The Technology Behind the “Unreal Engine 4 Elemental demo”

Martin Mittring
Senior Graphics Architect
Martin.Mittring@EpicGames.com
Epic Games, Inc.
Overview

- Real-time Demo
- Graphical Features
  - Indirect Lighting
  - Shading
  - Post Processing
  - Particles
- Questions
Elemental demo

- GDC 2012 demo behind closed doors
- Demonstrate and drive development of Unreal® Engine 4
- NVIDIA® Kepler GK104 (GTX 680)
- Direct3D® 11
- No preprocessing
- Real-time
  - 30 fps
  - FXAA
  - 1080p at 90%
Real-Time Demo
Transition to Unreal Engine 4

- Shrink
  - Removed rarely used features
  - Unify renderer interface using Direct3D 11 as guidance
- Research
  - Samaritan demo (Direct3D 11, Deferred shading, Tessellation, …)
  - Elemental demo (Global Illumination, …)
- Expand
  - Bigger changes (Derived Data cache, new Editor UI, …)
This caught our attention:

Interactive Indirect Illumination and Ambient Occlusion Using Voxel Cone Tracing [Crassin11]
Indirect Lighting
Volume ray casting [Groeller05]
- Start with some start bias
- Content adaptive step size
- Lookup radiance and occlusion
- Accumulate light with occlusion
- Stop if occluded or far enough

Cone trace
- Mip level from local cone width
- Progressively increasing step size
- Like “Ray-tracing into a simplified scene”

**Diffuse GI:**
- Multiple directions depending on normal
- Opening angle from cone count

**Specular Reflections:**
- Direction from mirrored eye vector
- Opening angle from Specular Power

- Not as precise as ray-tracing but
  - Fractional geometry intersection
  - No noise
  - Level of detail
[Crassin11] can be further optimized / approximated
- Lower Voxel Resolution
- Gather instead of scatter in Voxel Lighting pass
- Adaptive sampling, sample reuse

Additional Benefits
- Shadowed IBL
- Shadowed area lights from emissive materials
Voxel Cone Tracing Challenges

- Stepping through thin walls
- Wide cones show artifacts but narrow cones are slow
- Mip maps need to be direction dependent
- Creating voxel data from triangle meshes
- Run-time memory management
- Efficient implementation on GPU hardware
- Sparse data structures
Sparse Voxel Octree

- Mapping function allows locally higher resolution
  - World 3D position <=> Index and local 3D position
- Fully maintained on GPU
- Index to access render stage specific data
  - Per node/leaf data
  - 2x2x2 voxel data (placed at octree node corners)
  - 6x 3x3x3 voxel data (like 2x2x2 with additional border)
Voxel Lighting Pipeline

Voxelization

- Triangle Mesh Instance Data Materials
- Albedo color Emissive color Opacity Normal

Lighting

- Light position Light color Light function Shadow maps

Filter and Finalize

- 6 Directions: HDR color Opacity

Diffuse Sampling

Specular Sampling

on demand each frame

Screen Space
Voxelization 1/2

- Create voxel geometry data in a Region
  - Input: Octree, Triangle Mesh, Instance Data, Materials, Region
  - Output: Octree, 2x2x2 material attributes, normal

- Region revoxelization
  - Geometry changes
  - Material changes
  - Resolution changes

- Optimized for few dynamic objects
  - Revoxelize on demand
  - Region keeps static voxel data separate
- Pixel shader pass using hardware rasterizer
  - One rasterization pass per axis (X, Y, Z) to avoid holes
  - Shader evaluates artist defined material
  - Output: fragment queue that is processed by following CS

- Compute Shader pass
  - Update octree data structures (in parallel)
  - Stores voxel data in leaves

- 2 Pass method
  - Better occupancy for second pass (2x2 quad)
  - Shader compile time (reuse CS)
Voxel Lighting

- **Compute shading and store Radiance**
  - Input: 2x2x2 material attributes, normal
  - Output: 2x2x2 HDR color and opacity

- **Accumulate Irradiance and Shade**
  - Add direct light with shadow maps
  - Add ambient color
  - Combine with albedo color
  - Add emissive color
Filter Voxels and Finalize

- Generate mip-maps, Create redundant border, Compress
  - Input: 2x2x2 HDR color, occlusion and normal
  - Output: HDR multiplier, 6 x 3x3x3 LDR color and occlusion

- Generate directionally dependent voxel
  - See view dependent voxels in [Gobbetti05]
  - At leaf level from voxel normal
  - At node level from same direction only

Directionally independent

Directionally dependent
The Cone Trace function

- float4 HDRColorAndOcclusion = SVOLookupLevel (float3 Pos, int Mip, float3 Direction)
  - Traverse tree to find node index and node local position
  - 3 tri-linear filtered lookups in 32 bit volume texture to get 3 directions
  - Weight results based on direction (Ambient Cube [McTaggart04])

- float4 HDRColorAndOcclusion = SVOConeTrace (float3 Pos, float3 Direction, float ConeAngle)
  - Calls SVOLookup() many times
  - Get all lighting coming from the given direction in a cone
Specular Sampling 1/2

- Per pixel local reflections
  - Cone angle from Specular Power
  - Single cone usually sufficient
  - Complex BRDF possible
- Adaptive for better performance
  - Specular brightness
  - Depth difference
  - Normal difference
Specular Sampling 2/2

- Up-sample pass using Dispatch()

```c
uint Pos = 0;
InterlockedAdd(State[STATE_Count], 1, Pos);
InterlockedMax(State[STATE_ThreadGroupCountX], (Pos+63)/64); // saves one pass
RWScratchColors[Pos] = (ThreadId.y << 16) | ThreadId.x;
```

- Scatter passes use DispatchIndirect()
Diffuse Sampling 1/2

- Similar to Final Gathering [Jensen02]
- Problem:
  - Few samples for good performance
  - Enough samples for quality (cone angle)
  - Well distributed over hemisphere to reduce error
  - Don’t want noise
  - Don’t want to blur over normal details
- Diffuse is mostly low frequency
- Coherency important for efficiency
Diffuse Sampling 2/2

- Non-interleaved processing of interleaved 3x3 pattern [Segovia06]
  - 9 well distributed cones in world space
  - Loop over 9 directions, then XY
  - Reject samples behind surface normal
  - Output non interleaved

- Compositing Pass:
  - Recombine non interleaved sub images
  - Weight by normal and depth
  - 5x5 filter to account for missing samples
  - Multiply with Albedo color
Voxel Lighting Examples 1/3

disabled

enabled
Voxel Lighting Examples 2/3
Voxel Lighting Examples 3/3

disabled

enabled
Shading
Advances in Real-Time Rendering in 3D Graphics and Games Course
Deferred Shading

- Classic deferred shading in PS (one forward pass)

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>D24</td>
<td>Depth</td>
</tr>
<tr>
<td>Stencil</td>
<td>S8</td>
<td>Stencil masking</td>
</tr>
<tr>
<td>SceneColor</td>
<td>R16G16B16A16f</td>
<td>RGB: Emissive and Light Accumulation</td>
</tr>
<tr>
<td>GBufferA</td>
<td>R10G10B10A2</td>
<td>RGB: WS Normal, A: Lighting Model</td>
</tr>
<tr>
<td>GBufferB</td>
<td>R8G8B8A8</td>
<td>RGB: Specular, A: Ambient Occlusion</td>
</tr>
<tr>
<td>GBufferC</td>
<td>R8G8B8A8</td>
<td>RGB: Diffuse, A: Opacity or Decal Mask</td>
</tr>
<tr>
<td>GBufferD</td>
<td>R8G8B8A8</td>
<td>R: Specular Power*, GBA: Subsurface Color</td>
</tr>
</tbody>
</table>

* not in alpha channel because of frame buffer blending limitations
New Specular Power Encoding

- Higher Specular Power for IBL
- More definition for common values
- Tweaked to give pixel sharp reflection on a far sphere of width 1000 pixel

OldEncode(x): \(\sqrt{x / 500}\)
OldDecode(x): \(x \times x \times 500\)

NewEncode(x): \((\log_2(\text{Value}) + 1) / 19\)
NewDecode(x): \(\exp(2(\text{Value} \times 19 - 1))\)
Gaussian Specular

- Gaussian Specular for less aliasing [McKesson12]
- Our empirical approximation

\[
\text{Dot} = \text{saturate}(\text{dot}(N, H)) \\
\text{Threshold} = 0.04 \\
\text{CosAngle} = \text{pow}(\text{Threshold}, 1 / \text{BlinnPhongSpecularPower}) \\
\text{NormAngle} = (\text{Dot} - 1) / (\text{CosAngle} - 1) \\
\text{LightSpecular} = \exp(- \text{NormAngle} \times \text{NormAngle}) \times \text{Lambert}
\]
Area Light Specular

- **Soft Sphere Area Light**
  
  \[
  \text{LightAreaAngle} = \text{atan}(\text{AreaLightFraction} / \text{LightDistance})
  \]
  
  \[
  \text{ACos} = \text{acos}(\text{CosAngle})
  \]
  
  \[
  \text{CosAngle} = \text{cos}(\text{ACos} + \text{LightAreaAngle})
  \]

- **Energy conserving (approximation)**

  \[
  \text{SpecularLighting} /= \text{pow}(\text{ACos} + \text{LightAreaAngle}, 2) \times 10
  \]
Specular Comparison

- Point Light
- Area Light
- Point Light + IBL
- Area Light + IBL
Post Processing
New post processing graph

- **Graph:**
  - Created each frame
  - No User Interface
  - Dependencies define execution order
  - RT on demand, ref. counting, lazy release

- **Node:**
  - Many types but fixed function
  - Multiple inputs and outputs
  - Defines output texture format
Screen Space Ambient Occlusion

- Classic SSAO [Kajalin09]
  - Ambient occlusion computed as post process
  - Only requires z buffer and 3d point samples
  - Few samples are permutated with small screen aligned pattern

- Our technique is based on 2d point samples
- Angle based similar to HBAO [Sainz08]
- Using GBuffer normal improves quality further
- Complements Voxel Lighting with high frequency details
SSAO sampling

- We use 6 sample pairs = 12 samples into half res z buffer
- 16 rotations with scale interleaved in 4x4 pattern
SSAO with per pixel normal

- Per pixel normal further restricts angle

A) Given: z buffer in the sample direction
B) Get equi-distant z values from samples
C) AO (so far) = min((angle_left+angle_right)/180, 1)
D) Clamp against per pixel normal
E) AO (per pixel normal) = (angle_left+angle_right)/180

AO ≈ 1 - saturate(dot(VecA, Normal)/length(VecA))
SSAO Example

SSAO (Depth only)  SSAO with per pixel Normal
SSAO Example Close-up

SSAO (Depth only)  SSAO with per pixel Normal
Lens flares are out of focus reflections on the camera lens.

Image based method:
- Threshold and blur bright image parts
- Scale and mirror image multiple times
- Soft mask screen borders

Lens/Bokeh Blur (for out of focus):
- Render a textured sprite for each very bright low res pixel
- Ideally for each lens reflection with different radius
Image Based Lens Flares 2/2

- Source Image with Bloom
- IB Lens Flares (without Lens Blur)
- Lens Blur Sprite Image
- IB Lens Flares (with Lens Blur)
IB Lens Flares Examples

Emissive (Sun)  Emissive (Fire)  Reflections
HDR Histogram [Scheuermann07]

- 64 Buckets, logarithmic, no atomics

- Pass 1: Generate screen local histograms (CS) in parallel

  Clear groupshared histograms float[64][16]
  Sync
  Accumulate histograms in parallel
  Sync
  Accumulate many Histograms to one float4[16]
  Output one Histogram per line in 16 texels

- Pass 2: Combine all lines into one

- 64 Buckets are stored in 16 ARGB
Eye Adaptation

- Compute average brightness from Histogram (blue line)
  - Consider only bright areas (e.g. >90%)
  - Reject few very bright areas (very bright emissive, e.g. >98%)

- Compute single multiplier for whole view port
  - Smoothly blend with last frame average (white bar)
  - Bound in user specified region (green)

- Apply in tone mapper (white curve)
  - Read result in tone mapping VS
  - Pass to PS as interpolator
Particles
GPU accelerated particles

- **CPU**
  - Spawn particles (arbitrarily complex logic)
  - Memory management in fixed size buffers (unit: 16 particles)
  - Emitter management (Index buffer, draw call sorting)

- **GPU**
  - Motion from Newtonian mechanics (fixed function)
  - Lighting from non directional volume cascades (3D lookups)
  - GPU Radix depth sort if required [Merrill11] [Satish09]
  - Rendering
  - Additional forces from Vector Fields* (3D lookup)
  - Particle Curves to modulate particle attributes* (1D lookup)

* See next slides
Particle Attributes

- State-full simulation [Lutz04]
  - Allows more complex animations
- Stored over particle lifetime

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>R32G32B32A32f</td>
<td>World Space Position*, Time Phase</td>
</tr>
<tr>
<td>Velocity</td>
<td>R16G16B16A16f</td>
<td>World Space Velocity, Time Scale</td>
</tr>
<tr>
<td>Render Attrib.</td>
<td>R8G8B8A8</td>
<td>Size, Rotation</td>
</tr>
<tr>
<td>Simulation Attrib.</td>
<td>R8G8B8A8</td>
<td>Drag, Vector Field Scale, Random Seed</td>
</tr>
</tbody>
</table>

- Particle Curves: Time Phase and Scale
Particle Curves

- **Concept**
  - 1D Function of time
  - Artist driven (arbitrary complex)

- **Implementation**

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>R8G8B8A8</td>
<td>Modulate simulation or render attributes</td>
</tr>
</tbody>
</table>

- Filtered texture lookup (Piecewise linear, equidistant)
- Sample count depends on source curve (error threshold)
- Many 1D curves packed into single 2D texture
Particle Vector Fields

- Per volume attributes
  - World to Volume matrix
  - Force scale (accumulate)
  - Velocity scale (weighted blend)
  - Affect all particle systems globally or a single system

- Per volume element attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>OffsetVector</td>
<td>R16G16B16A16f</td>
<td>Force or Velocity Delta</td>
</tr>
</tbody>
</table>

- Can be imported from Maya
- Unified interface for many kind of complex motions
Volumetric direct and indirect lighting
> 1 Million Particles
Thanks

- NVIDIA, AMD
- Special thanks to Cyril Crassin and Evan Hart from NVIDIA
- Epic
  - Rendering team: Daniel Wright, Andrew Scheidecker, Nick Penwarden
  - Everyone that contributed to Unreal Engine 4
We Are Hiring
Epic Games is hiring

- Work on leading game engine
  - Unreal Engine 3
  - Upcoming: Unreal Engine 4
- Ship successful games
  - Gear Of War 1-3, Infinity Blade 1-2, ...
  - Upcoming: Fortnite, Infinity Blade: Dungeons
- Target many platforms:
  - Xbox 360, PlayStation 3, PC DX9/11, Mobile, Mac, next gen consoles
- Main office in North Carolina

www.EpicGames.com/jobs
References 1/2

- [Crassin11] Interactive Indirect Illumination and Ambient Occlusion Using Voxel Cone Tracing Interactive Indirect Illumination Using Voxel Cone Tracing, Sep 2011

- [Kawase04] Practical Implementation of High Dynamic Range Rendering

- [Segovia06] Non-interleaved Deferred Shading of Interleaved Sample Patterns
  http://liris.cnrs.fr/Documents/Liris-2476.pdf

- [Kajalin09] Screen Space Ambient Occlusion ShaderX7 - Advanced Rendering Techniques

- [Sainz08] Image-Space Horizon-Based Ambient Occlusion

- [Tabellion08] Practical Global Illumination with Irradiance Caching


  http://hal.inria.fr/docs/00/58/99/40/PDF/article.pdf

- [McKesson12] Gaussian Specular from Learning Modern 3D Graphics Programming
  http://www.arcsynthesis.org/gltut/Illumination/Tut11%20Gaussian.html

- [Scheuermann07] Efficient Histogram Generation Using Scattering on GPUs
References 2/2

  http://www.crs4.it/vic/cgi-bin/bib-page.cgi?id='Gobbetti:2005:FV'

- [Mittring11] The Technology Behind the DirectX 11 Unreal Engine "Samaritan" Demo

- [Lutz04] Building a Million-Particle System
  http://www.gamasutra.com/view/feature/130535/building_a_millionparticle_system.php?page=1

- [Lutz11] Everything about Particle Effects
  http://www.2ld.de/gdc2007

- [McTaggart04] Half-Life®2 / Valve Source™ Shading

- [Merrill11] High Performance and Scalable Radix Sorting
  Parallel Processing Letters, vol. 21, no. 2, 2011, pp. 245-272
  https://sites.google.com/site/duanemerrill/awards-publications

- [Satish09] Designing Efficient Sorting Algorithms for Manycore GPUs
  IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, 2009

  http://www.cs.princeton.edu/courses/archive/fall02/cs526/papers/course43sig02.pdf

Questions?
Bonus slides
Goal: Large, high quality, efficient

Down sample:

\[
\begin{align*}
A &= \text{downsample2}(\text{FullRes}) \\
B &= \text{downsample2}(A) \\
C &= \text{downsample2}(B) \\
D &= \text{downsample2}(C) \\
E &= \text{downsample2}(D)
\end{align*}
\]

Blur during downsample avoids aliasing

Without blur (1 sample):

With blur (4 samples):
Recombine (with increasing resolution):

\[
\begin{align*}
E' &= \text{blur}(E, b5) \\
D' &= \text{blur}(D, b4) + E' \\
C' &= \text{blur}(C, b3) + D' \\
B' &= \text{blur}(B, b2) + C' \\
A' &= \text{blur}(A, b1) + B'
\end{align*}
\]

Blurring while up sampling

- Improves quality
- Barely affects blur radius

\[
\text{blur} (\text{blur}(X, a), b) \approx \text{blur}(X, \text{max}(a, b))
\]

Combine with dirt texture
Bloom Example

Bloom with single Gaussian

Bloom with 5 Gaussians and Dirt
Smart blur:
- Average of 5 pixels
- Weighted by normal
- Weighted by depth difference

Applications:
- Reduce aliasing of specular materials (noticeable in motion)
- Reduce high frequency dither artifacts in Ambient Occlusion
- Can increase performance of with IBL or Voxel Lighting
- Using Gather() where possible (Depth, AO)
- Output: SpecularPower, Normal, AmbientOcclusion
- Reduce Specular Power [Toksvig05] [Bruneton11]

\[
L = \text{saturate}(\text{length}(\text{SumNormal}) \times 1.002) \\
\text{SpecularPower} \times= \frac{L}{L + \text{SpecularPower} \times (1 - L)}
\]
Advances in Real-Time Rendering in 3D Graphics and Games Course without GBuffer Blur with GBuffer Blur
Auxiliary to the graph

- Post Process Volume:
  - Linearly blends Post process properties
  - Priority depending on camera position
  - Soft transitions with Blend Radius
  - Weight can be controlled remotely

- Render Target Pool:
  - Allocation on demand, reference counting
  - Deferred release
  - Tools to look at intermediate Buffers
Voxel Lighting Examples 3/5
Voxel Lighting Examples 5/5