# SIGGRAPH 2012

The **39th** International **Conference** and **Exhibition** on **Computer Graphics** and **Interactive Techniques**

# The Technology Behind the "Unreal Engine 4 Elemental demo"

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

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

UNREAL ENGINE

SIGGRAPH2012

EPIC GAMES

# 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



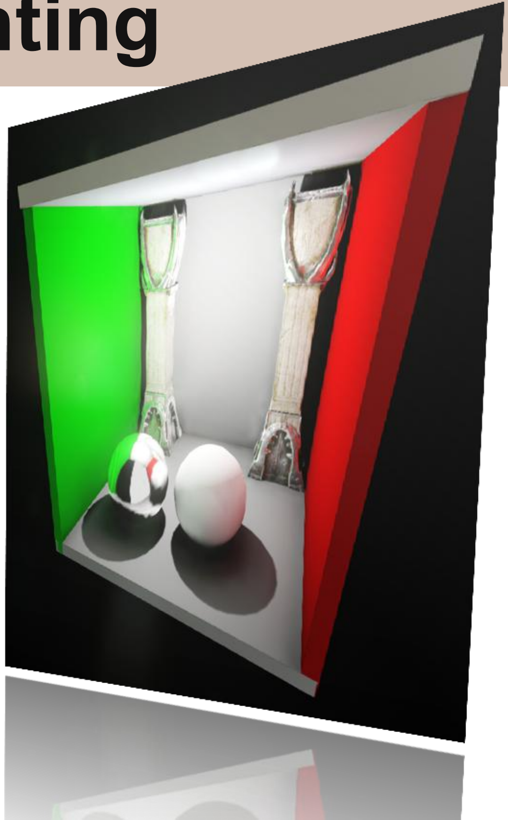
- Shrink
  - Removed rarely used features
  - Unify renderer interface using Direct3D 11 as guidance
- Research
  - Samaritan demo (Direct3D 11, Deferred shading, Tessellation, …)
  - Elemental demo (Global Illumination, …)
- Expand
  - Bigger changes (Derived Data cache, new Editor UI, …)

- This caught our attention:


Interactive Indirect Illumination and Ambient Occlusion Using Voxel Cone Tracing [Crassin11]

# Indirect Lighting

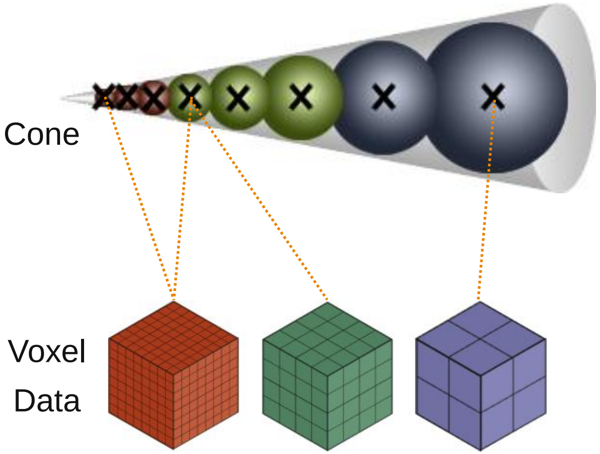




# Voxel Cone Tracing Concept



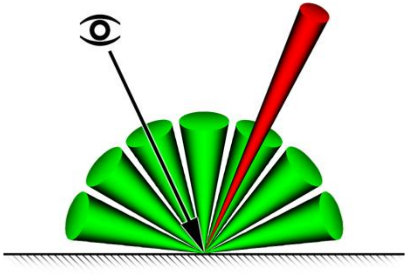
- Volume ray casting [Groeller05]
  - Start with some start bias
  - Content adaptive step size
  - Lookup radiance and occlusion
  - Accumulate light with occlusion
  - Stop if occluded or far enough
- Cone trace
  - Mip level from local cone width
  - Progressively increasing step size



# Using Voxel Cone Tracing for GI 1/2



- 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



# Using Voxel Cone Tracing for GI 2/2

- [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

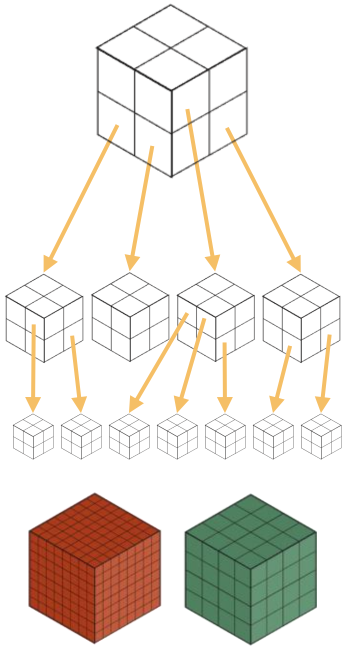


- Stepping through thin walls
- Wide cones show artifacts but narrow cones are slow
- Mip maps need to be direction dependent
- Creating voxel data from triangle meshes
- Run-time memory management
- Efficient implementation on GPU hardware
- Sparse data structures

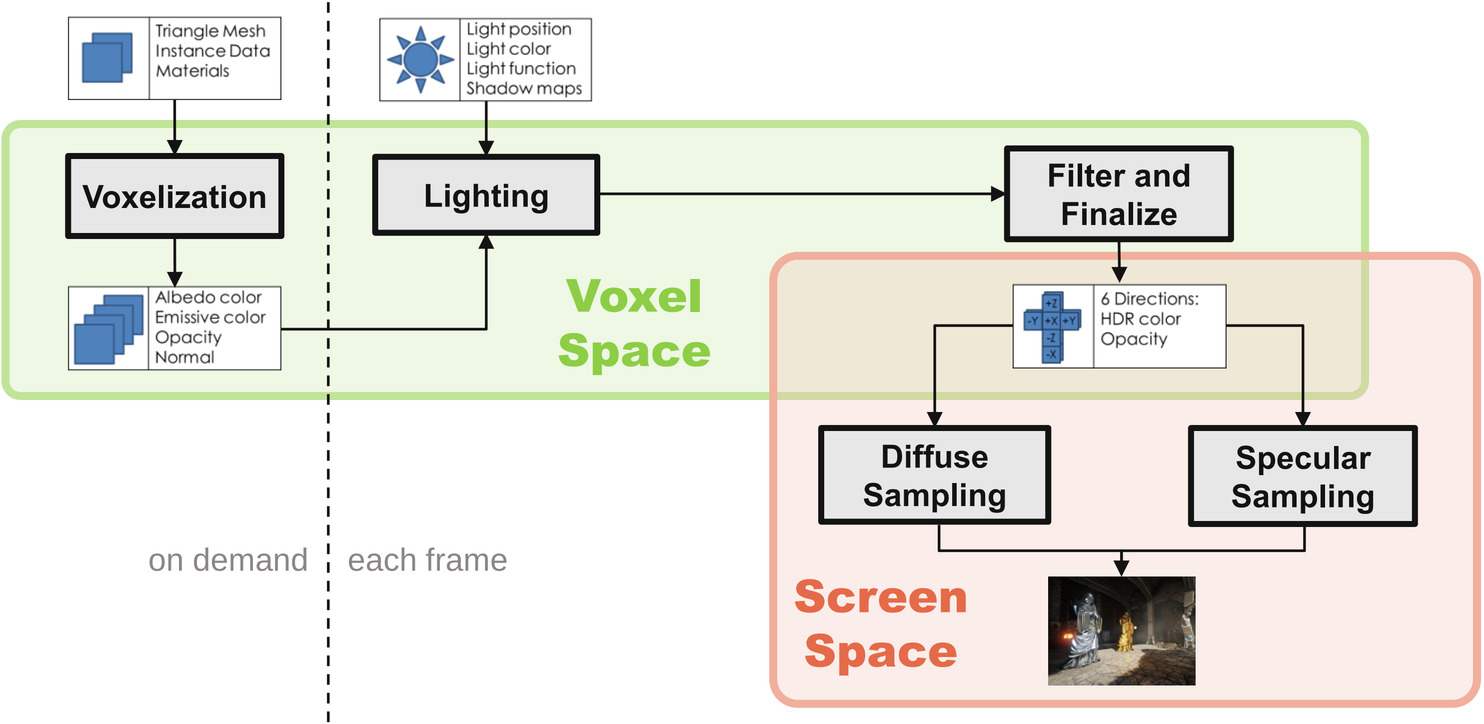
# Sparse Voxel Octree



- 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

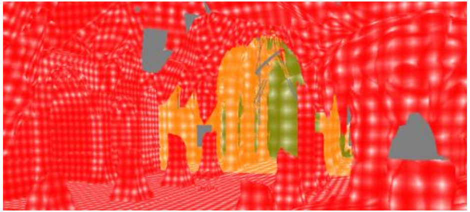


# Voxelization 1/2



- 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

3D Scene

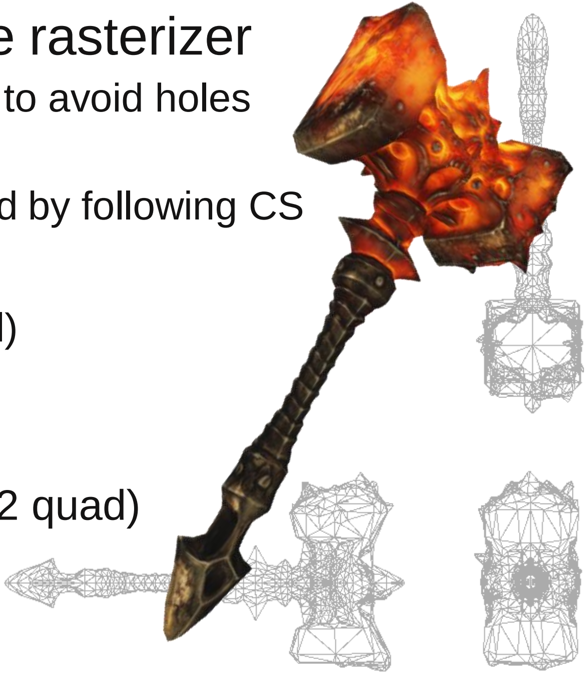


Voxel resolution as color

# Voxelization 2/2



- 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)



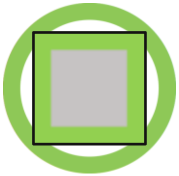
# Voxel Lighting



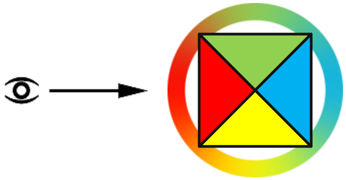
- Compute shading and store Radiance
  - Input: 2x2x2 material attributes, normal
  - Output: 2x2x2 HDR color and opacity
- Accumulate Irradiance and Shade
  - Add direct light with shadow maps
  - Add ambient color
  - Combine with albedo color
  - Add emissive color


# 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

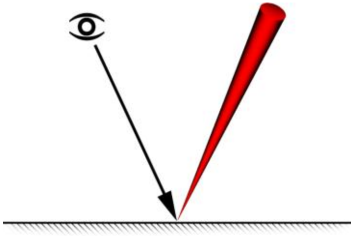
# The Cone Trace function

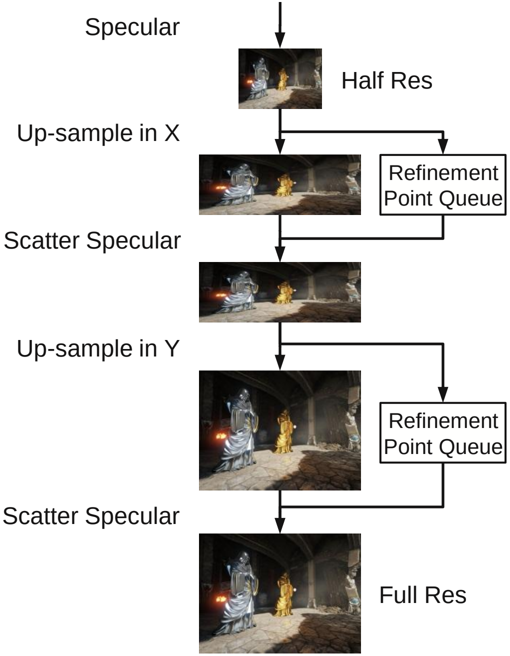


- `float4 HDRColorAndOcclusion = SVOLookupLevel (float3 Pos, int Mip, float3 Direction)`
  - Traverse tree to find node index and node local position
  - 3 tri-linear filtered lookups in 32 bit volume texture to get 3 directions
  - Weight results based on direction (Ambient Cube [McTaggart04])
- `float4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



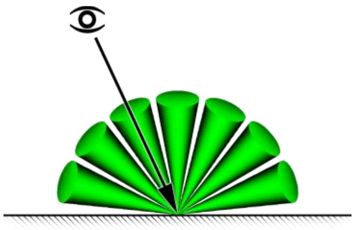

- Up-sample pass using Dispatch()

```
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

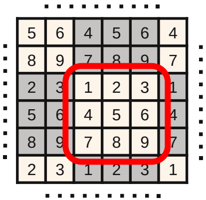
- Similar to Final Gathering [Jensen02]
- Problem:
  - Few samples for good performance
  - Enough samples for quality (cone angle)
  - Well distributed over hemisphere to reduce error
  - Don't want noise
  - Don't want to blur over normal details
- Diffuse is mostly low frequency
- Coherency important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



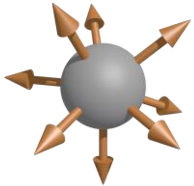
# Diffuse Sampling 2/2



- 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




Interleaved



Samples on Sphere


Non Interleaved

# Voxel Lighting Examples 1/3

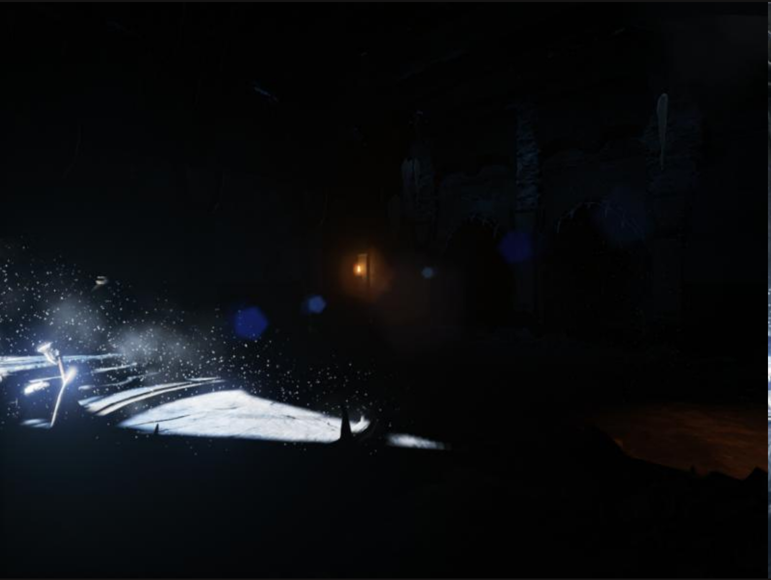


disabled enabled

# Voxel Lighting Examples 2/3

# Voxel Lighting Examples 3/3

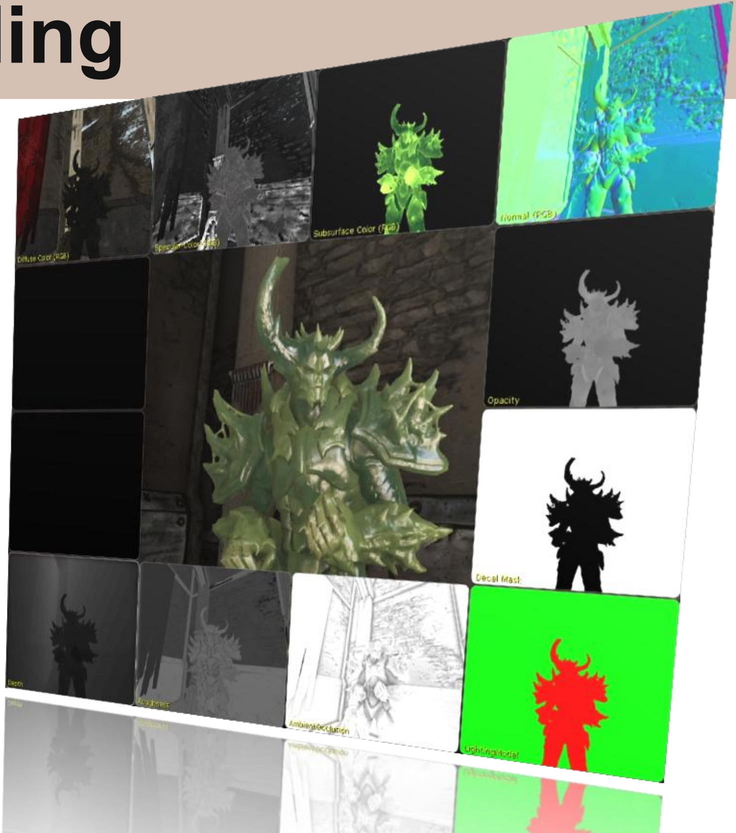


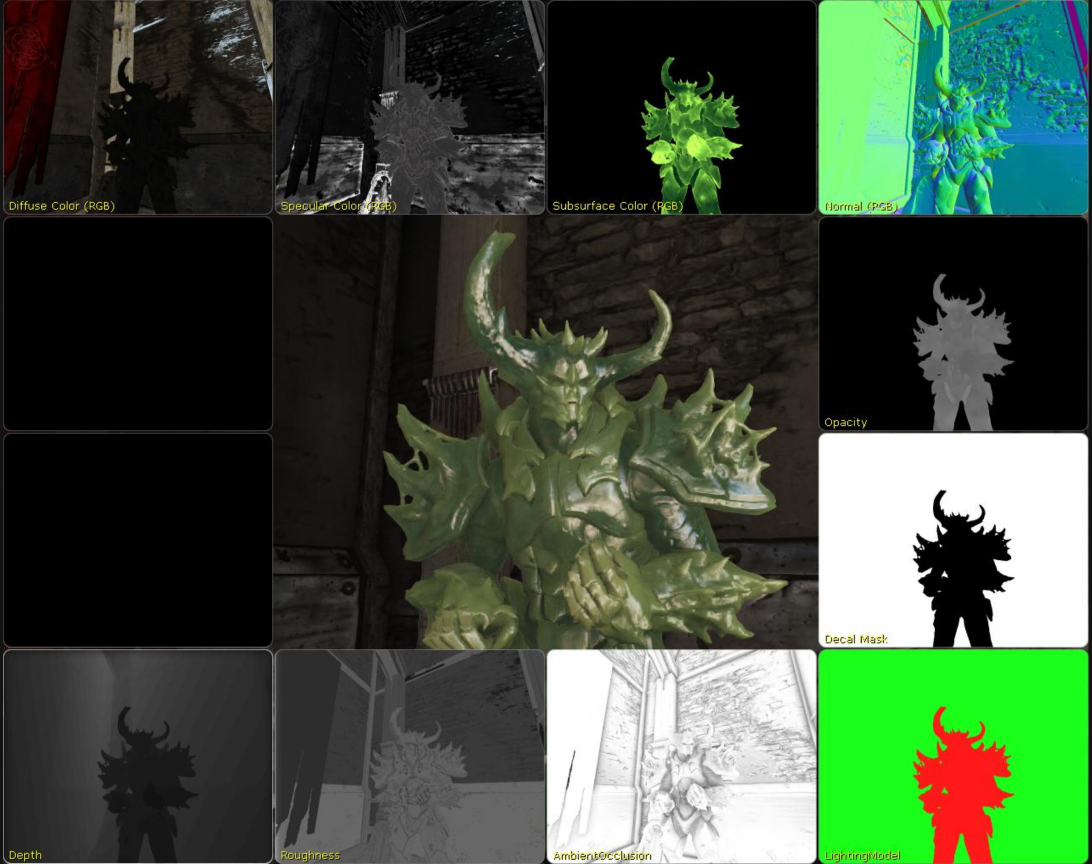


disabled

enabled

# Shading


Diffuse Color (RGB)
Subsurface Color (RGB)
Normal (RGB)
Opacity
Decal Mask
Depth
Roughness
AmbientOcclusion
LightingModel


Diffuse Color (RGB)
Specular Color (RGB)
Subsurface Color (RGB)
Normal (RGB)
Opacity
Decal Mask
Depth
Roughness
AmbientOcclusion
LightingModel

# Deferred Shading



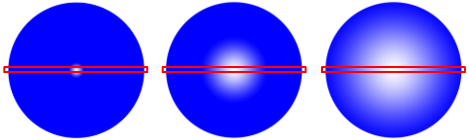
- Classic deferred shading in PS (one forward pass)

| Name | Format | Usage |
|---|---|---|
| Depth | D24 | Depth |
| Stencil | S8 | Stencil masking |
| SceneColor | R16G16B16A16f | RGB: Emissive and Light Accumulation |
| GBufferA | R10G10B10A2 | RGB: WS Normal, A: Lighting Model |
| GBufferB | R8G8B8A8 | RGB: Specular, A: Ambient Occlusion |
| GBufferC | R8G8B8A8 | RGB: Diffuse, A:Opacity or Decal Mask |
| GBufferD | R8G8B8A8 | R: Specular Power*, GBA: Subsurface Color |

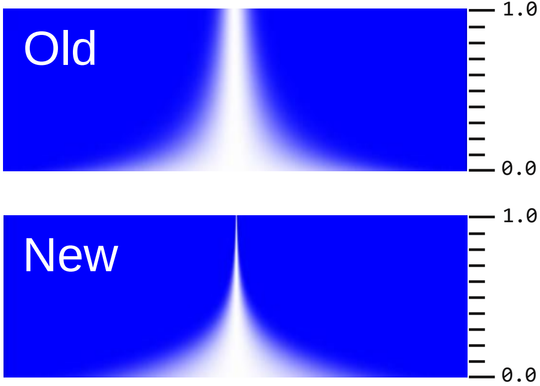
* not in alpha channel because of frame buffer blending limitations

# 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 * x * 500
```

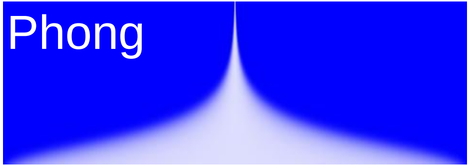


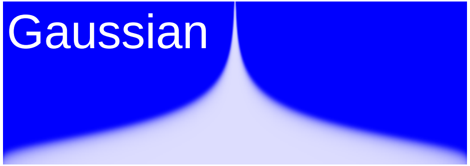

```
NewEncode(x): (log2(Value) + 1) / 19

NewDecode(x): exp2(Value * 19 - 1)
```

# Gaussian Specular

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

```
Dot = saturate(dot(N, H))
Threshold = 0.04
CosAngle = pow(Threshold, 1 / BlinnPhongSpecularPower)
NormAngle = (Dot - 1) / (CosAngle – 1)
LightSpecular = exp(- NormAngle * NormAngle) * Lambert
```

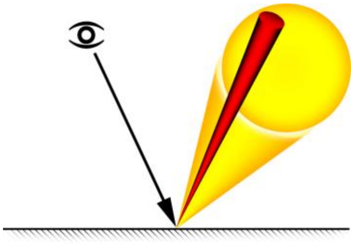


# Area Light Specular

- Soft Sphere Area Light

```
LightAreaAngle = atan(AreaLightFraction / LightDistance)
ACos = acos(CosAngle)
CosAngle = cos(ACos + LightAreaAngle)
```



- Energy conserving (approximation)

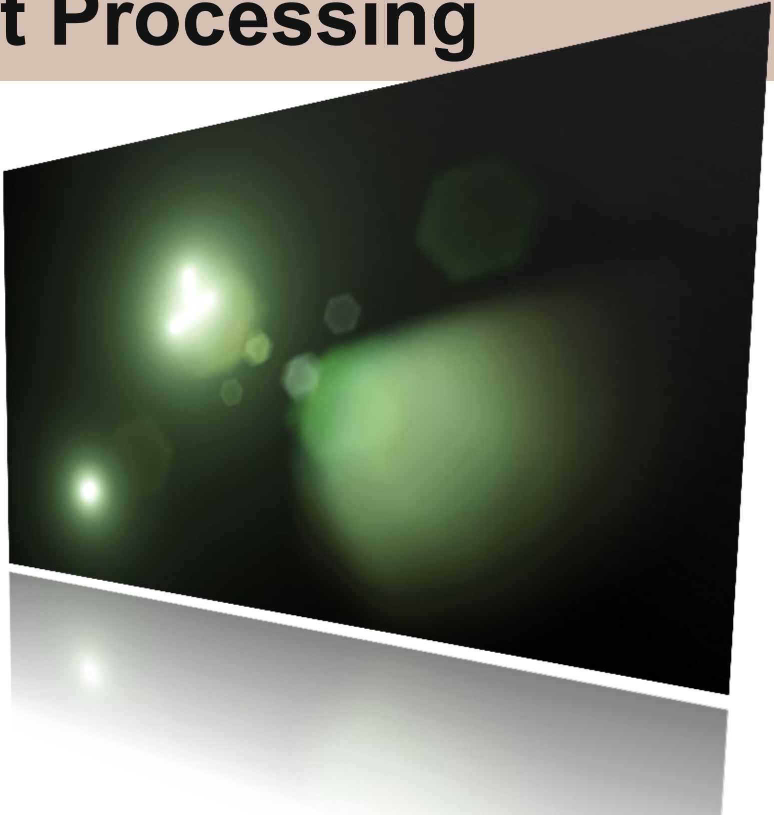
```
SpecularLighting /= pow(ACos + LightAreaAngle, 2) * 10
```

# Specular Comparison



# Post Processing

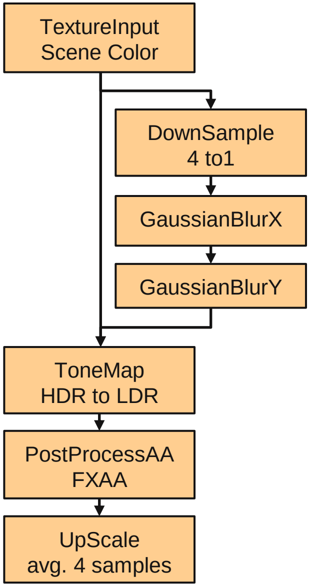




# New post processing graph



- 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




Example Graph

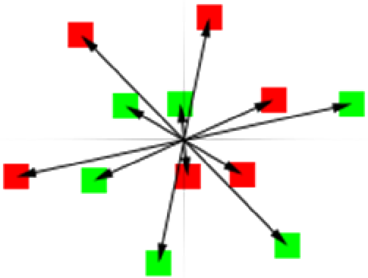
# Screen Space Ambient Occlusion



- 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

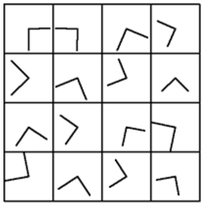
# SSAO sampling

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



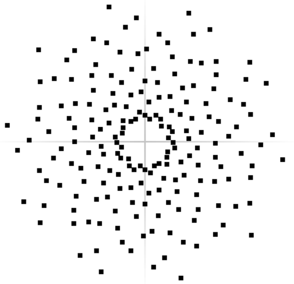
6 Samples pairs

+



16 rotations with scale in 4x4 pixel block

=



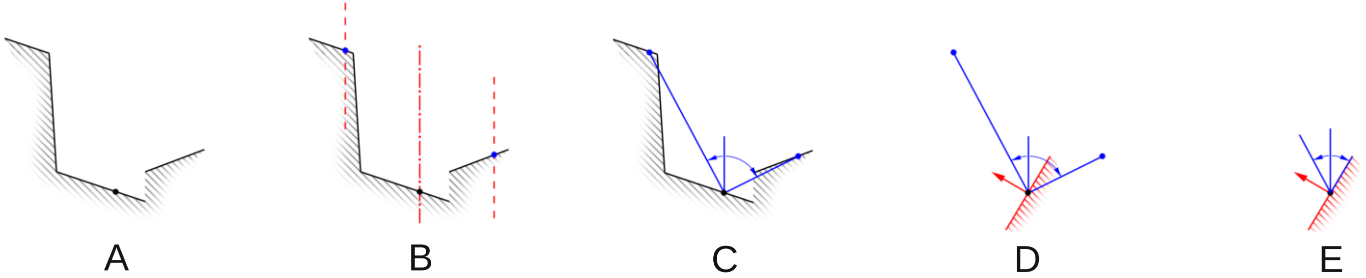
192 samples in 4x4 pixel block

# SSAO with per pixel normal



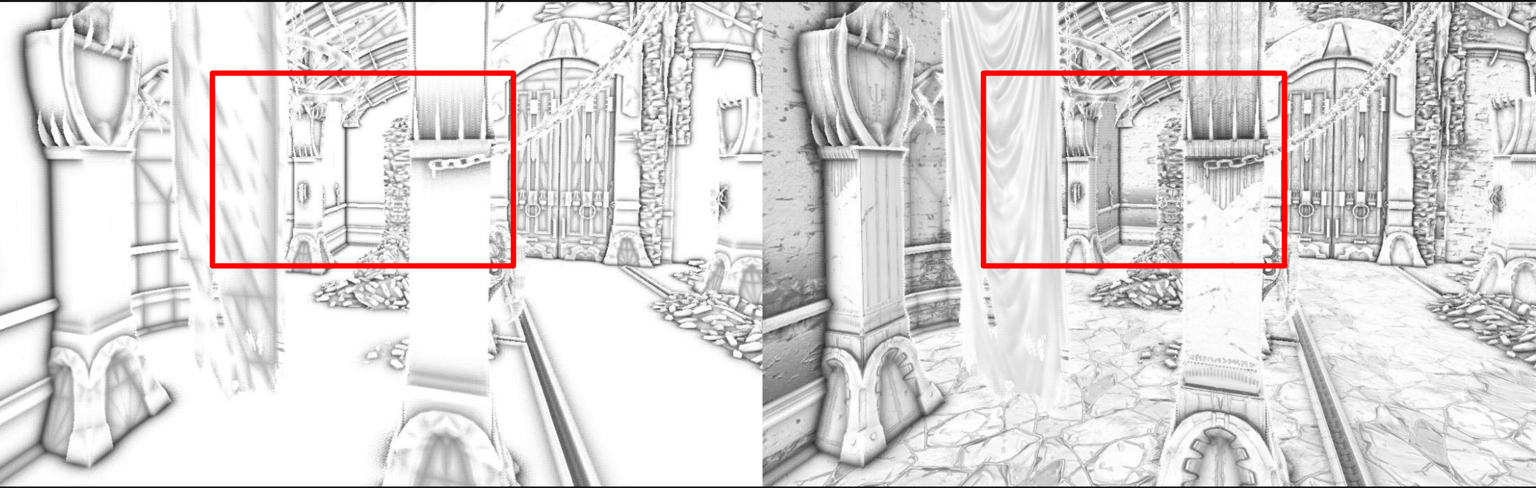
- Per pixel normal further restricts angle

```
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



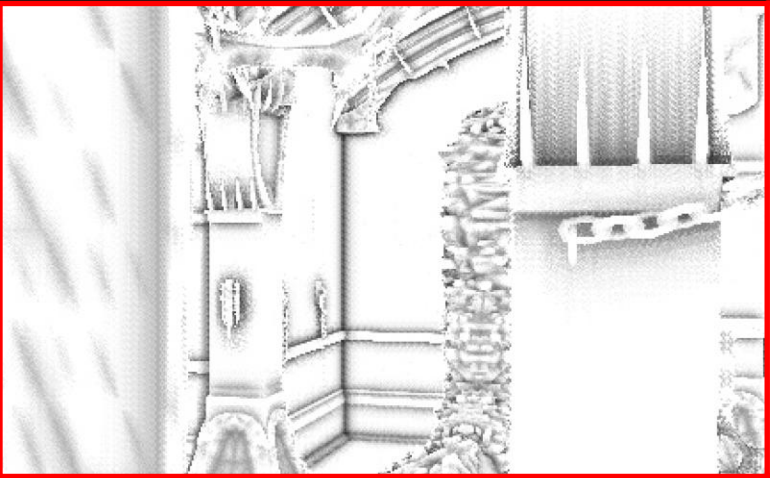


SSAO (Depth only)

SSAO with per pixel Normal

# SSAO Example Close-up





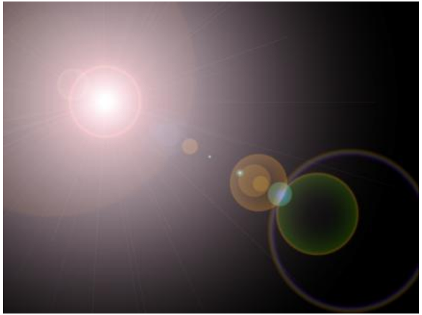
SSAO (Depth only)

SSAO with per pixel Normal

# Image Based Lens Flares 1/2



- 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

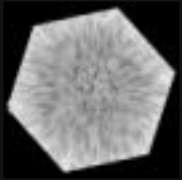


# Image Based Lens Flares 2/2




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)

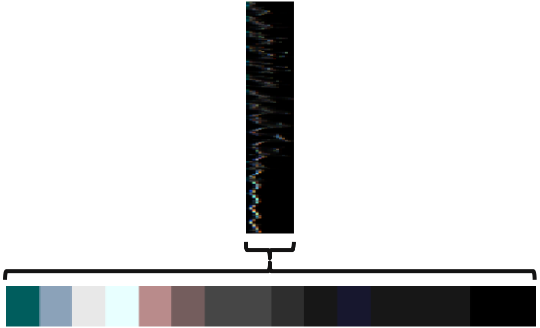
Emissive (Fire)

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
```

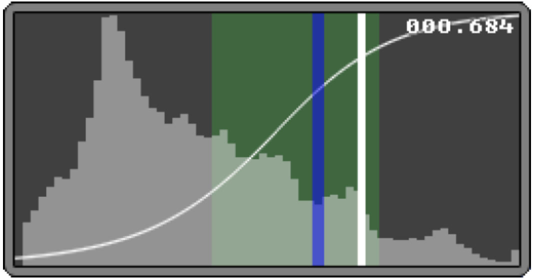


- Pass 2: Combine all lines into one
- 64 Buckets are stored in 16 ARGB

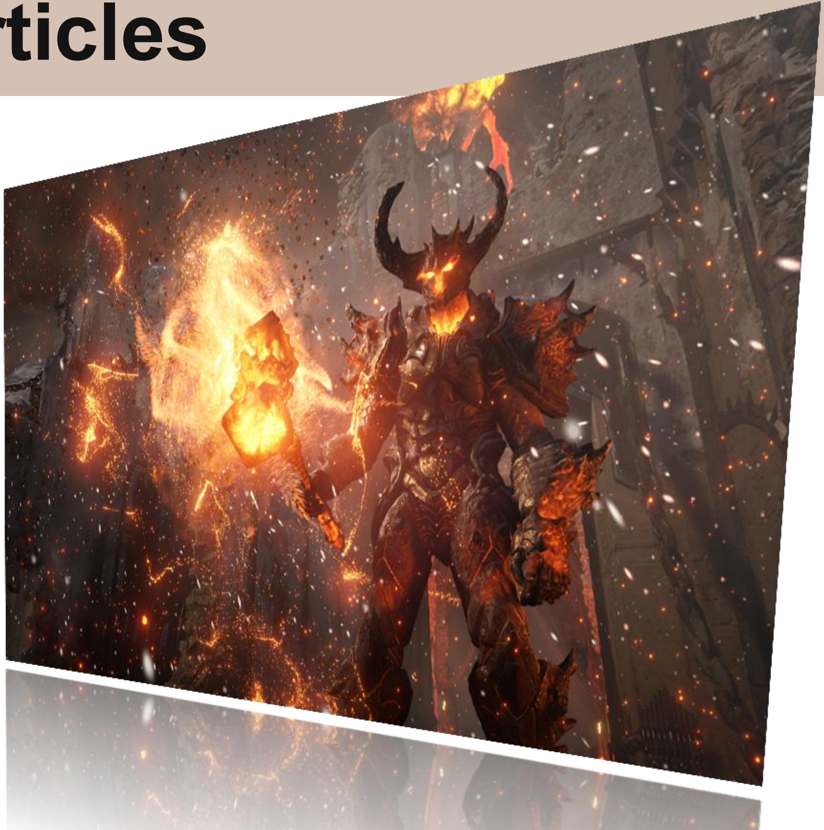
# Eye Adaptation



- 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



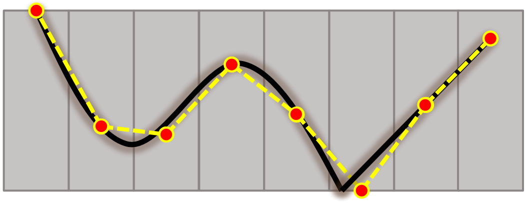
- State-full simulation [Lutz04]
  - Allows more complex animations
- Stored over particle lifetime

| Name | Format | Usage |
|---|---|---|
| Position | R32G32B32A32f | World Space Position*, Time Phase |
| Velocity | R16G16B16A16f | World Space Velocity, Time Scale |
| Render Attrib. | R8G8B8A8 | Size, Rotation |
| Simulation Attrib. | R8G8B8A8 | Drag, Vector Field Scale, Random Seed |

- Particle Curves: Time Phase and Scale

# Particle Curves

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

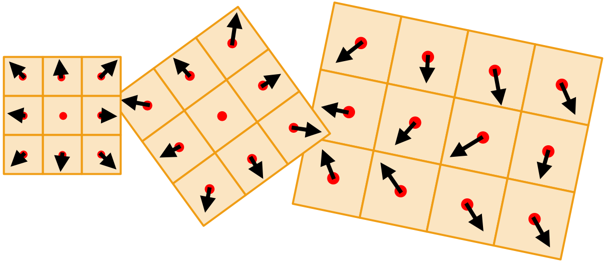


- Implementation

| Name | Format | Usage |
| --- | --- | --- |
| Attributes | R8G8B8A8 | Modulate simulation or render attributes |

  - 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

| Name | Format | Usage |
|---|---|---|
| OffsetVector | R16G16B16A16f | Force or Velocity Delta |

- Can be imported from Maya
- Unified interface for many kind of complex motions

# Shadow receiving Translucency

# Volumetric direct and indirect lighting

# > 1 Million Particles

# Thanks

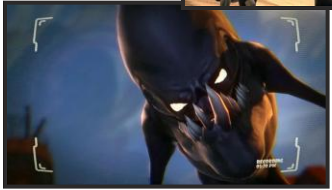
- NVIDIA, AMD
- Special thanks to Cyril Crassin and Evan Hart from NVIDIA
- Epic
  - Rendering team: Daniel Wright, Andrew Scheidecker, Nick Penwarden
  - Everyone that contributed to Unreal Engine 4

EPIC GAMES

# 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
  http://research.nvidia.com/sites/default/files/publications/GIVoxels-pg2011-authors.pdf
- [Kawase04] Practical Implementation of High Dynamic Range Rendering
  http://www.daionet.gr.jp/~masa/archives/GDC2004/GDC2004_PIoHDRR_SHORT_EN.ppt
- [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
  http://www.nvidia.com/object/siggraph-2008-HBAO.html
- [Tabellion08] Practical Global Illumination with Irradiance Caching
  http://cgg.mff.cuni.cz/~jaroslav/papers/2008-irradiance_caching_class/10-EricSlides.pdf
- [Toksvig05] Mipmapping normal maps. Journal of Graphics Tools 10, 3, 65–71
  ftp://download.nvidia.com/developer/Papers/Mipmapping_Normal_Maps.pdf
- [Bruneton11] A Survey of Non-linear Pre-filtering Methods for Efficient and Accurate Surface Shading
  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
  http://arcsynthesis.org/gltut/Illumination/Tut11%20On%20Performance.html
- [Scheuermann07] Efficient Histogram Generation Using Scattering on GPUs
  http://developer.amd.com/gpu_assets/GPUHistogramGeneration_I3D07.pdf

# References 2/2



- [Gobbetti05] Far Voxels: A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms
  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
  http://udn.epicgames.com/Three/rsrc/Three/DirectX11Rendering/MartinM_GDC11_DX11_presentation.pdf
- [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
  http://www2.ati.com/developer/gdc/D3DTutorial10_Half-Life2_Shading.pdf
- [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://mgarland.org/files/papers/gpusort-ipdps09.pdf
- [Jensen02] A Practical Guide to Global Illumination using Photon Mapping Siggraph 2002
  http://www.cs.princeton.edu/courses/archive/fall02/cs526/papers/course43sig02.pdf
- [Groeller05] A Simple and Flexible Volume Rendering Framework for Graphics-Hardware based Raycasting
  http://cumbia.informatik.uni-stuttgart.de/ger/research/pub/pub2005/vg2005-stegmaier.pdf

# Questions?

# Bonus slides



Bonus
Slides

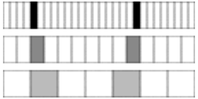
# Bloom 1/2 [Kawase04]

- Goal: Large, high quality, efficient
- Down sample:

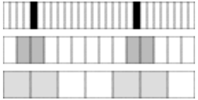
```
A = downsample2(FullRes)
B = downsample2(A)
C = downsample2(B)
D = downsample2(C)
E = downsample2(D)
```

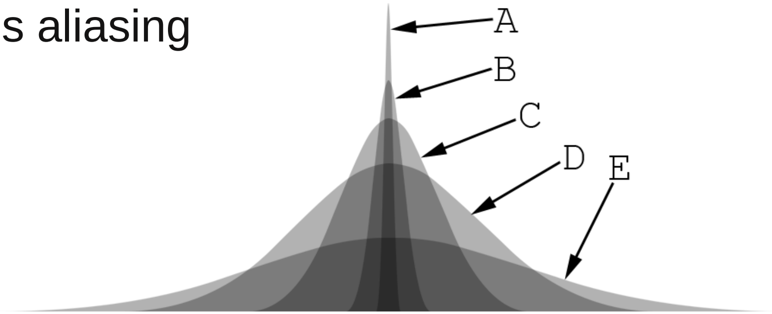
- Blur during downsample avoids aliasing

Without blur (1sample):



With blur (4 samples):





# Bloom 2/2



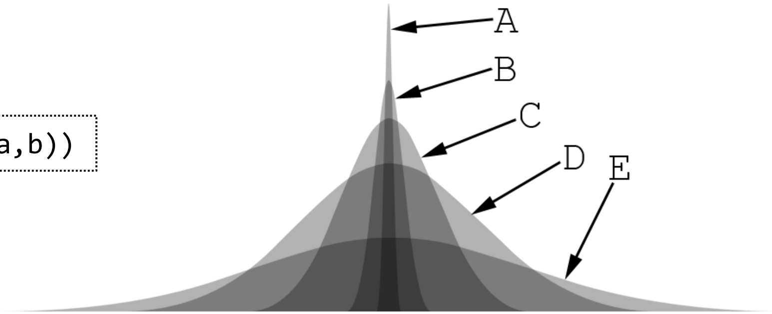
- Recombine (with increasing resolution):

```
E’= blur(E,b5)
D’= blur(D,b4)+E’
C’= blur(C,b3)+D’
B’= blur(B,b2)+C’
A’= blur(A,b1)+B’
```

- Blurring while up sampling
  - Improves quality
  - Barely affects blur radius

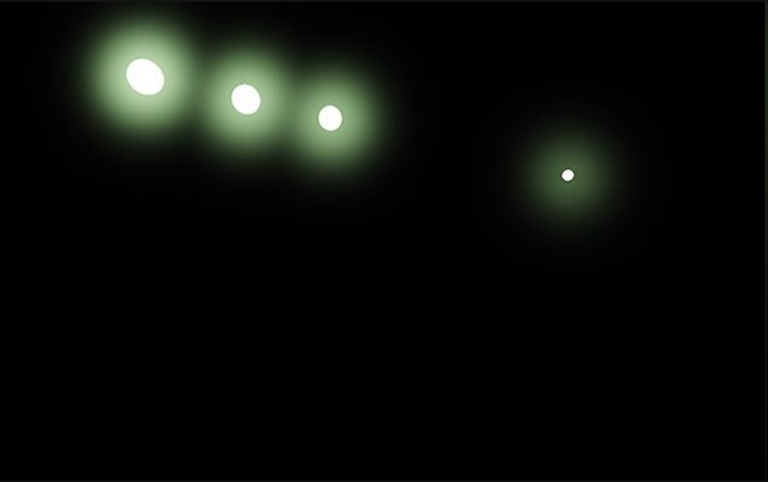
```
blur(blur(X,a),b) ~= blur(X,max(a,b))
```

- Combine with dirt texture



# Bloom Example



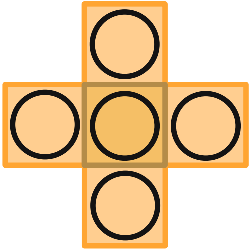


Bloom with single Gaussian

Bloom with 5 Gaussians and Dirt

# GBuffer Blur 1/2

- Smart blur:
  - Average of 5 pixels
  - Weighted by normal
  - Weighted by depth difference

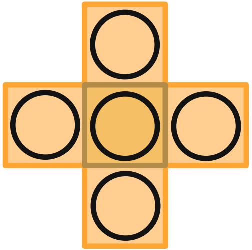


- Applications:
  - Reduce aliasing of specular materials (noticeable in motion)
  - Reduce high frequency dither artifacts in Ambient Occlusion
  - Can increase performance of with IBL or Voxel Lighting

# GBuffer Blur 2/2

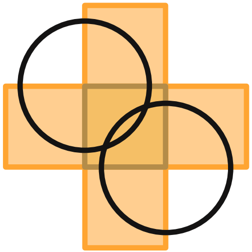
- Using Gather() where possible (Depth, AO)
- Output: SpecularPower, Normal, AmbientOcclusion
- Reduce Specular Power [Toksvig05] [Bruneton11]

```
L = saturate(length(SumNormal) * 1.002)
SpecularPower *= L / (L + SpecularPower * (1 - L))
```

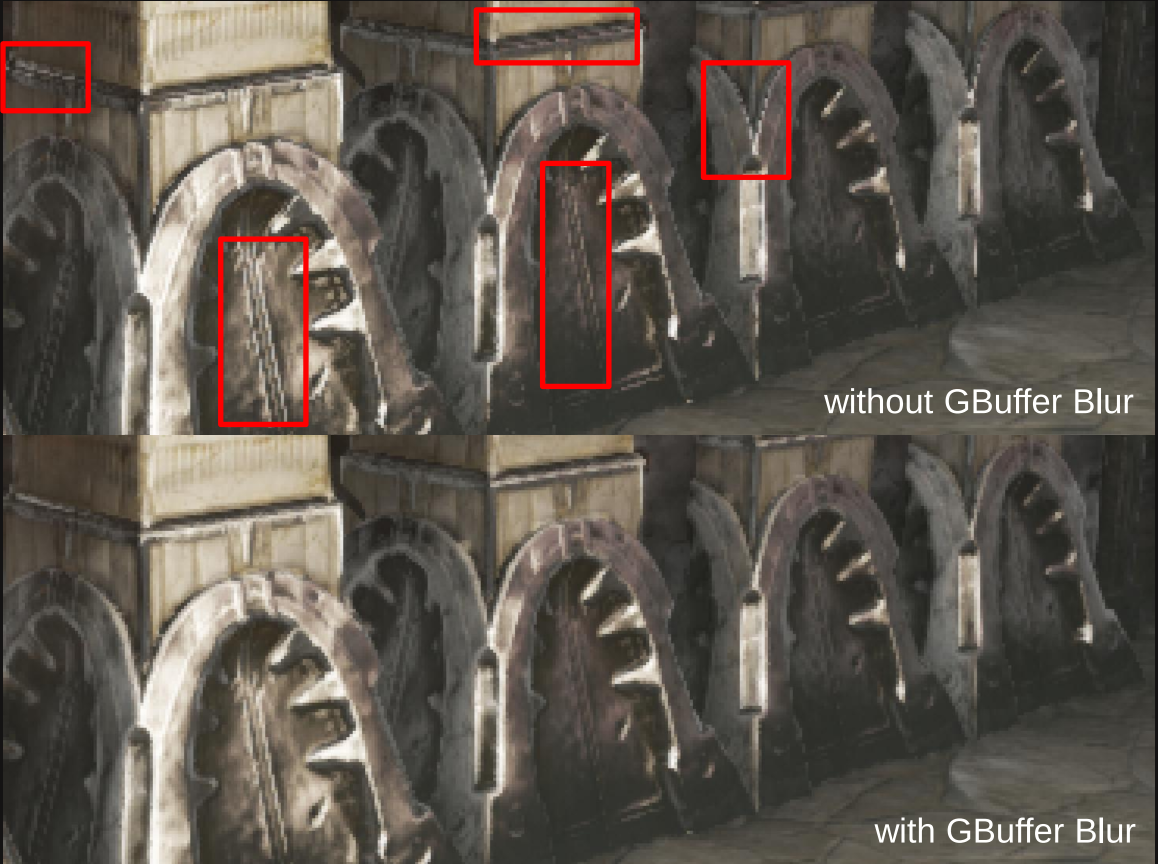


Kernel using 5 samples


single Gather



Kernel using 2 Gather


without GBuffer Blur
with GBuffer Blur

# SSS Material Example



unshadowed

shadowed

# 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