A Practical Implementation for Scalable High-Performance Volumetric Effect Rendering From Low-Power Tablets to High-End Graphics Processors
Motivation
Evolution

“Soon computers will be fast” - Billy Zelsnack

Visual Quality

DX8  DX9  DX10  DX11
Subsystems Evolve Differently

Ripe for Better Volumetric Effects!
High-Level Overview
Volume Primitives
Displaced Sphere
Noise Displacement
Can we correctly render more than one?

✧ Can we “unify”?  
  – A correctly shadows B from light  
  – A correctly occludes B from eye  
  – Solution: populate intermediate voxel grid  

✧ Can we make it fast?  
  – Divide volume into pieces  
  – Process only occupied pieces  
  – Process only visible pieces  
  – Decouple filling from rendering  

✧ What are the quality/performance tradeoffs?  
✧ What use cases are practical?
Color

- Ambient Occlusion

- Color Ramp
Metavoxels

Voxels
A Simple Scene
Metavoxel Grid
Sparse Metavoxel Grid
Overview
Bin Particles

foreach metavoxel covered
foreach particle

Append Particle To Metavoxel’s Bin

Raymarch Metavoxel from Eye

foreach non-empty metavoxel

Fill Metavoxel

Propagate Light

Per-Metavoxel Particle Bin
Binning

Bounding Box

Bounding Sphere
Fill Metavoxel
Multiple Metavoxels
A Twist
Fill Volume Pixel/Fragment Shader

- Draw a “quad’ covering one metavoxel slice
  - Each pixel covers a voxel column
- Foreach voxel in column
  - Foreach particle
    - Blend particle properties if it covers voxel
- Foreach voxel in column
  - Light voxel by current light intensity
  - Attenuate light by voxel’s density
- Save final light amount to “light propagation” texture
Light Propagation Texture

- 2D
- Volume’s dimensions
- Including space for one-voxel borders
- Need for propagating lighting to next metavoxel (wrt light)
- Doubles as a shadow map
  - ish
  - One-voxel border artifacts
  - Need light/shadow value at “closest solid surface”
    - We don’t write to light propagation texture after light propagation hits a solid surface
    - Not perfect – leaves resolution mismatch gaps
Shadow Artifacts
Render The Volume
Raymarch Eye View
Compositing Metavoxel

✦ “Over” Blend
  – Blend PS RGB output over render target RGB
  – Blend PS alpha output over render target alpha
  – Blend weight given by PS alpha output
✦ “Under” Blend
  – Blend render target RGB over PS RGB output
  – Blend PS alpha output over render target alpha
  – Blend weight given by render target alpha
✦ Use over blending when rendering sorted back to front
✦ Use under blending when rendering sorted front to back
✦ Final result (from both cases) then blended with back buffer with over blending
Host System

- **Binning**
  - CPU processes each particle, appending to each covered metavoxel
- For each non-empty metavoxel
  - Fills constant buffers
  - Draw “fill metavoxel” pixel shader
  - Draw “raymarch from eye” pixel shader
- **Metavoxel array**
  - Fill many
  - Render many
  - Loop until filled and rendered all
  - Large enough array supports static use cases and update-every-n-frames
- We have not yet focused on optimizing host system
  - Not currently bottleneck
  - Could be if more particles used
Recognition

- Marc Fauconneau Dufresne
- Tomer Bar on
- Jon Kennedy
- Jefferson Montgomery
- Andrew Lauritzen
- Randall Rauwendaal
- Dave Houlton
- Dave Bookout
- Quentin Froemke

✦ Special thanks for management support!
  - Mike Burrows
  - Aaron Cody
Call To Action

See http://psychologyineducation.wordpress.com/2013/08/26/extrinsic-rewards-and-painting-fences/
Questions?