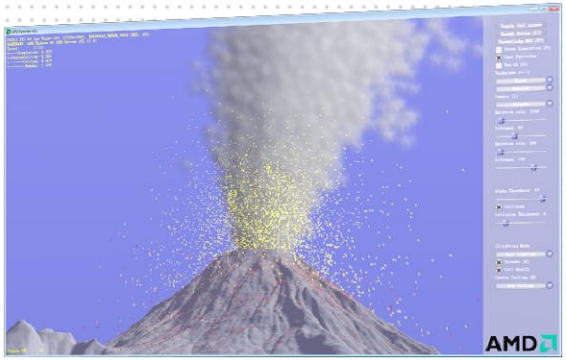
Compute-Based GPU Particle Systems



Gareth Thomas
Developer Technology Engineer, AMD

GAME DEVELOPERS CONFERENCE 2014
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21

Agenda

- Overview
- Collisions
- Sorting
- Tiled Rendering
- Conclusions

Overview

- Why use the GPU?
  - Highly parallel workload
  - Free your CPU to do game code
  - Leverage compute

Overview

- Emit
- Simulate
- Sort
- Rendering
  - Rasterization or Tiled Rendering

Data Structures

Particle Pool

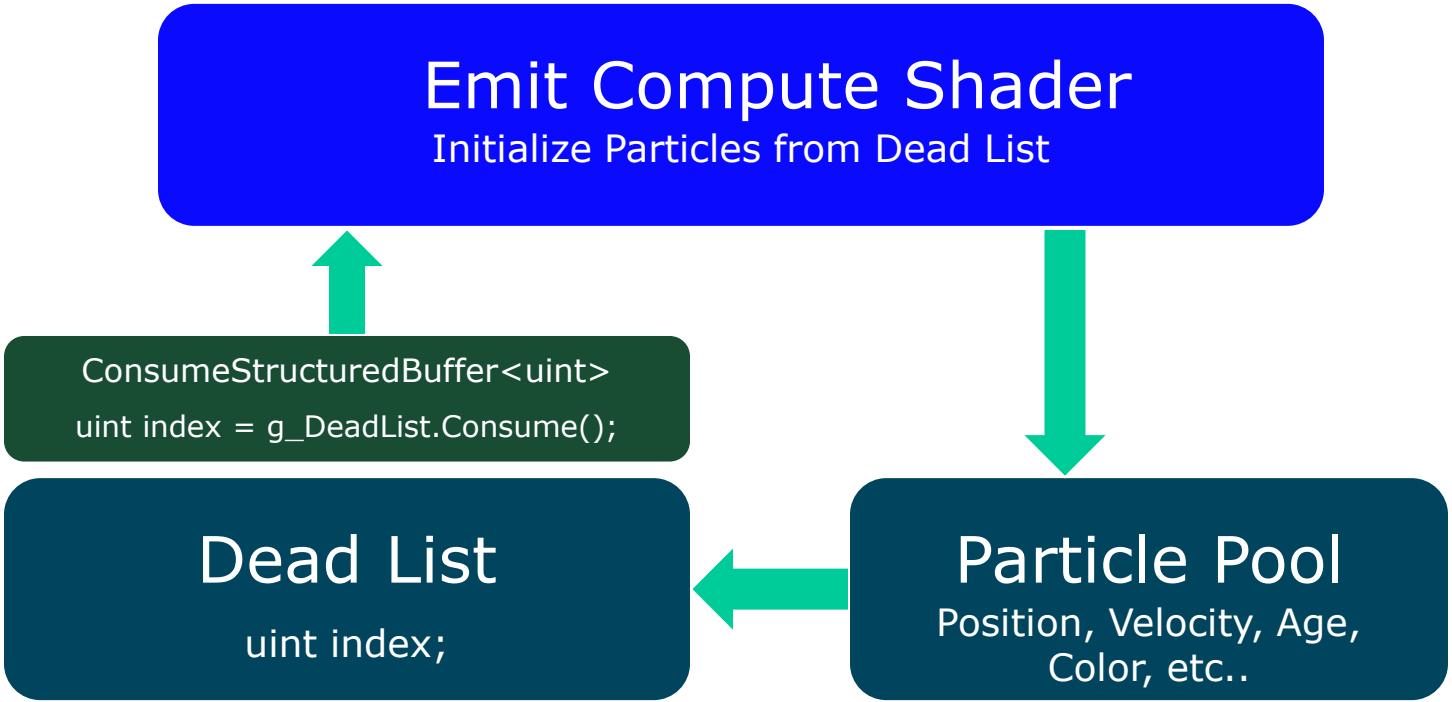
Position, Velocity, Age, Color, etc..

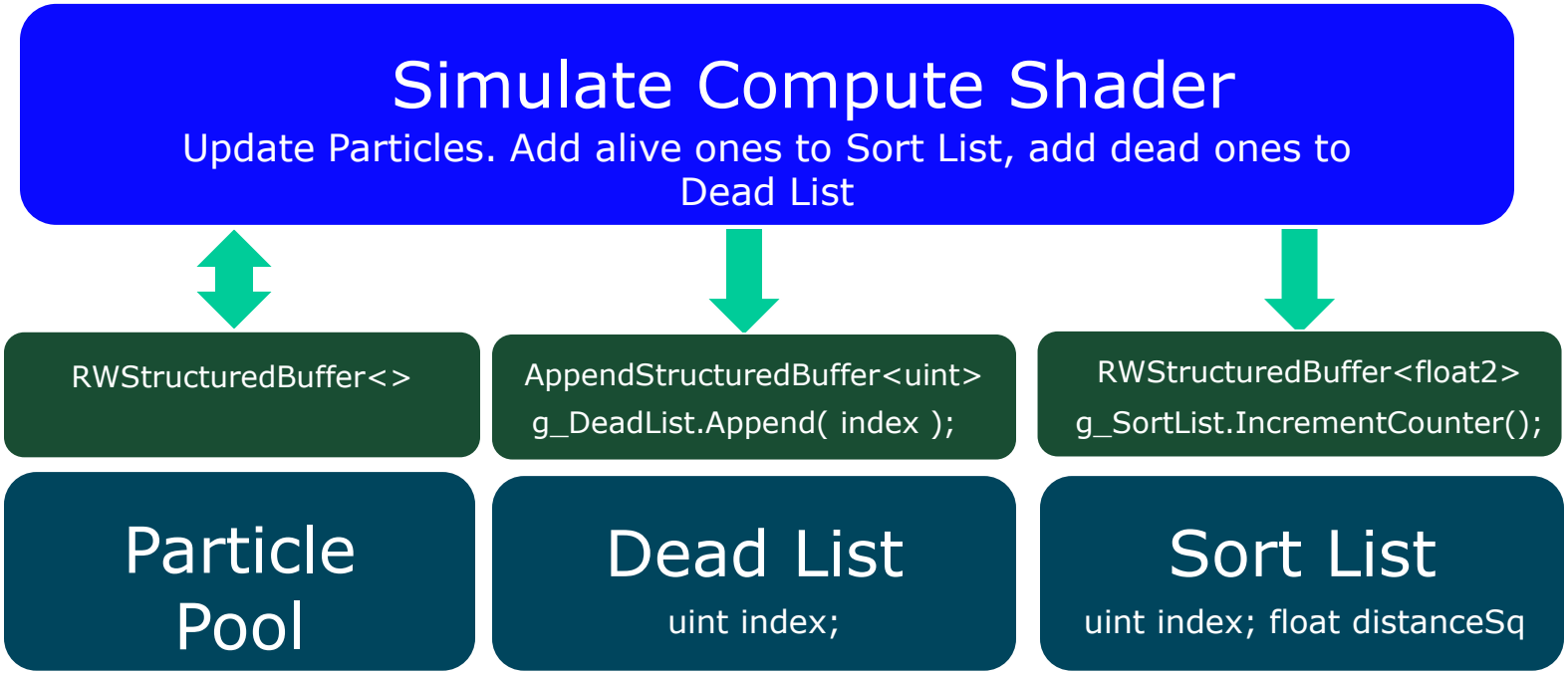
Sort List

uint index; float distanceSq;

Dead List

uint index;

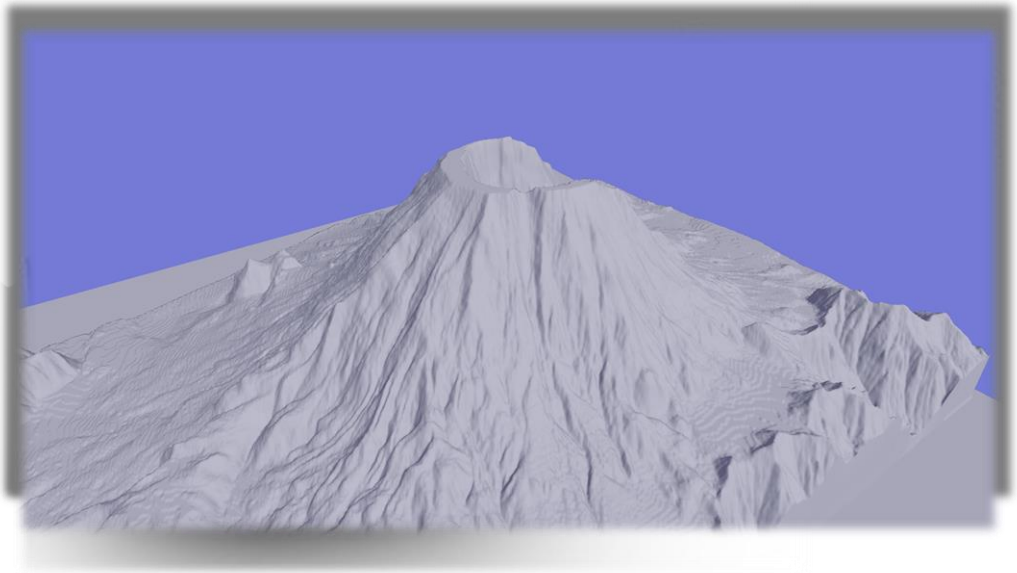

Emit Compute Shader
Initialize Particles from Dead List
ConsumeStructuredBuffer<uint>
uint index = g_DeadList.Consume();
Dead List
uint index;
Particle Pool
Position, Velocity, Age,
Color, etc..


Simulate Compute Shader
Update Particles. Add alive ones to Sort List, add dead ones to Dead List
RWStructuredBuffer<>
AppendStructuredBuffer<uint>
g_DeadList.Append(index);
RWStructuredBuffer<float2>
g_SortList.IncrementCounter();
Particle Pool
Dead List
uint index;
Sort List
uint index; float distanceSq

Collisions

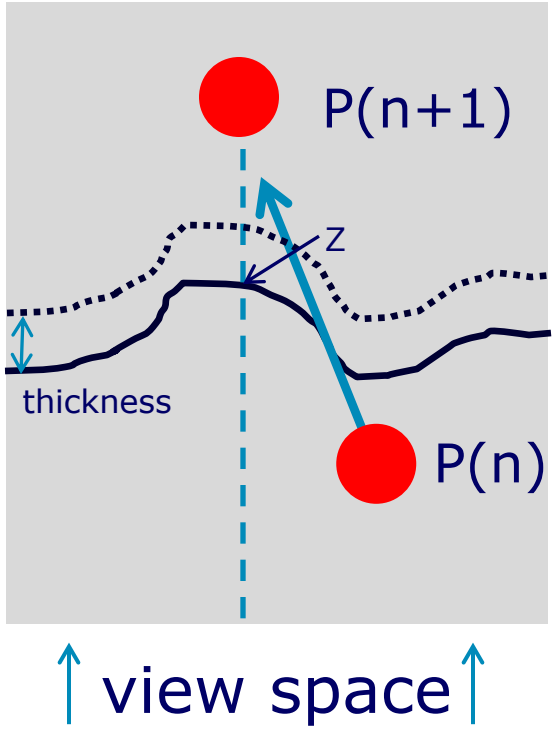
- Primitives
- Heightfield
- Voxel data
- Depth buffer

[Tchou11]



Depth Buffer Collisions

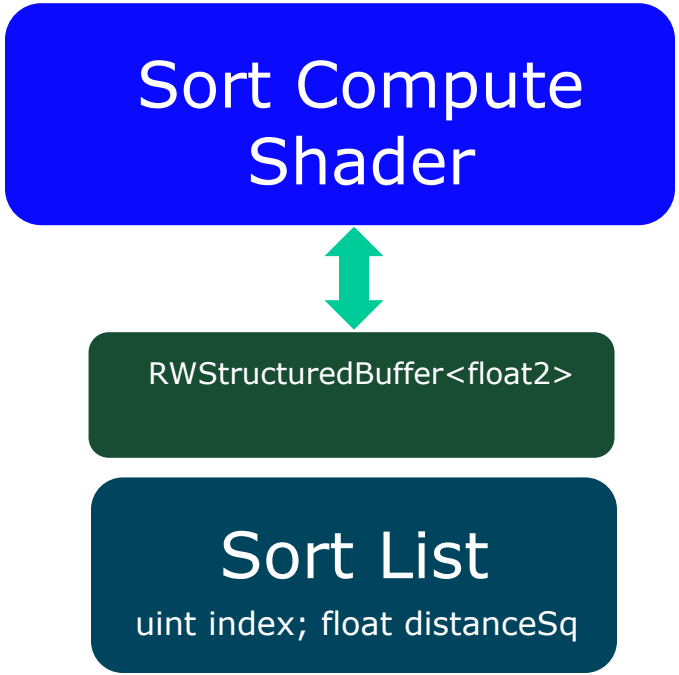
- Project particle into screen space
- Read Z from depth buffer
- Compare view-space particle position vs view-space position of Z buffer value
- Use thickness value



Depth Buffer Collision Response

- Use normal from G-buffer
- Or take multiple taps a depth buffer
  - Watch out for depth discontinuities

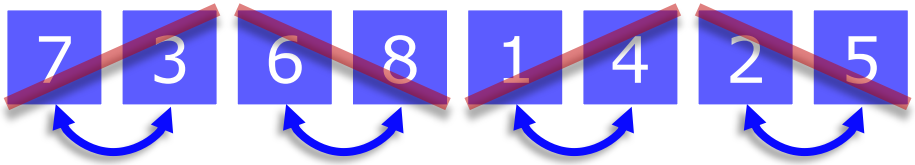
- Sort for correct alpha blending
  - Additive blending just saturates the effect
  - Bitonic sort parallelizes well on GPU



Bitonic Sort

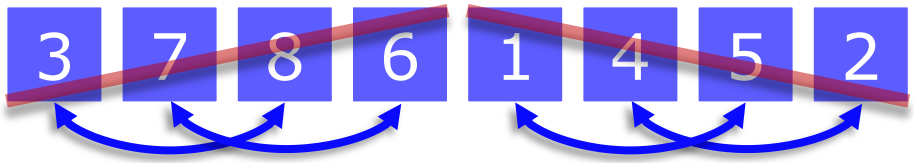

```
for( subArraySize=2; subArraySize<ArraySize; subArraySize*=2)
{
    for( compareDist=subArraySize/2; compareDist>0; compareDist/=2)
    {
        // Begin: GPU part of the sort
        for each element n
            n = selectBitonic(n, n^compareDist);
        // End: GPU part of the sort
    }
}
```

Bitonic Sort (Pass 1)



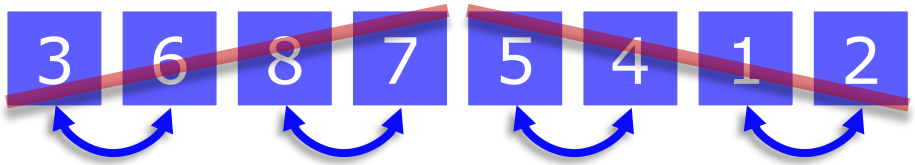

```
for( subArraySize=2; subArraySize<ArraySize; subArraySize*=2)        // subArraySize == 2
{
    for( compareDist=subArraySize/2; compareDist>0; compareDist/=2)  // compareDist == 1
    {
        // Begin: GPU part of the sort
        for each element n
            n = selectBitonic(n, n^compareDist);
        // End: GPU part of the sort
    }
}
```

Bitonic Sort (Pass 2)



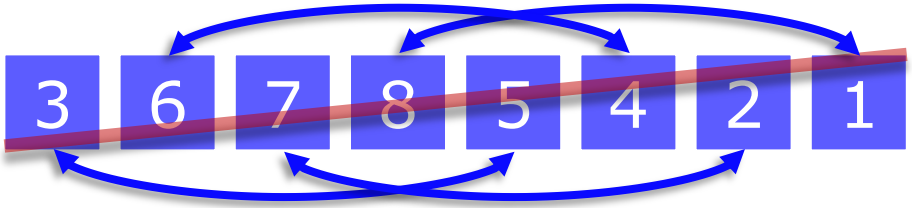

```
for( subArraySize=2; subArraySize<ArraySize; subArraySize*=2)         // subArraySize == 4
{
    for( compareDist=subArraySize/2; compareDist>0; compareDist/=2)   // compareDist == 2
    {
        // Begin: GPU part of the sort
        for each element n
            n = selectBitonic(n, n^compareDist);
        // End: GPU part of the sort
    }
}
```

Bitonic Sort (Pass 3)



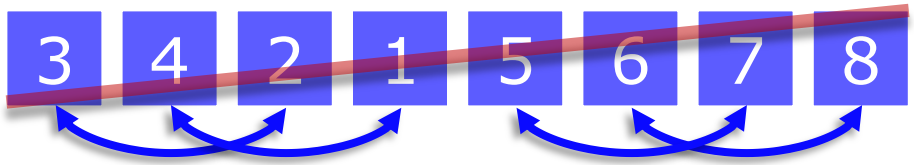

```
for( subArraySize=2; subArraySize<ArraySize; subArraySize*=2)         // subArraySize == 4
{
    for( compareDist=subArraySize/2; compareDist>0; compareDist/=2)   // compareDist == 1
    {
        // Begin: GPU part of the sort
        for each element n
            n = selectBitonic(n, n^compareDist);
        // End: GPU part of the sort
    }
}
```

Bitonic Sort (Pass 4)



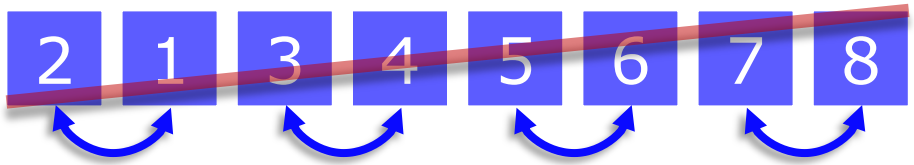

```
for( subArraySize=2; subArraySize<ArraySize; subArraySize*=2)         // subArraySize == 8
{
    for( compareDist=subArraySize/2; compareDist>0; compareDist/=2)   // compareDist == 4
    {
        // Begin: GPU part of the sort
        for each element n
            n = selectBitonic(n, n^compareDist);
        // End: GPU part of the sort
    }
}
```

Bitonic Sort (Pass 5)

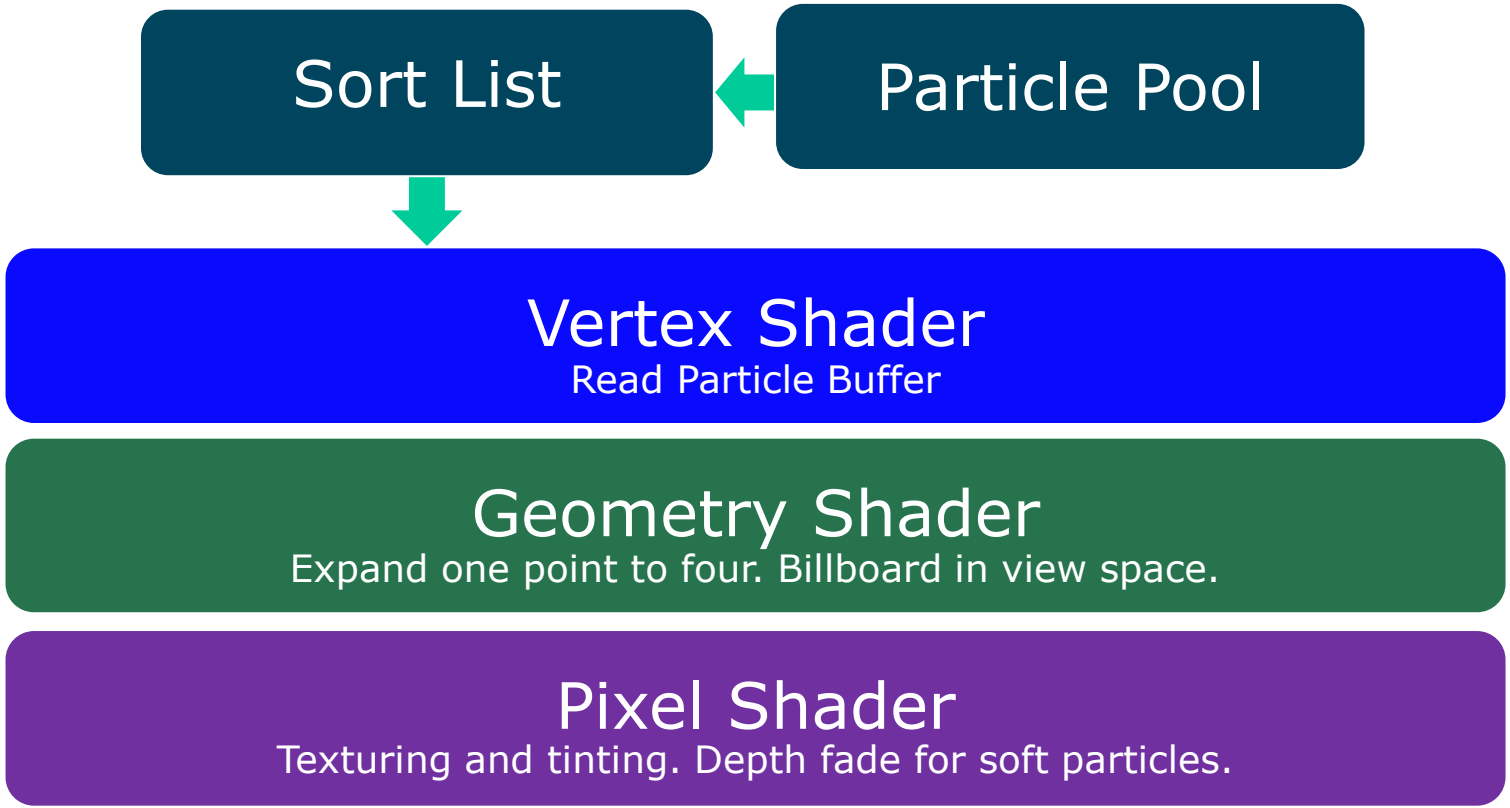


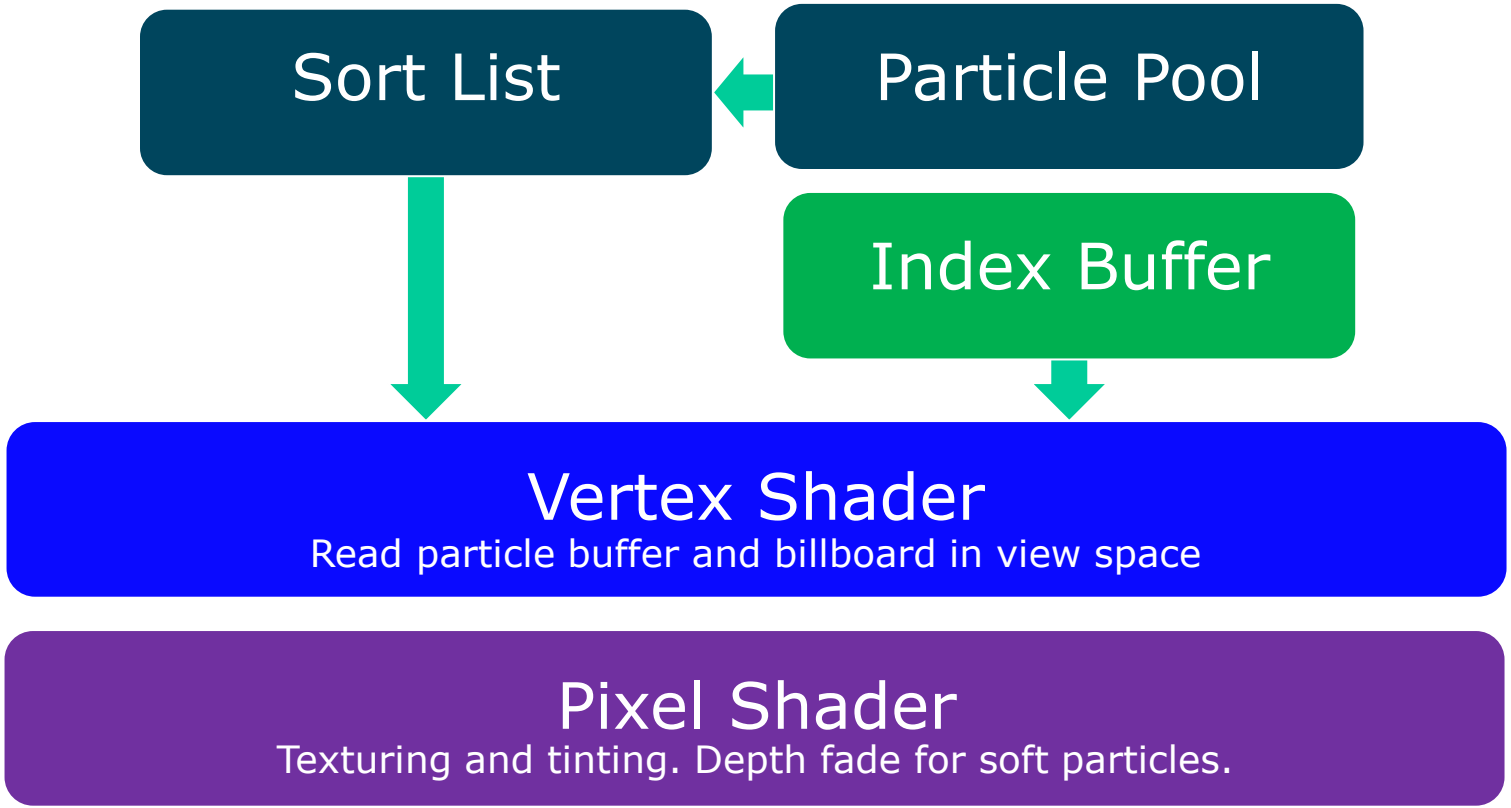

```
for( subArraySize=2; subArraySize<ArraySize; subArraySize*=2)         // subArraySize == 8
{
    for( compareDist=subArraySize/2; compareDist>0; compareDist/=2)   // compareDist == 2
    {
        // Begin: GPU part of the sort
        for each element n
            n = selectBitonic(n, n^compareDist);
        // End: GPU part of the sort
    }
}
```

Bitonic Sort (Pass 6)



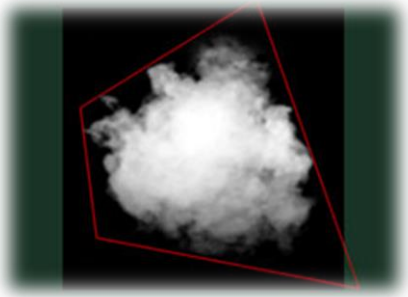

```
for( subArraySize=2; subArraySize<ArraySize; subArraySize*=2)         // subArraySize == 8
{
    for( compareDist=subArraySize/2; compareDist>0; compareDist/=2)   // compareDist == 1
    {
        // Begin: GPU part of the sort
        for each element n
            n = selectBitonic(n, n^compareDist);
        // End: GPU part of the sort
    }
}
```


Sort List
Particle Pool
Vertex Shader
Read Particle Buffer
Geometry Shader
Expand one point to four. Billboard in view space.
Pixel Shader
Texturing and tinting. Depth fade for soft particles.


Sort List
Particle Pool
Index Buffer
Vertex Shader
Read particle buffer and billboard in view space
Pixel Shader
Texturing and tinting. Depth fade for soft particles.

Rasterization

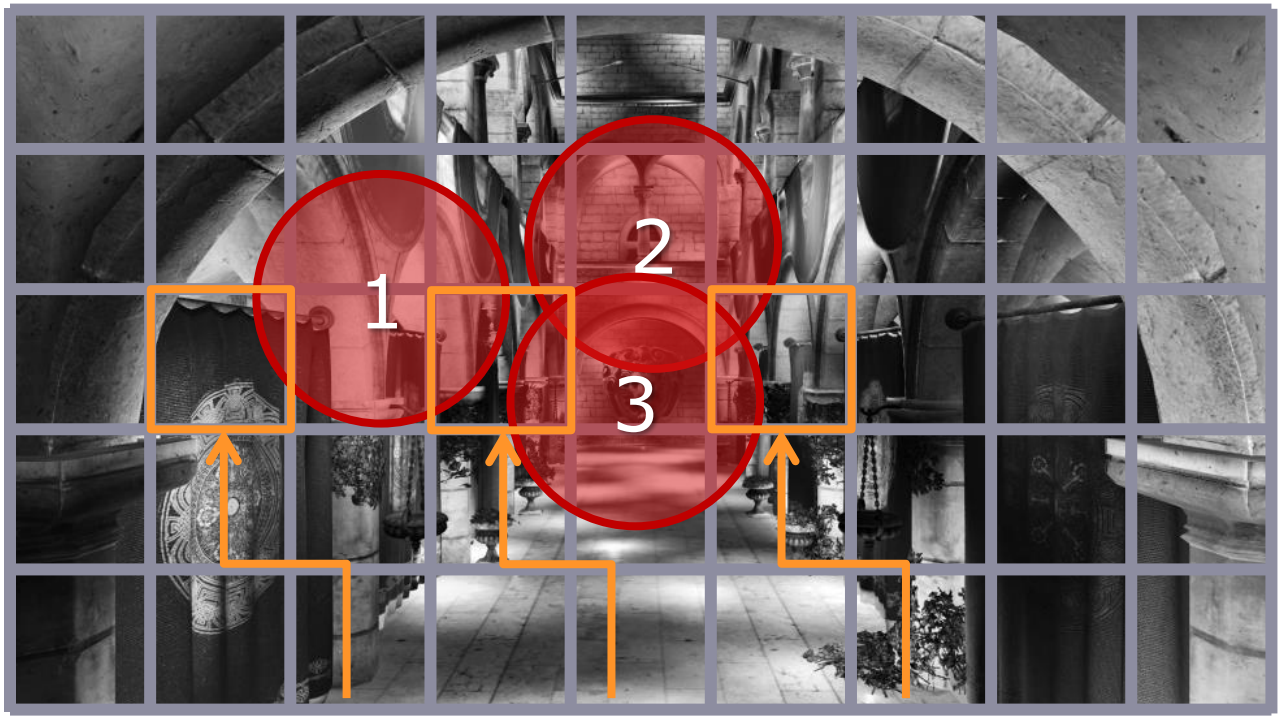
- DrawIndexedIndirectInstanced() or DrawIndirectInstanced()
  - VertexId = particle index (or VertexId/4 for VS billboarding)
  - 1 instance
- Heavy overdraw on large particles – restricts game design
  - Fit polygon billboard around texture [Persson09]
  - Render to half size buffer [Cantlay07]
    - Sorting issues
    - Loss of fidelity



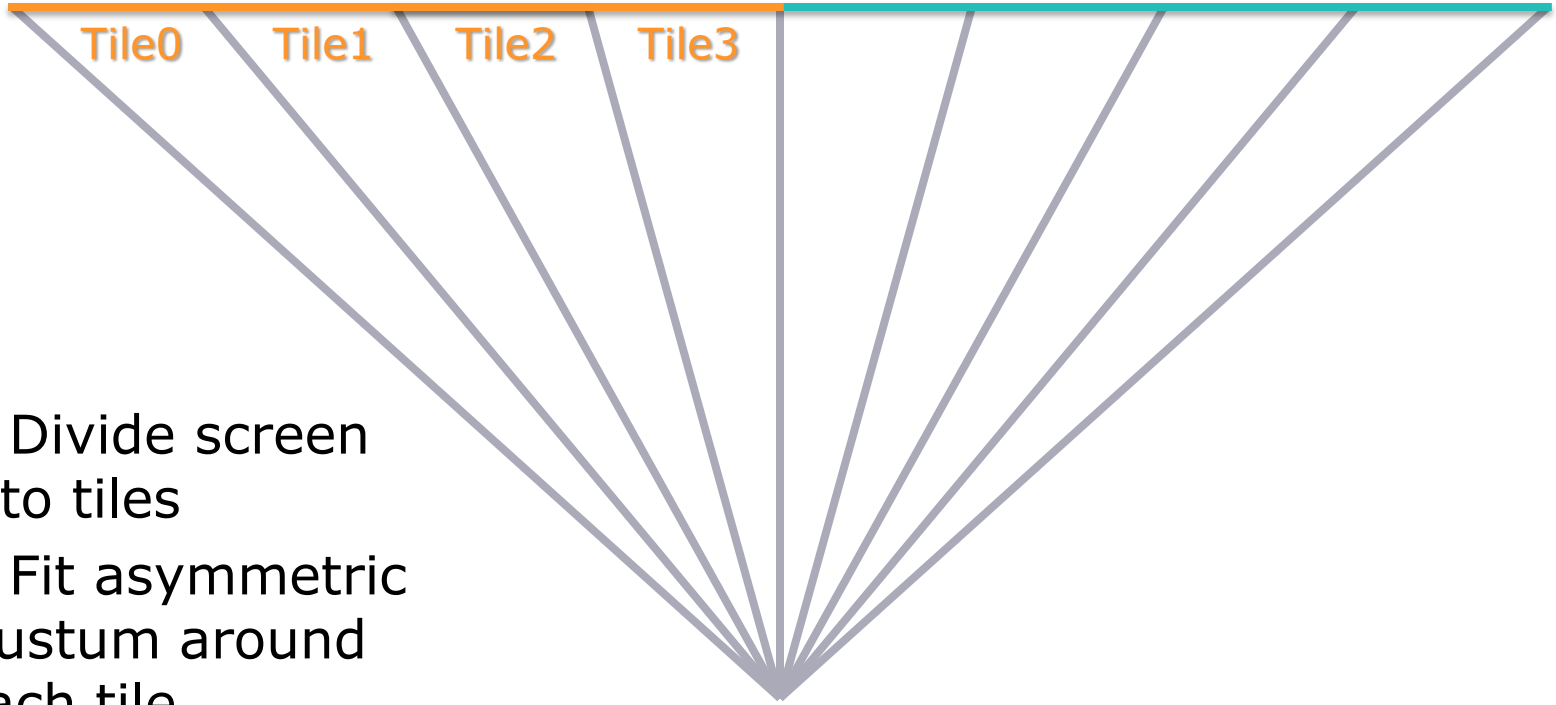
Tiled Rendering

- Inspired by Forward+ [Harada12]
  - Screen-space binning of particles instead of lights
- Per-tile
  - Cull & Sort
  - Per pixel/thread
    - Evaluate color of each particle
    - Blend together
- Composite back onto scene

Tiled ~~light~~ *particle* culling

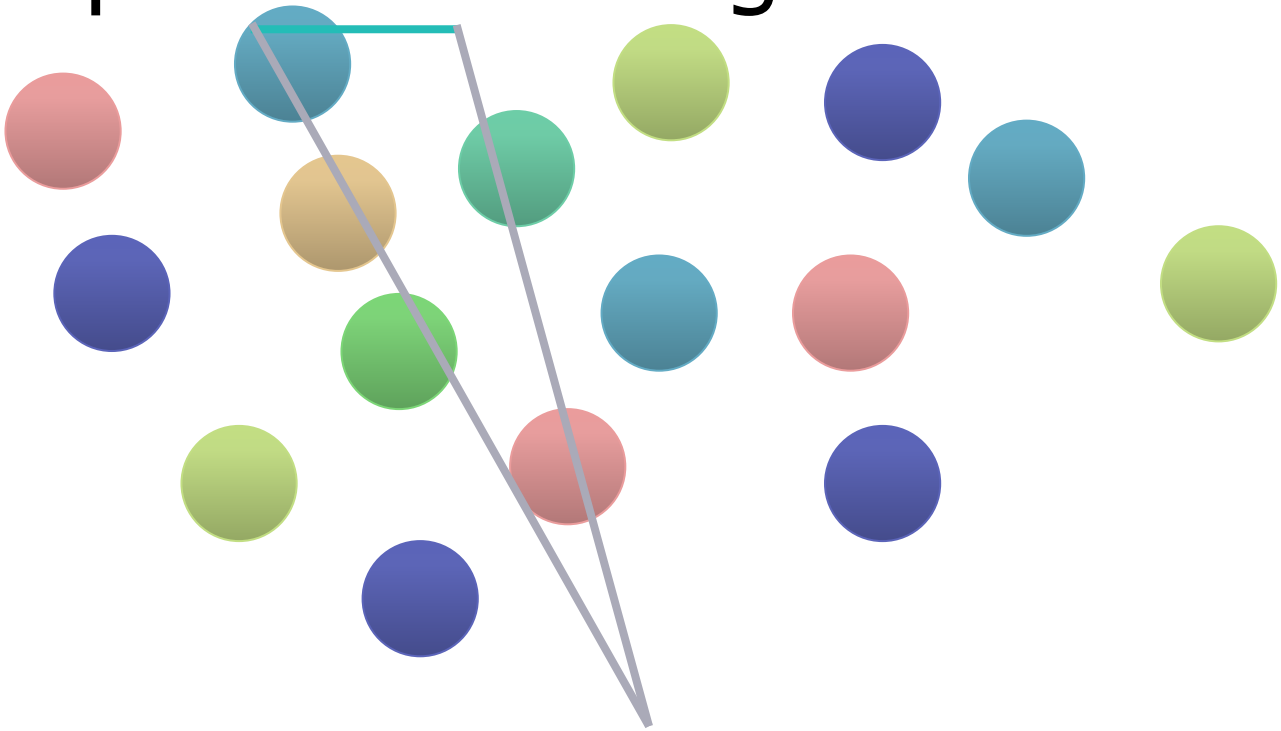


Tiled particle culling



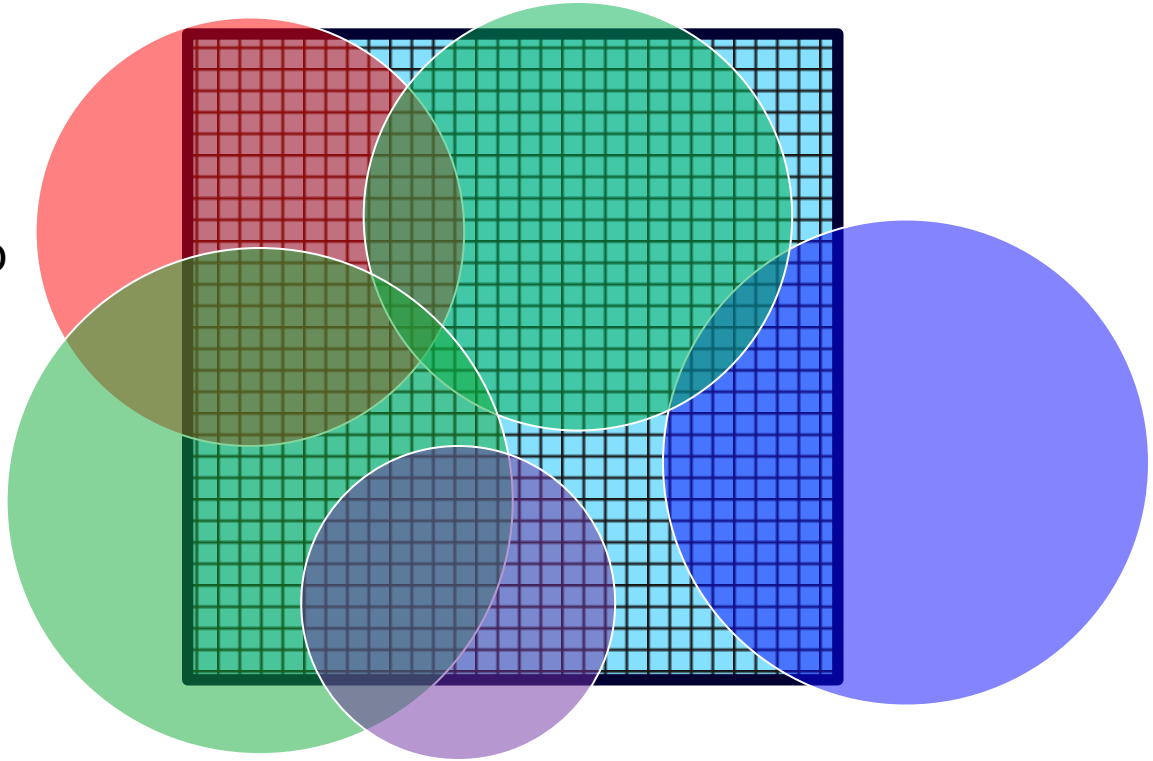
- Divide screen into tiles
- Fit asymmetric frustum around each tile

Tiled particle culling



Thread Group View

- numthreads[32,32,1]
- Culling 1024 particles in parallel
- Write visible indices to LDS



Per Tile Bitonic Sort

- Because each thread adds a visible particle
  - Particles are added to LDS in arbitrary order
  - Need to sort
- Only sorting particles in tile rather than global list

Tiled Rendering (1 thread = 1 pixel)

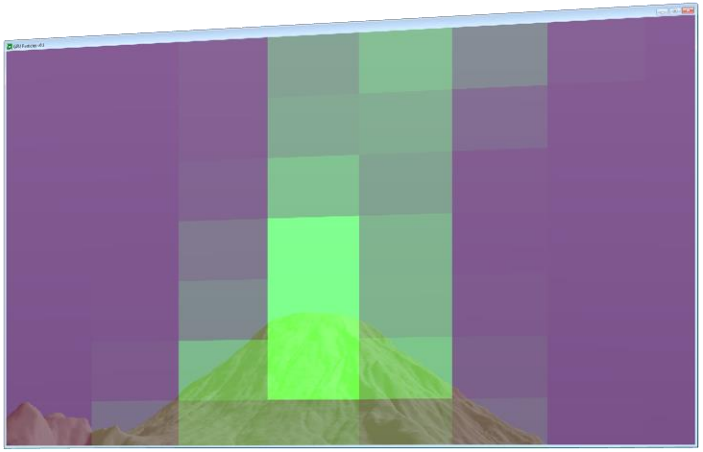
- Set accum color to float4(0, 0, 0, 0)
- For each particle in tile (back to front)
  - Evaluate particle contribution
    - Radius check
    - Texture lookup
    - Optional normal generation and lighting
  - Manually blend
    - color = (srcA x srcCol) + (invSrcA x destCol)
    - alpha = srcA + (invSrcA x destA)
- Write to screen size UAV

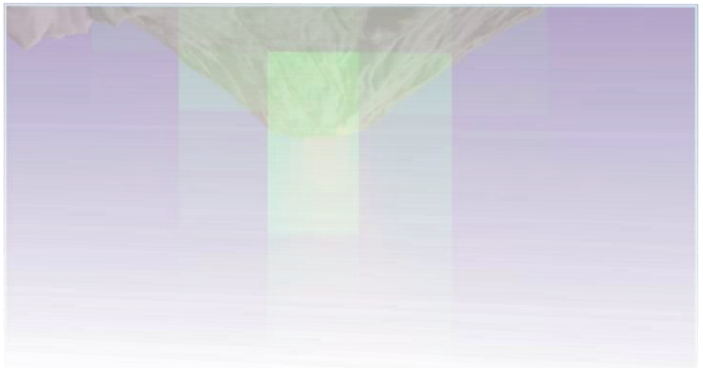
Tiled Rendering, improved!

- Set accum color to float4(0, 0, 0, 0)
- For each particle in tile (**front to back**)
  - Evaluate particle contribution
  - Manually blend [Bavoil08]
    - color = (invDestA x srcA x srcCol) + destCol
    - alpha = srcA + (invSrcA x destA)
  - **if (accum alpha > threshold)**
    accum alpha = 1 and bail
- Write to screen size UAV

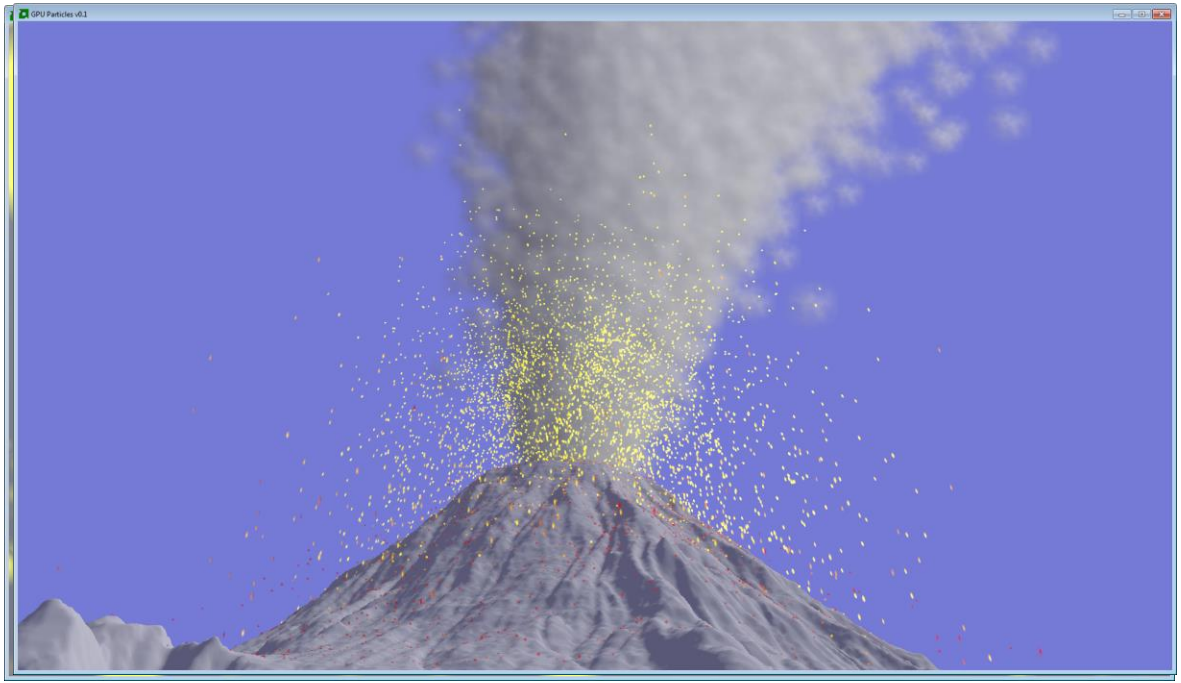
Coarse Culling

- Bin particles into 8x8
- UAV0 for indices
  - Array split into sections using offsets
- UAV1 for storing particle count per bin
  - 1 element per bin
  - Use InterlockedAdd() to bump counter
- For each alive particle
  - For each bin
    - Test particle against bin's frustum planes
    - Bump counter in UAV1 to get slot to write to
    - Add particle index to UAV0





Demo

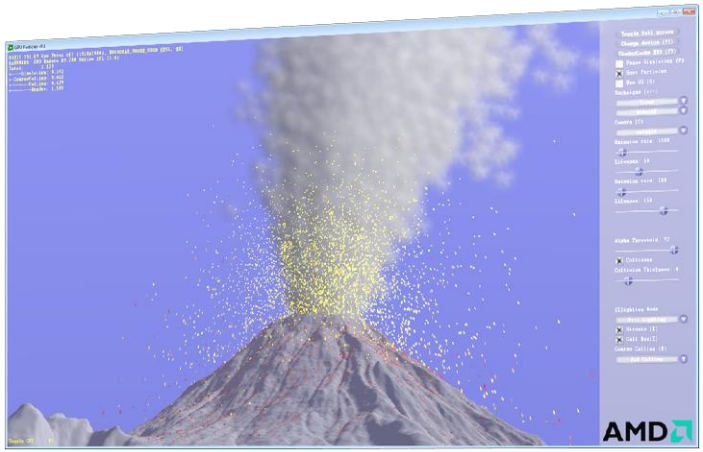


Demo with full source available soon

Performance Results

mode	frame time (ms)*
Rasterization	4.86
Tiled	3.15

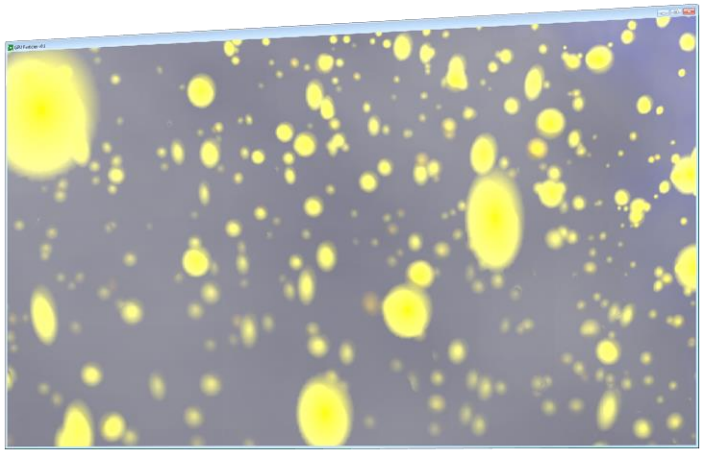
Breakdown	frame time (ms)*
Simulation	0.39
Coarse Culling	0.06
Tile Culling	0.43
Render	1.60



*AMD Radeon R9 290X @ 1080p

Performance Results

mode	frame time (ms)*
Rasterization	25.0
Tiled	5.1



*R9 290X @ 1080p

Conclusions

- Leverage compute for particle simulations
  - Depth buffer collisions
  - Bitonic sort for correct blending
- Tiled rendering
  - Faster than rasterization
  - Great for combating heavy overdraw
  - More predictable behavior
- Future work
  - Volume tracing
  - Add arbitrary geometry for OIT

Questions?

Demo with full source available soon

http://developer.amd.com/tools/graphics-development/amd-radeon-sdk/

References

- [Tchou11] Chris Tchou, “Halo Reach Effects Tech”, GDC 2011
- [Persson09] Emil Persson, http://www.humus.name/index.php?page=News&ID=266
- [Cantlay07] Iain Cantlay, “High-Speed, Off-Screen Particles”, GPU Gems 3 2007
- [Harada12] Takahiro Harada et al, “Forward+: Bringing Deferred Lighting to the Next Level”, Short Papers, Eurographics 2012
- [Bavoil08] Louis Bavoil et al, “Order Independent Transparency with Dual Depth Peeling”, 2008