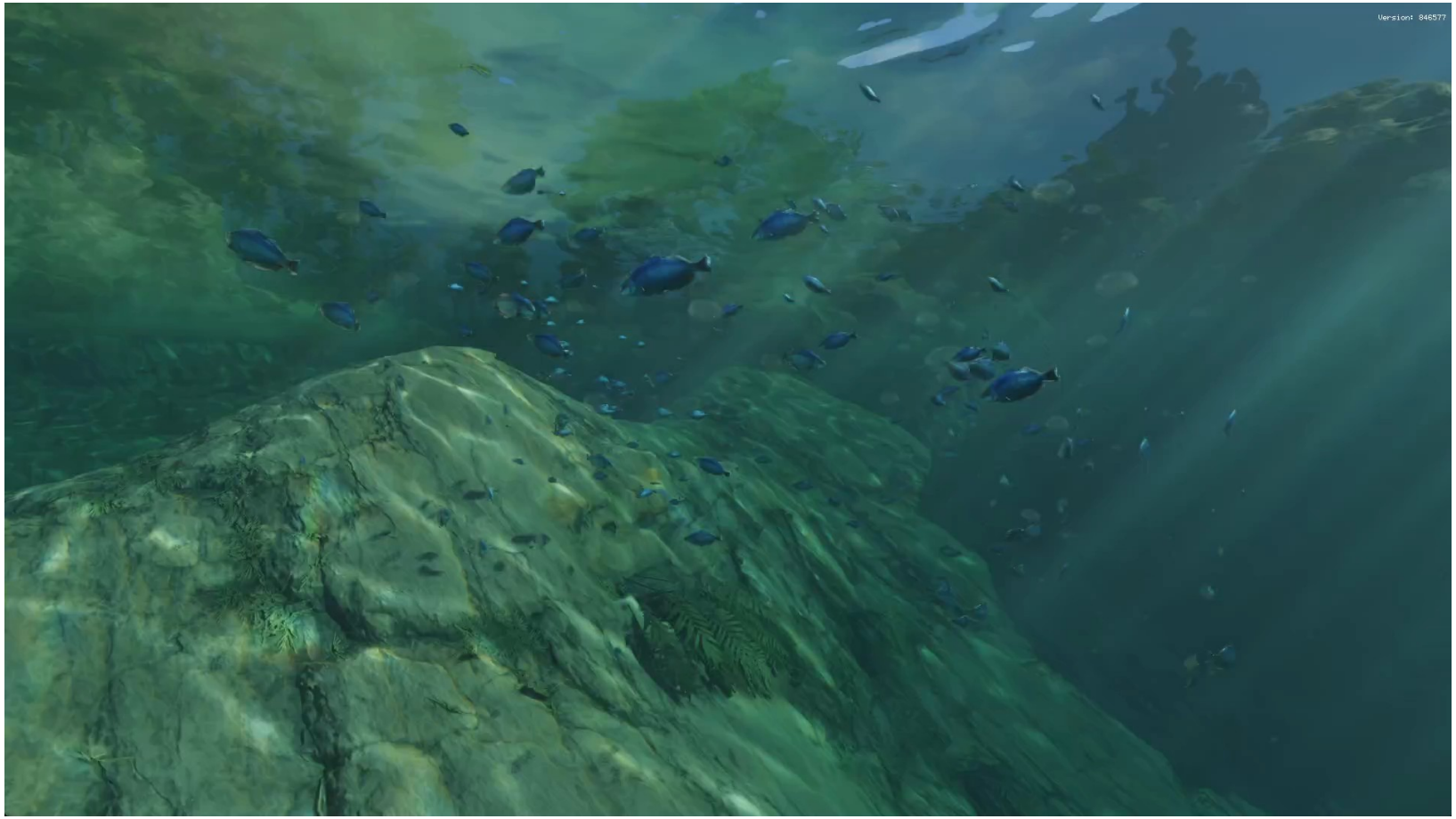


- Effect parameters
  - Arithmetic and resource types
  - Can be set/updated by gameplay code
  - Available in parameter expressions
  - Sourced from a global list of all effect parameters
    - Game code is independent of the configured effect

Effect Data


vfx_fishing_zone
VfxDefinition
_test_parameters
debug_draw
debug_ignore_isInWater
bounding_box_max
bounding_box_min
VfxEffectParameter
_debug_draw_bounding_box
circle
fish
water_bubbles
VfxNode (Particle)
spawnBurst1_count_expression
Spawner_expression
particle
alphaInitial_expression
forceSimplexFrequency_expression
forceSimplexStrength_expression
positionInitial_expression
velocityConeAngle_expression
velocityConeDirection_expression
velocityConeSpeed_expression
VfxParameterExpression
spawnRate_expression
Enabled:
Particle:
p_water_bubbles
Max Particle Count:
512
ignore_isInWater
effect.debug_ignore_isInWater
isAdditive
timerInitial
0.0000
timerUpdate
1.0000
edgeFadeOutFactor
0.1000
depthFadeDistance
0.0500
alphaInitial
0.3 * random_range(0.5, 1)
alphaUpdate
brightnessInitial
1.0000
(1.00, 1.00, 1.00, 1.00)
image
vfx_water_bubble_animated
distortionImage
lifetimeInitial
lifetimeUpdate
waterRippleEnabledInitial
rand1f() < 0.1
mesh
vfx_generic_quad_backface
States
Particle
Effect

Video



Resource Format

- VfxResource
  - VfxParticleResource (1..*)
    - Compute shader
    - Storage buffer
      - Patch points (meshes, images, sounds)
    - Particle state size
  - Particle hierarchy
  - Effect parameters data & layout

Runtime Data

- Instance
  - Parameters
- Particle chunk
  - Compute pipeline
  - State buffer
  - Active indices list (2x for ping-pong)
  - Free indices list
  - Capacity: max particle count
- Pass allocation offsets via push constants

Execution

- Most simple approach
- One dispatch per particle per instance
- Group count from max particle count
- Single descriptor set for everything
- Push constants per dispatch
  - Effect parameters, parameter values, particle state array, …
- Breadth-first over all instances, sorted by pipeline
  - Just one barrier in between hierarchy levels

```
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdBindPipeline( light_06109d38-0ca7-480a-afac-eadabffb9f32 )
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
vkCmdBindPipeline( particle_343af444-b614-44e8-89b1-2f15013bfd94 )
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdPipelineBarrier(1)
vkCmdDispatch(2, 1, 1)
vkCmdBindPipeline( particle_50c3a7fb-cad9-4c43-8ab9-dc24889cc5d5 )
vkCmdPushConstants(VK_SHADER_STAGE_COMPUTE_BIT, (4 bytes))
vkCmdDispatch(1, 1, 1)
```

RX 480:


50,300.000 µs
50,400.000 µs
50,500.000 µs
50,600.000 µs
50,700.000 µs
50,800.000 µs
50,900.000 µs
CS
Overlay
Duration filter
vfx_simulation
vfx_surface_picker
Tick0
prepare
138 148 1567 1847 2101 2120 2251 2269
139 1066 1827 2023 2159 2254
150 1667 1866 2103 2134 2253 2314
1828 2064 2119 2250
703 1976 2135 2301
469 1830 2085 2127 2252 2300
2034 2136 2302
707 2061 2137
2001 2138
2036 2139
1998 2140
2069 2141 2270
2002 2142
1996 2143 2266
2070 2144 2267
2037 2160 2271
1676 1865 2104 2272
2021 2150
2072 2145
1997 2146
2071 2147
1845 2100 2148
1846 2105
1848 2107 2149
1849 2099 2151

Obvious Optimizations

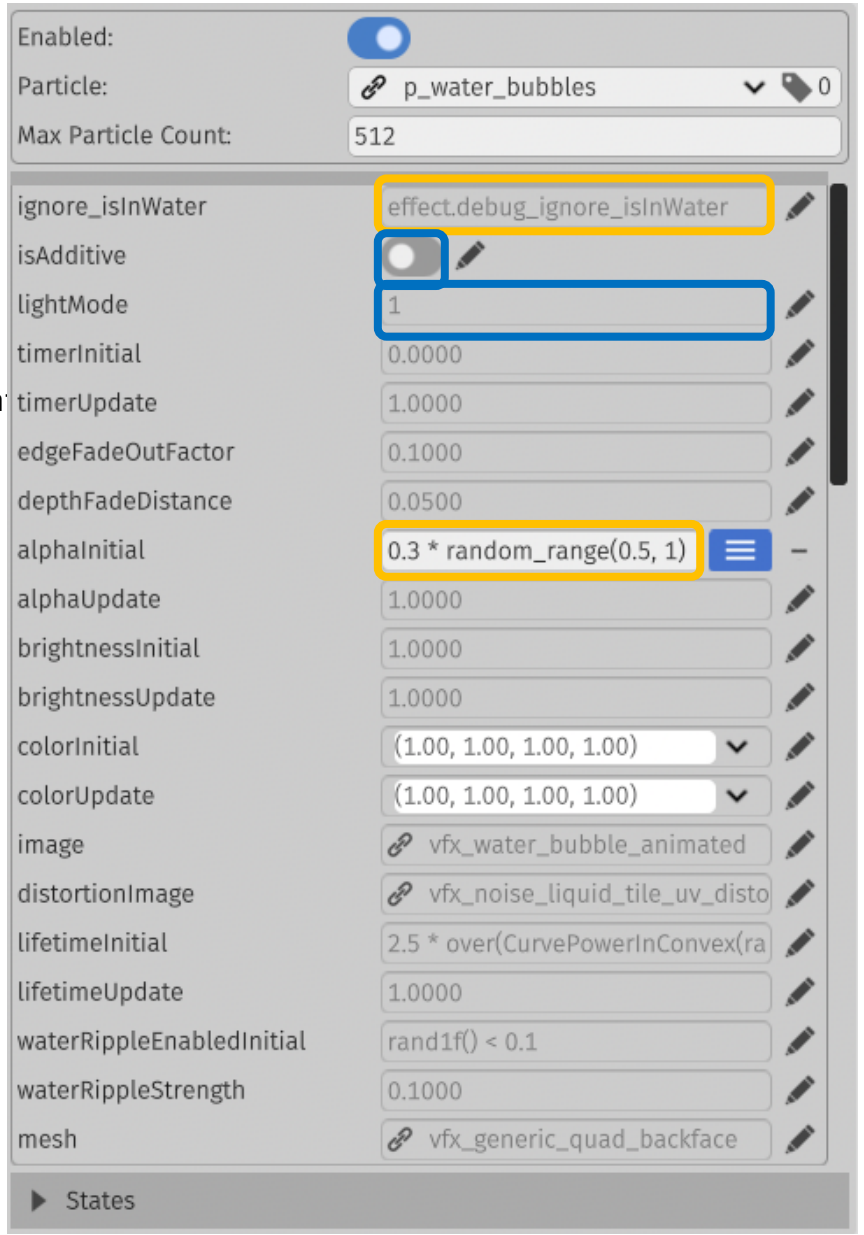
- Many threads don’t operate on active particles and exit early
  - Compute group count from active particle count (instead of maximum)
  - Execute multiple particle chunks in one dispatch
    - From one effect
    - From multiple instances

Shader code

```
void cs_main( uint3 dispatchId : SV_DispatchThreadID )
{
        // ...
        if( dispatchId.x >= particleCount ) return;
        s_particleIndex = g_dataBuffer.Load( s_oldActiveIndicesOffset + dispatchId.x * 4 );
        // ...
        const VfxEffectData effect = fillVfxEffectData( s_effectParameters );
        VfxParticleState particle = unpackParticleState( s_particleState );
        const VfxParticleParameters parameters = evaluateParameters( particle, effect, engine );
        const bool isAlive = updateParticle( particle, parameters, effect, engine, isNew );
        if( isAlive ) { /* pack & store state, append to new active list */ }
        else { /* append to free list */ }
}
```

Generated Code

```
VfxParticleParameters evaluateParameters(
    VfxParticleTypeParameterBuffer parameterBuffer, VfxParticleState sta
    VfxEffectData effect, VfxEngineData engine )
{
    VfxParticleParameters parameters;
    parameters.ignore_isInWater = effect.debug_ignore_isInWater;
    parameters.isAdditive.x = getX( parameterBuffer.isAdditive );
    parameters.lightMode = parameterBuffer.lightMode;
    parameters.timerInitial = parameterBuffer.timerInitial;
    parameters.timerUpdate = parameterBuffer.timerUpdate;
    parameters.edgeFadeOutFactor = parameterBuffer.edgeFadeOutFactor;
    parameters.depthFadeDistance = parameterBuffer.depthFadeDistance;
    parameters.alphaInitial = 0.3 * random_range(0.5, 1);
    // ...
    return parameters;
}
```



Particle API

- Draw
  - Meshes
  - Models
  - Point lights
  - Decals
  - Water
  - Foliage displacement
  - Fog
  - Sound

  ⇒No retained state

  ⇒Supports multiple draws per update
- Debug Draw

Next Slide

Previous transform via hashmap (for now) for motion vectors

Read back on CPU

- Sample
  - Images
  - Fog
  - Fog surface
  - Water
  - Water features
  - Local heightmap
  - World SDF
  - Meshes
- Random [PCG 2014]
- Spawn Particle
- … and many more

Random points from pre-process, see later

Draw Mesh API

```
struct TransparentDrawParameters
{
            float3                  position;
            Quaternion              orientation;
            float3                  scale;
            GpuCombinedImage2D      image;
            GpuCombinedImage2D      normals;
            float3                  color;
            float                   alpha;
            float                   metallic;
            float                   roughness;
            float                   reflectance;
            float2                  uvOffset;
            float2                  uvScale;
            UvDistortion            uvDistortion;
            float                   depthFadeDistance;
            float                   edgeFadeOutFactor;
            AlphaClip               alphaClip;
            int                     vertexAlphaMode;
            bool                    isAdditive;
            bool                    discardInsideWater;
            bool                    discardOutsideWater;
            bool                    depthFadeOnWater;
            VfxSortParameters       sort;
            uint                    lightMode;
            float                   exposureCorrectionFactor;
            Flipbook                flipbook;
};
```

```
enum VfxSortType
{
            None,
            YoungestFirst,
            OldestFirst,
            DistanceToNearPlane,
            DistanceToCamera,
            Custom,
};
struct VfxSortParameters
{
            VfxSortType             type;
            float                   offset;
};
```

```
void drawTransparentMesh(
    VfxMesh mesh,
    TransparentDrawParameters parameters
)
```

Draw Mesh Implementation

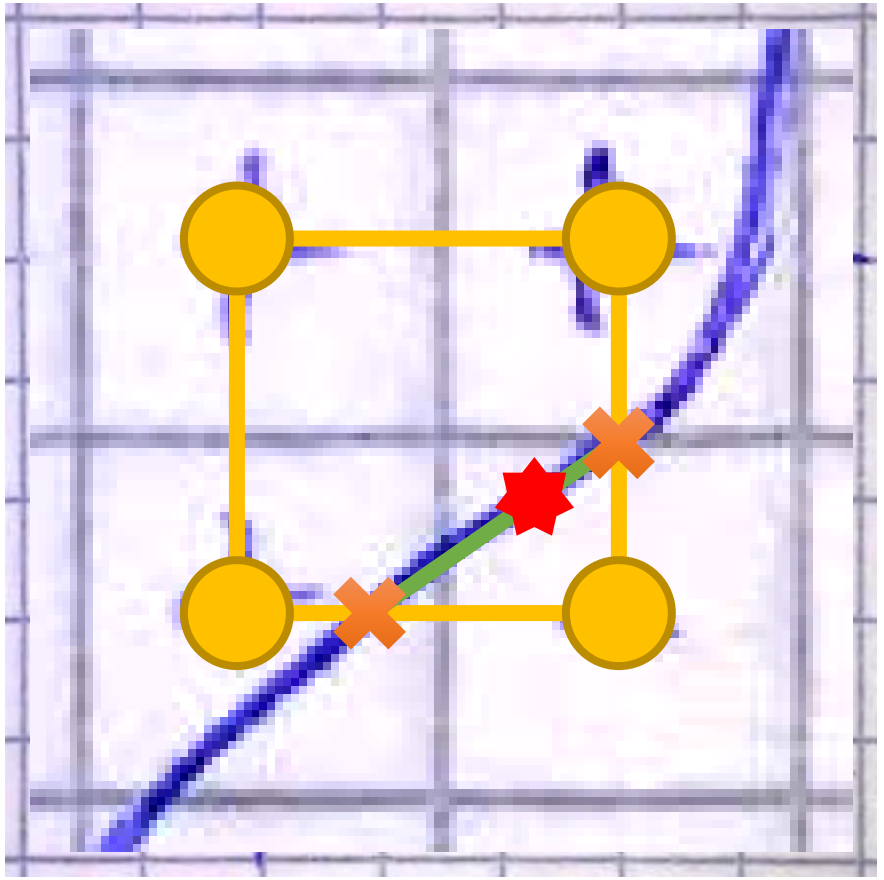
- Sort all draws by sort key using radix sort
- Write new index buffers
  - For above & below water

31-16	15-0
Instance index	Vertex index

- Single draw call for each pass

Surface Points

- Water: CPU
  - Waterfalls, splashes
- Fog: GPU
  - Sample SDF
  - One thread per Voxel
  - If surface: Spawn point with hash offset
  - Hash map to track new points
- VFX: Point list
  - Spawn particles at points



Video

Spawn Particle API

- Spawn parameters defined via data

```
Spawn_Spawner spawn;
spawn.position = position + (rand3f()*2-1) * radius;
spawnParticle( state, spawn );
```

```
bool updateParticle( inout VfxParticleState state,
        VfxParticleParameters parameters, VfxEffectData effect,
        VfxEngineData engine, bool isNew )
{
        if( !effect.isActive )
        {
                return false;
        }

        const uint targetSpawnCount = effect.age / parameters.spawnRate;
        const uint spawnCount = min( targetSpawnCount - state.spawnCount, 4 );
        for( uint spawnIndex = 0u; spawnIndex < spawnCount; ++spawnIndex )
        {
                Spawn_Spawner spawn;
                spawn.position = effect.position + (rand3f()*2-1) * parameters.randomSpawnRadius;
                spawnParticle( state, spawn );
        }
        state.spawnCount = targetSpawnCount;

        return true;
}
```

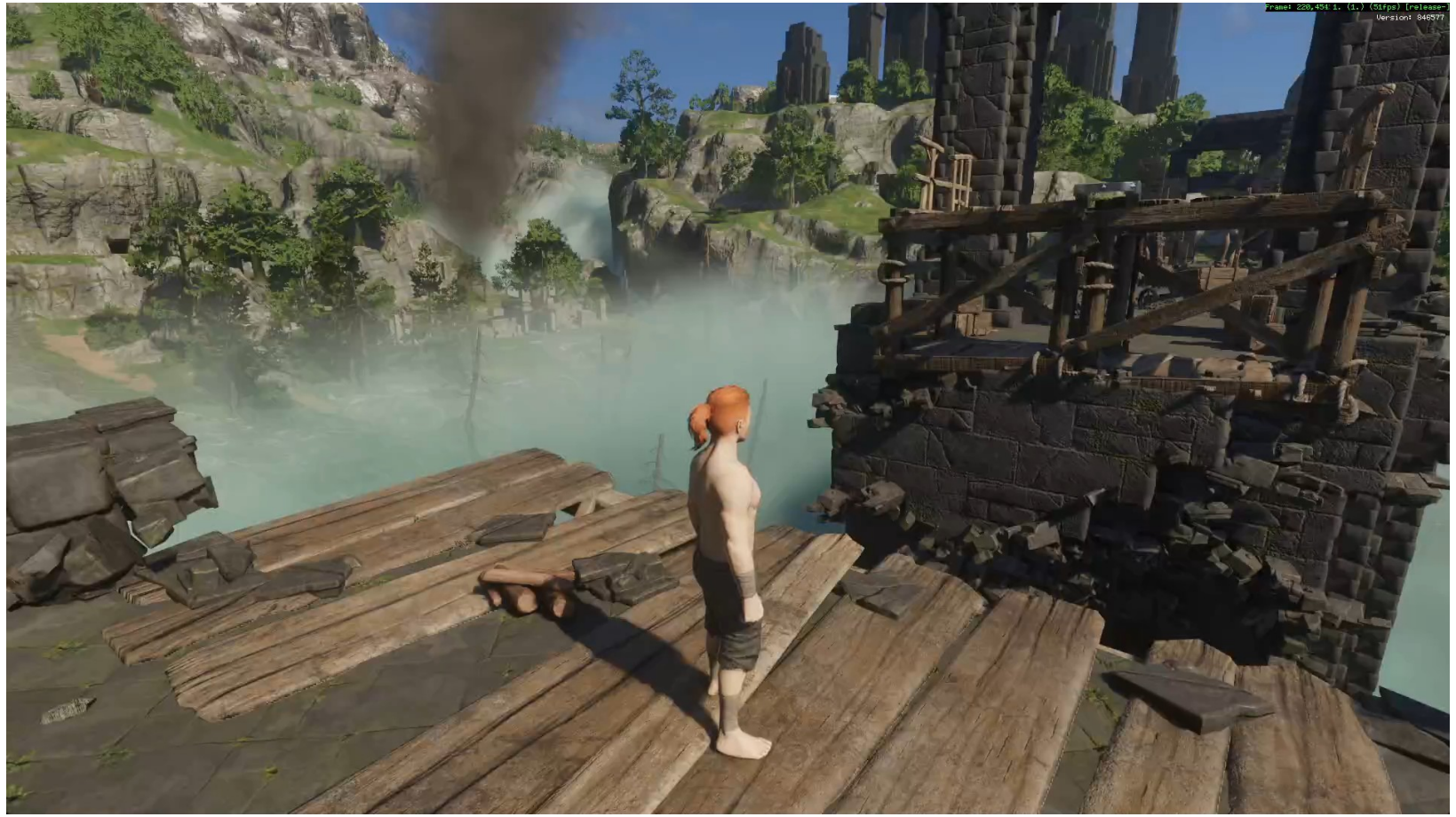
```
bool updateParticle( inout VfxParticleState state,
        VfxParticleParameters parameters, VfxEffectData effect,
        VfxEngineData engine, bool isNew )
{
        if( isNew )
        {
                const VfxSound sound = parameters.sound;
                playSound( state.position, sound );
        }

        drawDebugLine( effect.position, state.position, 1.0 );
        drawDebugCross( state.position, 1.0, 1.0 );

        state.age += engine.fixedTimeStep;
        if( state.age > 1.0 )
        {
                return false;
        }

        return true;
}
```

Video



Video

Video

Video

Video

Conclusion

- Great results
- Technical artistry as written in the books
- No coding support from graphics programmers needed
- Many feature requirements, but seldom blocks
- Many small dispatches are fast

- But…

Issues

- Occasional NaNs
  - Checks in the API
- GPU crashes and hangs
  - Unbounded loops
    - Find and fix
    - Clamp delta-time
    - Maybe add better API/helpers
  - Invalid descriptors
    - Type safety
    - Defensive API
  - Uninitialized variables remain a problem
- Shader compilers struggle
- Bad dependencies in the asset build
- But there’s more…

Some numbers

- 376 Particles
- 1689 Vfx definitions
- 16519 Vfx nodes

- 30+ Minutes runtime compilation 😱

Can you spot the issue?

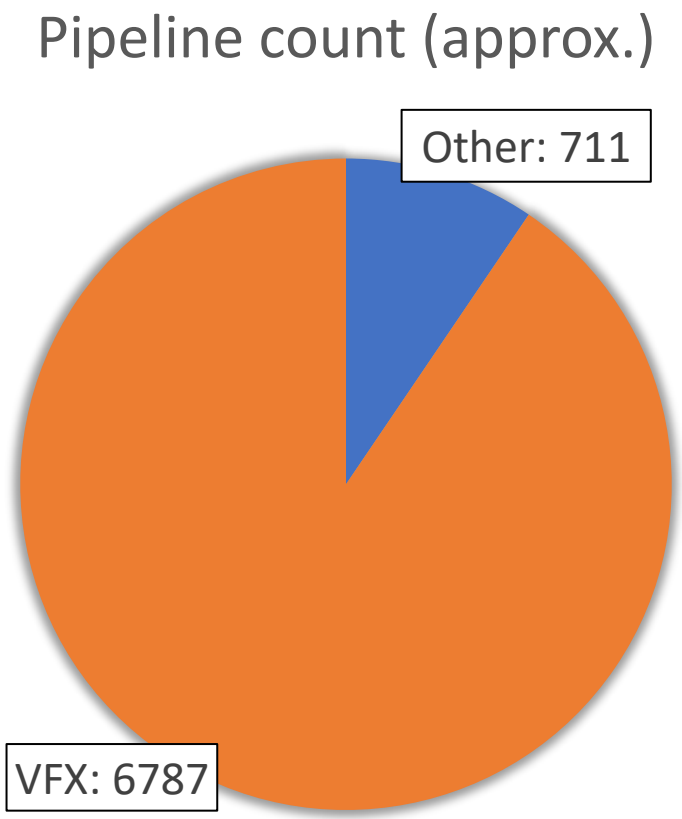
A little anecdote (RTX 2070)

	VK_EXT_memory_budget	vkAllocateMemory	
	8,4 GB	5,5 GB	+2,9 GB
-VFX	6,9 GB	″	

1,5 GB for VFX pipelines!!!

Memory Issues

- Too many pipelines
- Large pipelines



Large shader binaries (RTX 2070)

- VK_KHR_pipeline_executable_properties
- “Binary Size”
- ∑ = 761 MB
- Largest pipeline: 489 KB

Inlining

```
void drawModel( GpuModel model, ModelDrawParameters parameters )
{
    if( !isRenderingEnabled() || !g_update.enableOpaqueMeshes )
    {
        return;
    }

    const uint hashId = ( s_chunkUpdate2.chunkId << 20 ) | ( s_instanceIndex << 12 ) | ( s_particleIndex << 4 ) | s_modelCounter;

    s_modelCounter += 1;

    const ShaderWorldPosition worldPosition = createWorldPosition( parameters.position );

    uint instanceIndex;
    InterlockedAdd( g_modelInstanceCounter[ 0u ], 1u, instanceIndex );

    if( instanceIndex < g_update.maxModelInstanceCount )
    {
        const uint modelIndex = model.modelIndex;

        const ShaderWorldPosition cameraPosition = createWorldPosition( g_update.cameraPosition );
        const float3 cameraRelativePosition = subWorldPosition( worldPosition, cameraPosition );

        PackedShaderWorldTransform transform;
        transform.position              = packShaderWorldPositionUint3( worldPosition );
        transform.scale                 = parameters.scale;
        transform.orientation           = parameters.orientation.value;

        PackedShaderWorldTransform previousTransform = transform;

        uint previousTransformInstanceIndex = hashLookup( g_gpuHashMapSource, hashId );

        if( previousTransformInstanceIndex != KEEN_GPU_HASH_EMPTY )
        {
            previousTransform = g_previousModelTransformSource[ previousTransformInstanceIndex ];
        }

        RenderInstanceData instance = (RenderInstanceData)0;
        instance.packedTransform        = transform;
        instance.dissolveParameters     = float4( parameters.dissolve.offset, 1.0f / max( 0.001f, parameters.dissolve.radius ) );
        instance.dissolveEdgeBrightness = parameters.dissolve.edgeBrightness;
        instance.alphaClip              = asfloat( (uint)( parameters.alphaClip * 255.0f ) & 0xffu );
        instance.modelIndex             = modelIndex;
        instance.groupMask              = 0xffffffffu;
        instance.tintColor              = packColorRGBA( float4( linearTosRGB( saturate( parameters.color ) ), 1.0f ) );
        instance.flagsAndEmissiveFactor = packColorRGBA( float4( saturate( parameters.emissiveFactor ), 0.0f ) ) << 8u;

        g_modelInstances[ instanceIndex ] = instance;

        g_prevModelInstances[ instanceIndex ].packedTransform = previousTransform;

        uint insertResult = hashInsert( g_gpuHashMapTarget, hashId, instanceIndex );

        if( insertResult != KEEN_GPU_HASH_EMPTY )
        {
            g_previousModelTransformTarget[ instanceIndex ] = transform;
        }
    }
    else
    {
        InterlockedAdd( g_modelInstanceCounter[ 0u ], -1u );
    }
}
```

```
if(floor(state.modelIndex) == 0)
    drawModel( parameters.model_0, drawParams );
else if(floor(state.modelIndex) == 1)
    drawModel( parameters.model_1, drawParams );
else if(floor(state.modelIndex) == 2)
    drawModel( parameters.model_2, drawParams );
else if(floor(state.modelIndex) == 3)
    drawModel( parameters.model_3, drawParams );
else if(floor(state.modelIndex) == 4)
    drawModel( parameters.model_4, drawParams );
else if(floor(state.modelIndex) == 5)
    drawModel( parameters.model_5, drawParams );
else if(floor(state.modelIndex) == 6)
    drawModel( parameters.model_6, drawParams );
else if(floor(state.modelIndex) == 7)
    drawModel( parameters.model_7, drawParams );
```

Inlining

```
GpuModel model;

if(floor(state.modelIndex) == 0)
    model = parameters.model_0;
else if(floor(state.modelIndex) == 1)
    model = parameters.model_1;
else if(floor(state.modelIndex) == 2)
    model = parameters.model_2;
else if(floor(state.modelIndex) == 3)
    model = parameters.model_3;
else if(floor(state.modelIndex) == 4)
    model = parameters.model_4;
else if(floor(state.modelIndex) == 5)
    model = parameters.model_5;
else if(floor(state.modelIndex) == 6)
    model = parameters.model_6;
else if(floor(state.modelIndex) == 7)
    model = parameters.model_7;

drawModel( model, drawParams );
```

- Saved 60s offline compile time on one console platform 🥲
- Other patterns:
  - Procedural noise
- Gradient calculations thereof 💀
- Replaced with noise texture

Inlining

- Tried `[noinline]`
- Maps to `OpFunction`/`OpFunctionCall`
- Unstable

Large amount of pipelines

```
VfxParticleParameters evaluateParameters(
    VfxParticleTypeParameterBuffer parameterBuffer,
    VfxParticleState state, VfxEffectData effect, VfxEngineData engine )
{
    VfxParticleParameters parameters;
    parameters.ignore_isInWater = effect.debug_ignore_isInWater;
    parameters.isAdditive.x = getX( parameterBuffer.isAdditive );
    parameters.lightMode = parameterBuffer.lightMode;
    parameters.timerInitial = parameterBuffer.timerInitial;
    parameters.timerUpdate = parameterBuffer.timerUpdate;
    parameters.edgeFadeOutFactor = parameterBuffer.edgeFadeOutFactor;
    parameters.depthFadeDistance = parameterBuffer.depthFadeDistance;
    parameters.alphaInitial = 0.3 * random_range(0.5, 1);
    // ...
    return parameters;
}
```

Failed: Expressions as simple data

- Collect all expressions per particle
- Replace with expression index
- Shaders became too large and unstable

WIP: Data driven evaluation of expressions

- Get rid of VfxNode -> Code dependency
- Parse HLSL expression
- Simplify as data
- VM is too general
- Parameterize common expressions
- Data driven variable reads
- Use GS memory to get rid of inlining

```
0.2 * random_range(0.3, 1)
0.2 * random_range(0.7, 1)
0.5 * random_range(0.7, 1)
1 * random_range(0.1, 1)
1 * random_range(0.2, 1)
1 * random_range(0.3, 1)
1 * random_range(0.5, 1)
1 * random_range(0.7, 1)
1 * random_range(0.8, 1)
1.2 * random_range(0.2, 1)

=> $0 * random_range($1, $2)
```

Conclusion Pt. 2

- Artists are not graphics programmers
  - Unusual shader code
  - Probably should go through reviews…
- GPUs are not really CPUs
  - Be aware of code that looks nice but will execute horribly
- Shader permutations can still be problematic
  - Be aware of your dependencies
  - An uber shader is not always the solution

Thank you!

References

- [PCG 2014] Melissa E. O'Neill, PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation