NVIDIA

Modern Real-Time Rendering Techniques

Louis Bavoil
NVIDIA

Outline

- Practical real-time rendering algorithms for:
  - DirectX 11 Tessellation
  - Transparency
    - Particle Rendering
    - Order Independent Transparency
  - Post-Processing Effects
    - Screen Space Ambient Occlusion
    - Depth Of Field

DirectX 11 Tessellation

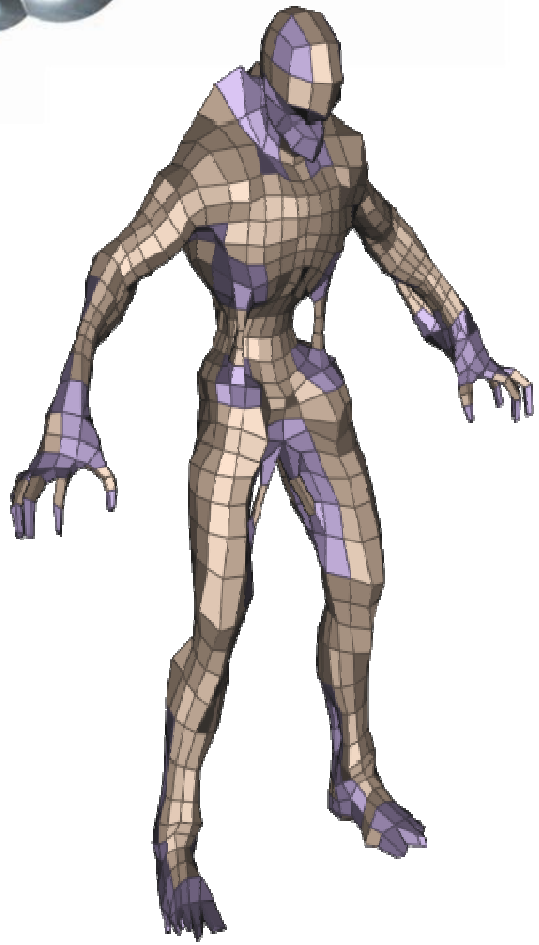
Geometric Detail in Games

- Pixels are meticulously shaded
  - But geometric detail is modest

Geometric Detail in Films

- Pixels are meticulously shaded
  - And geometric detail is substantial
- Dynamic tessellation + displacement mapping
  - Defacto standard for film rendering
  - Enables richer content and animation

Displacement Mapping



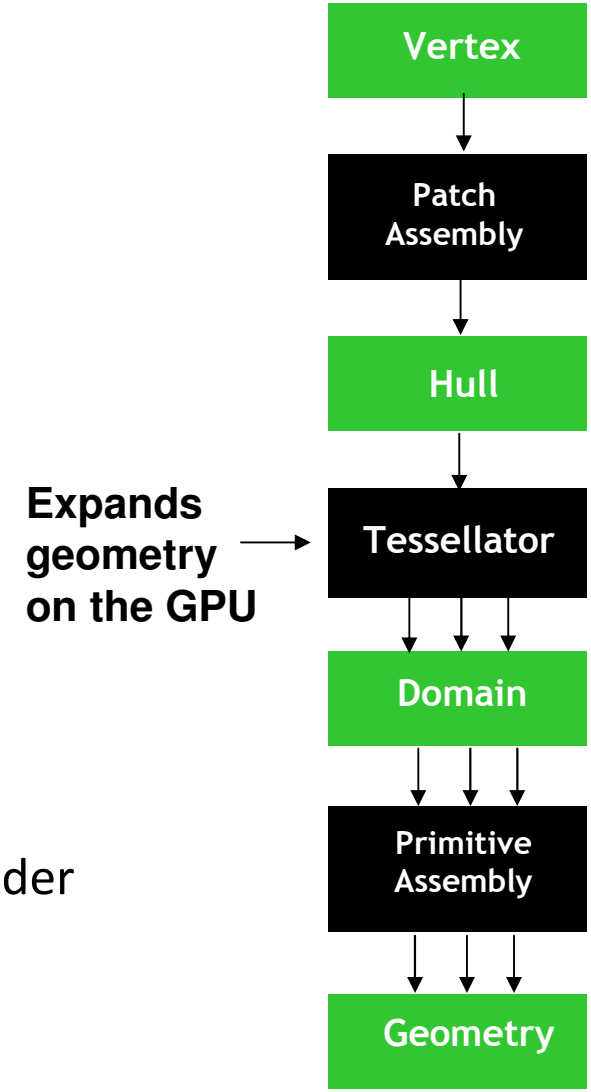
Input control cage

After tessellation

After tessellation + displacement mapping

DX11 Tessellation Pipeline

- Hull shader
  - Runs pre-expansion
  - Input = 1 patch (control points)
- Domain shader
  - Runs post-expansion
  - Input = 1 tessellated vertex
  - Can perform displacement mapping
  - Can shade at intermediate frequency between vertices and pixels
- Fixed-function tessellation stage
  - Configured by LOD output from hull shader
  - Produces triangles or lines



DX11 Tessellation Benefits

- Memory footprint & bandwidth savings
  - Store coarse geometry, expand on-demand
  - Enables higher geometry throughput
- Computational efficiency
  - Dynamic Level Of Detail (LOD)
  - Animate coarse geometry pre-expansion
- Scalability
  - Dynamic LOD allows for performance/quality tradeoffs
  - Scale into the future – resolution, compute power

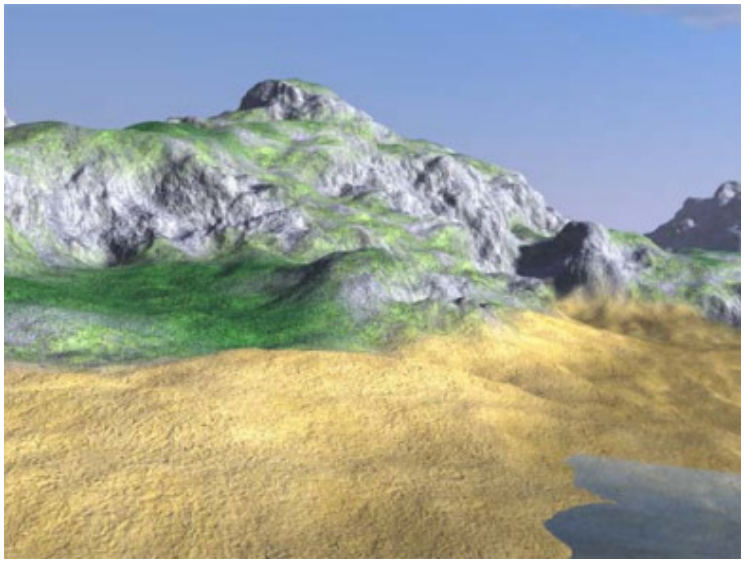
DX11 Tessellation On/Off

Tessellation in Metro 2033

Displacement mapping enables film-level geometric complexity in real-time

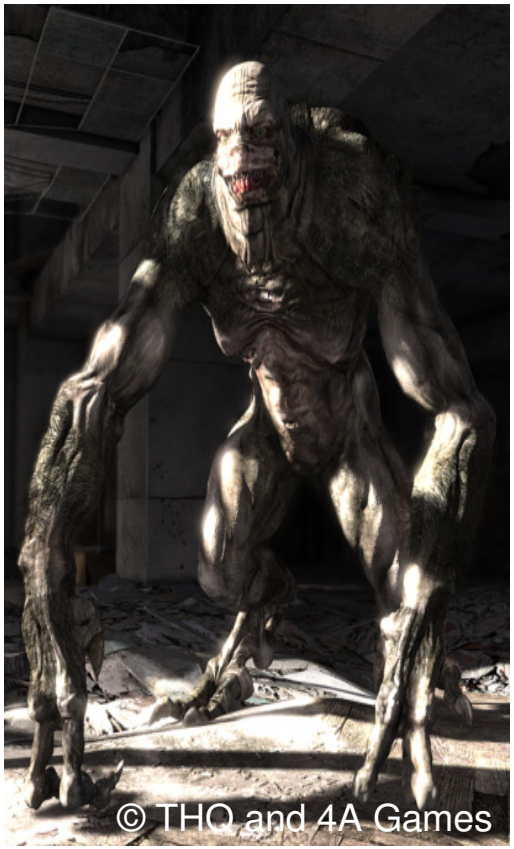
Screenshots from Metro 2033

More Tessellation Use Cases



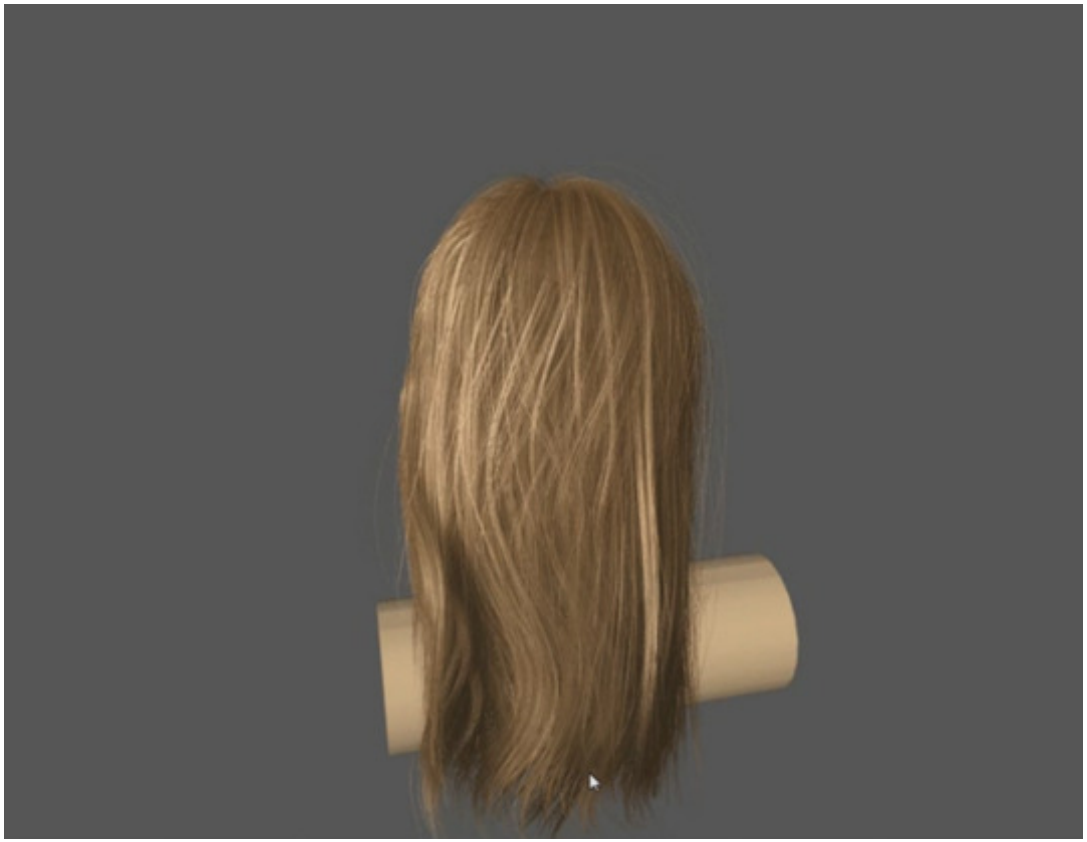
Tessellation Algorithms

- For subdivision surfaces
  - "Phong Tessellation"
    [Boubekeur and Alexa 08]
  - See GDC 2010 slides
    [Ni 10]
- For hair strands
  - See SIGGRAPH 2010 slides
    [Tariq and Yuksel 10]



Phong Tessellation

Hair Video

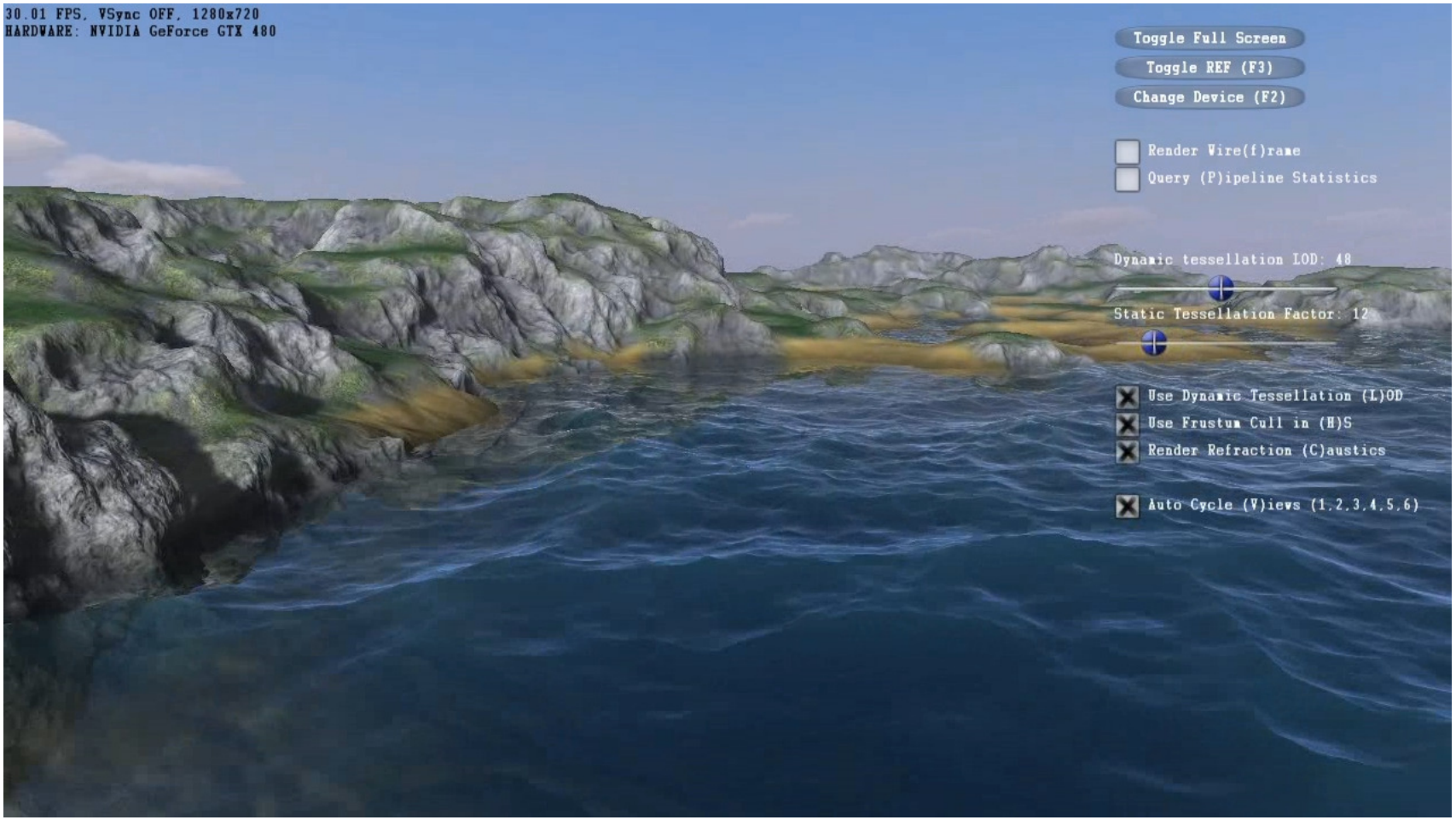


18,000 rendered strands on the GPU from a few 100 simulated strands

Grass Video


Toggle full screen
Toggle HDR (F3)
Change device (F2)
Render (G)rass
Render Wire(f)rame
Render Visu(t)rals
Query Statistics

Island Video



Particle Rendering

Rendering Particles

- Particle = camera-facing quad with texture
  - Useful for rendering smoke, steam, snow, waterfalls, etc
- Typically bottlenecked by raster ops or pixel shading
  - Possible optimization = low-resolution rendering

From Batman: Arkham Asylum

Low-Resolution Particles

- Render particles into off-screen low-res render target
  - E.g. half or quarter resolution
- Need to perform depth test against opaque depth buffer
  - In the pixel shader
  - And/or using a downscaled hardware depth buffer
- Main issue: blocky artifacts due to low-res depth test

Full-res particles Low-res particles

[Cantlay 07]

Fixing Blocky Artifacts

- Downscale depths using a max filter
  - Helps but doesn't remove all z-test artifacts
- Mixed resolution rendering
  - Can fix all artifacts
  - But refinement pass is expensive
- Joint Bilateral Upsampling
  - Can use a 2x2 kernel with
    - Spatial weights = bilinear weights (hard coded)
    - Range weights = $\exp(-\sigma.(z_{halfres} - z_{fullres})^2)$
  - May cause artifacts for island pixels
    - If all weights are close to zero

Low-res particles

[Cantlay 07]
[Kopf et al. 07]
[Shopf 09]

Nearest-Depth Filter

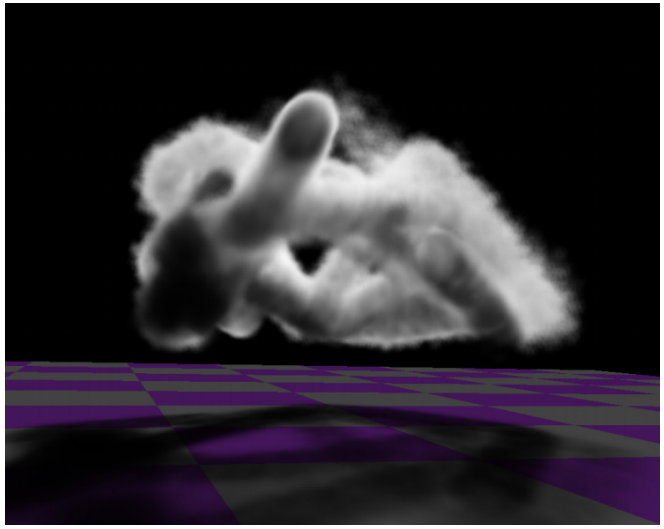
- 2x2 upsampling filter used in Batman: Arkham Asylum
  - For the low-resolution particles (half-res and quarter-res)

- Step 1: fetch linear depths
  - Fetch the full-res depth z_c at center uv_c
  - Fetch the nearest 2x2 low-res depths z_{sample}
  - For each sample, keep track of uv_{min} with minimum $|z_{sample} - z_c|$

- Step 2: fetch low-res color
  - If $|z_{sample} - z_c| < \varepsilon$ for all samples
    - Then fetch low-res texture with **bilinear** filtering at uv_c
    - Else fetch low-res texture with **point** filtering at uv_{min}

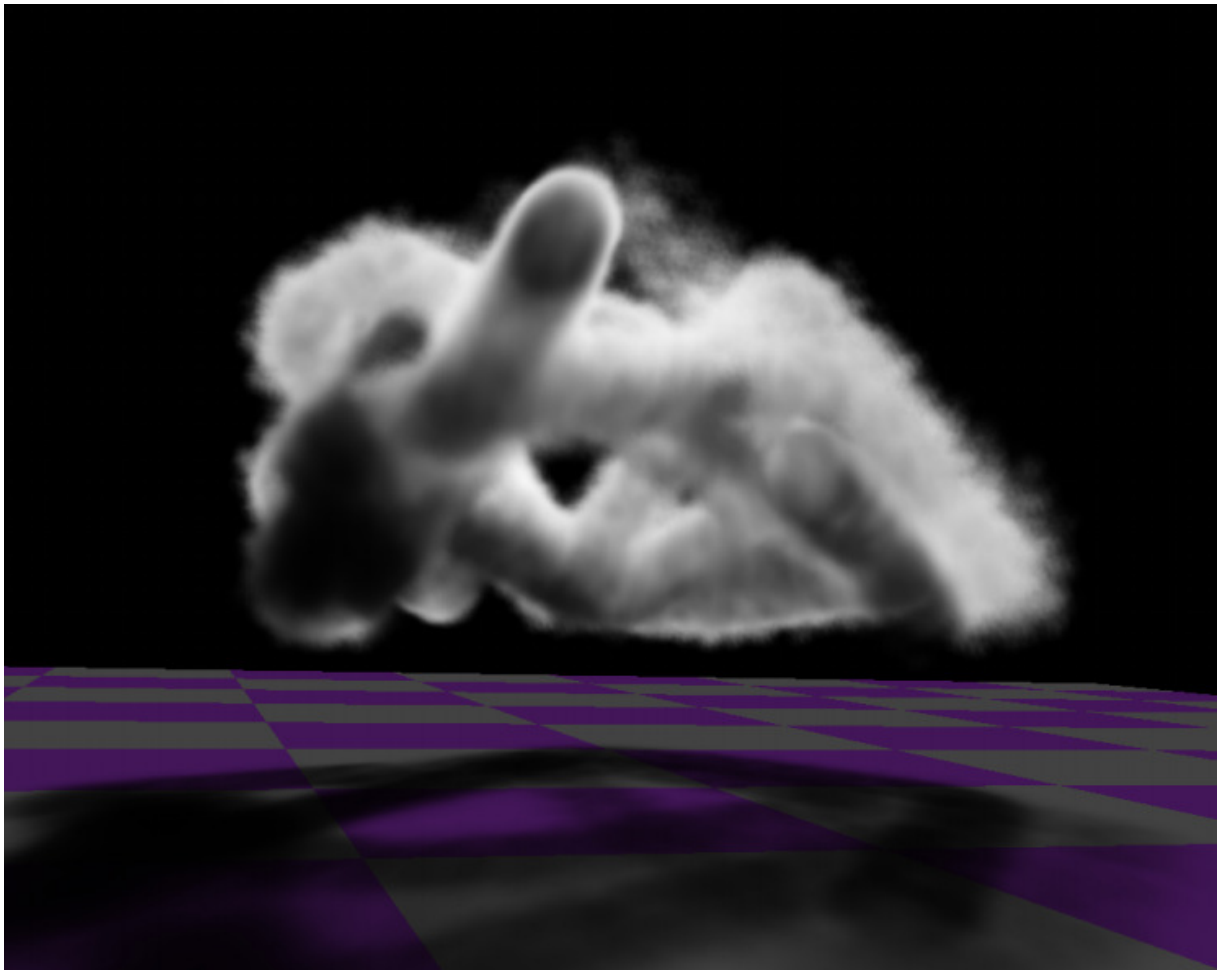
Shadowing Particles

- Particles can…
  - Receive opaque shadows
    - From opaque objects using regular shadow mapping
  - Cast and receive volumetric shadows from particles
    - With Fourier Opacity Mapping
- Fourier Opacity Mapping (FOM)
  - Used in Batman: Arkham Asylum



Fourier Opacity Mapping
[Jansen and Bavoil 10]

FOM Video



Order Independent Transparency

Alpha Blending

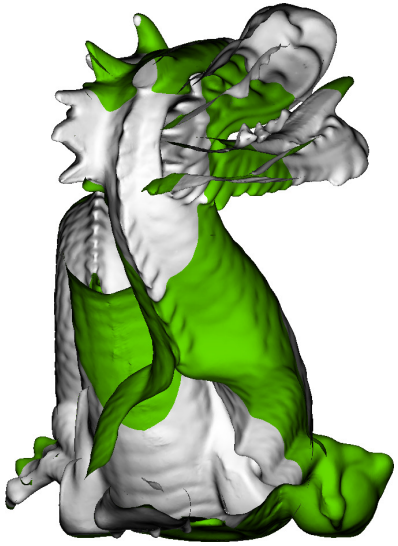
- Alpha Blending
  - Useful for rendering alpha-blended vegetation, semi-transparent objects, etc.
  - Requires depth-sorted order
    - Back to front or front to back

- Sorting triangles
  - Can be expensive to do for all triangles
  - Not correct if triangles intersect one another

Depth Peeling

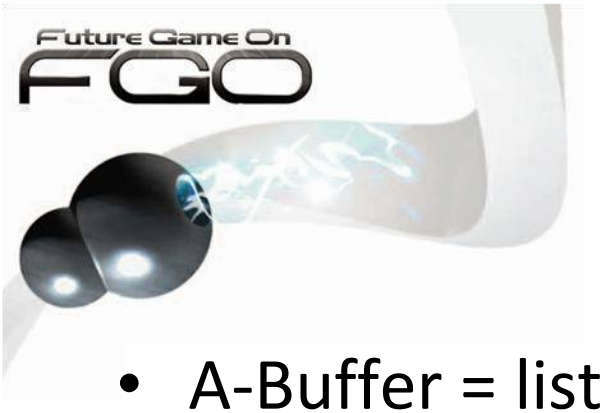
- Depth peeling and dual depth peeling
  - Captures color layers in depth-sorted order
  - Blends layers on the fly

First layer



Second layer

[Everitt 01]
[Bavoil and Myers 08]

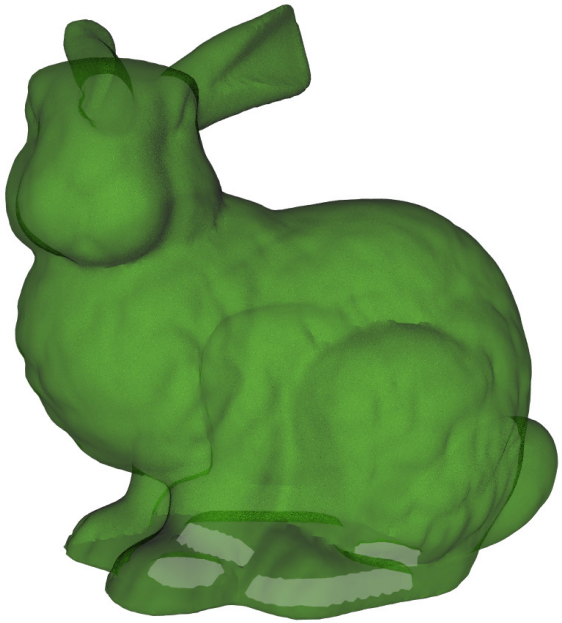


A-Buffers

- A-Buffer = list of fragments per pixel [Carpenter 84]
- Approach: capture all the translucent fragments in rasterization order, not in depth order
  - Stencil-Routed K-Buffer [Myers and Bavoil 07]
  - Per-Pixel linked Lists [Gruen and Thibieroz 10]
  - CUDA rasterization [Liu et al. 10]
- Issue: video memory
  - Need to store all the fragments before processing them

Stochastic Transparency

- Stochastic Transparency [Enderton et al. 10]
  - Extension of alpha to coverage
  - Pros: No sorting needed, naturally works with MSAA
  - Con: Noisy output



With 8 MSAA samples per pixel

k-NSS

- k-NSS = k Nearly-Sorted Sequences
  - Used for volume rendering of tetrahedral meshes
    - [Callahan et al. 05] [Bavoil et al. 07]
  - Hybrid algorithm
    - Object space: Sort triangles by centroid depth
    - Image space: Reorder fragments in a sliding window of size k
      - Assuming that fragments are no more than k out-of-order
      - Pixel shader outputs nearest or farthest fragment from k-buffer
      - Fragments are blended on the fly using fixed-function alpha blending

- With DX11 ps_5_0 pixel shaders
  - Can implement k-NSS fragment sorting
    - Using InterlockedMin in pixel shader
    - Similar to “multi depth test” from [Liu et al. 10]

Screen Space Ambient Occlusion

Ambient Occlusion (AO)

- AO can be defined as the fraction of sky seen from each point
- Gives perceptual clues of curvature and spatial proximity

Without AO

With AO

SSAO

- SSAO = Screen Space Ambient Occlusion
- Approach introduced concurrently by
  - [Shanmugam and Orikan 07]
  - Crytek [Mittring 07] [Kajalin 09]
- Post-processing pass
  - Use depth buffer as representation of the scene
  - Can multiply SSAO over shaded colors

HBAO

- HBAO = Horizon-Based Ambient Occlusion
  - SSAO algorithm developed by NVIDIA
  - [Bavoil and Sainz 08] [Bavoil and Sainz 09]

- Normal-Free HBAO
  - “Low Quality” mode in DX10 SDK SSAO sample
  - Input = view-space depths only (no normals needed)
  - Integrates AO in the full sphere
  - Used in Battlefield: Bad Company 2 [Andersson 10]

With and Without Normals

HBAO with normals

Normal-Free HBAO

HBAO OFF
From Battlefield: Bad Company 2 © EA/DICE

HBAO ON
From Battlefield: Bad Company 2 © EA/DICE

HBAO Performance

- Typical parameters
  - Normal-free HBAO pass
    - Half-res, 8x6 depth samples per AO pixel
    - Max footprint width = 5% of screen width
  - Blur passes (horizontal + vertical)
    - Full-res, fetch packed (ao,z)
    - Blur radius = 20 pixels to remove flickering
  - MSAA disabled for SSAO passes
    - SSAO aliasing is rarely objectionable

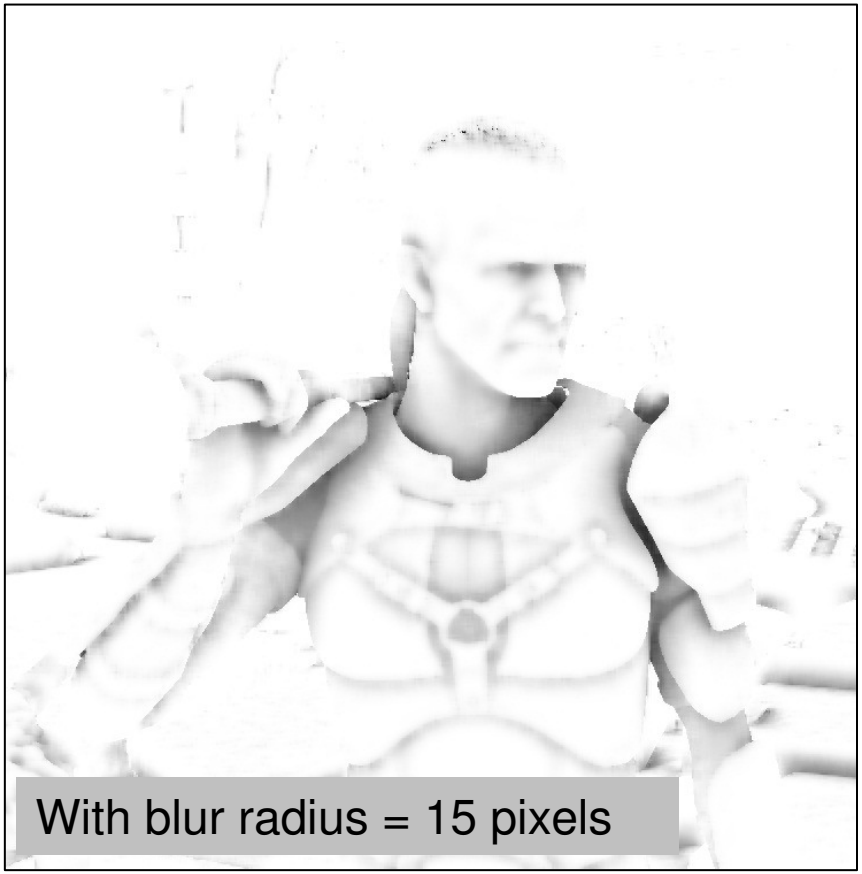
- Typical 1920x1200 performance on GeForce GTX460
  - Total HBAO cost ~= 5 ms
    - 2.3 ms for occlusion pass
    - 2.8 ms for blur passes

Other SSAO Algorithms

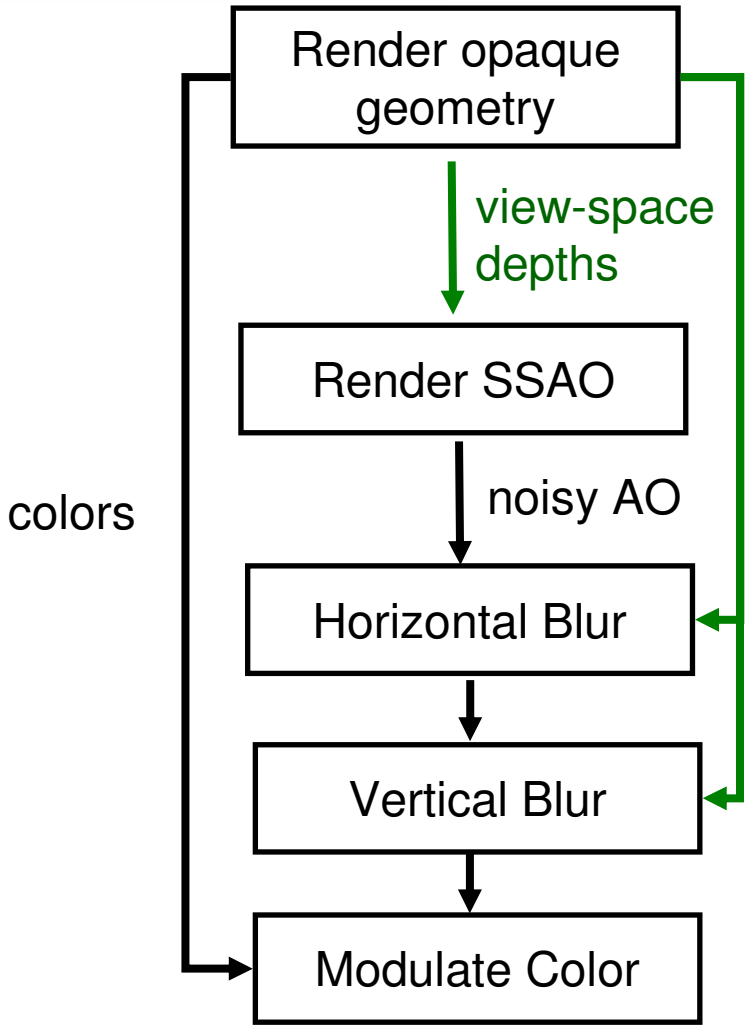
- "Crytek Algorithm" [Kajalin 09]
  - Used in Crysis
  - Uses 3D sample points around surface point
- "Volumetric Obscurance" [Loos and Sloan 10]
  - Used in the game Toy Story 3 [Ownby et al. 10]
  - Similar to "Volumetric AO" [Szirmay-Kalos et al. 10]
  - Uses 2D sample points around pixel (like HBAO)
- Other algorithms can also do color bleeding
  - More expensive due to additional color fetches

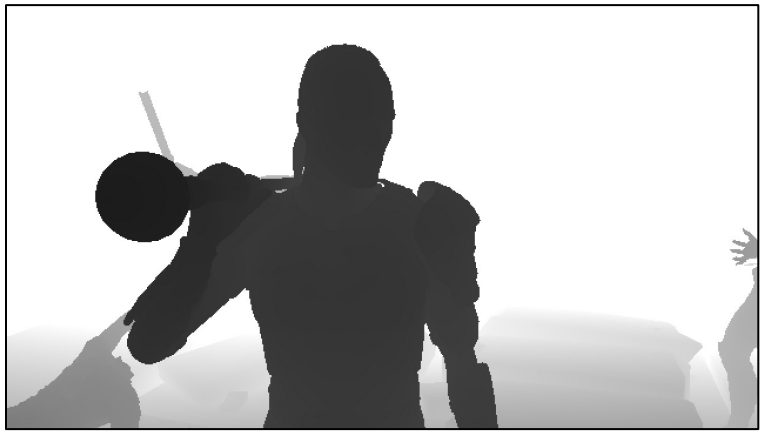
Cross Bilateral Filter

- To remove noise and flickering



Typical SSAO Pipeline





Input = view-space depths

Output = SSAO image

SSAO Performance

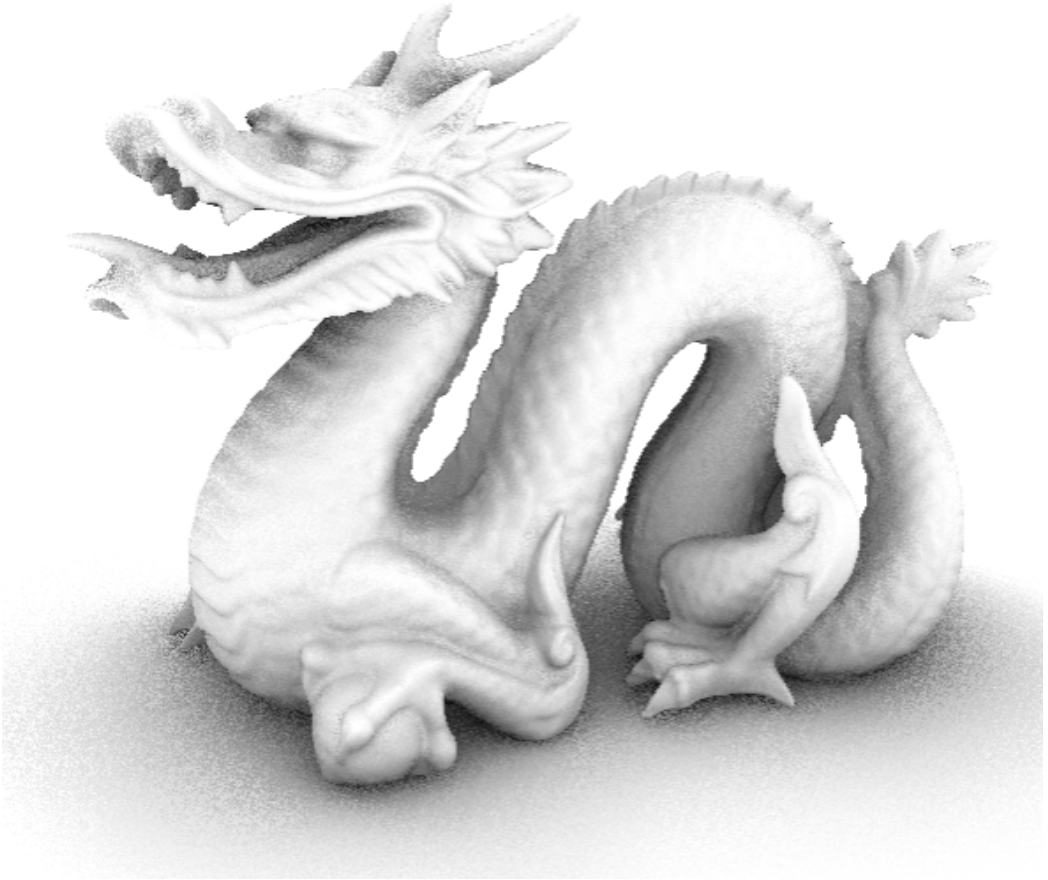
- General recommendations for SSAO
  - Keep the kernel footprint tight
    - To minimize texture cache misses
    - By sourcing low-resolution depth texture
    - By clamping the radius in screen space
  - Try using temporal filtering
    - Allows reducing the number of samples per frame in the SSAO and blur shaders

Temporal Filtering

- Spatial filtering limitations
  - Performance: blur radius >= 20 pixels may be required to remove all noise and flickering
  - Quality: blurring removes high-frequency details

- Temporal filtering [Smedberg and Wright 09] [Soler et al. 10]
  - Performance: one texture lookup from previous frame
  - Use different noise textures from frame to frame
  - Can do spatial filter followed by temporal [Herzog et al. 10]
  - For SLI perf, need to avoid inter-frame dependencies

Temporal Filtering Video



Depth Of Field

DOF Use Cases in Games

From Metro2033, © THQ and 4A Games

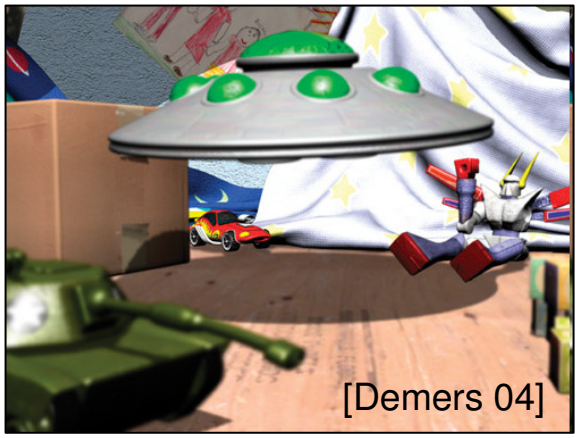
From Metro2033, © THQ and 4A Games

From Call of Duty 4, © Activision and Infinity Ward

DOF Algorithms

- Survey available in GPU Gems [Demers 04]

- Layered DOF
  - Assumes objects can be sorted in layers
    - Not always applicable
  - Blur background (or foreground) separately
    - With fixed-size kernel
  - Issue with transitions between blur / sharp

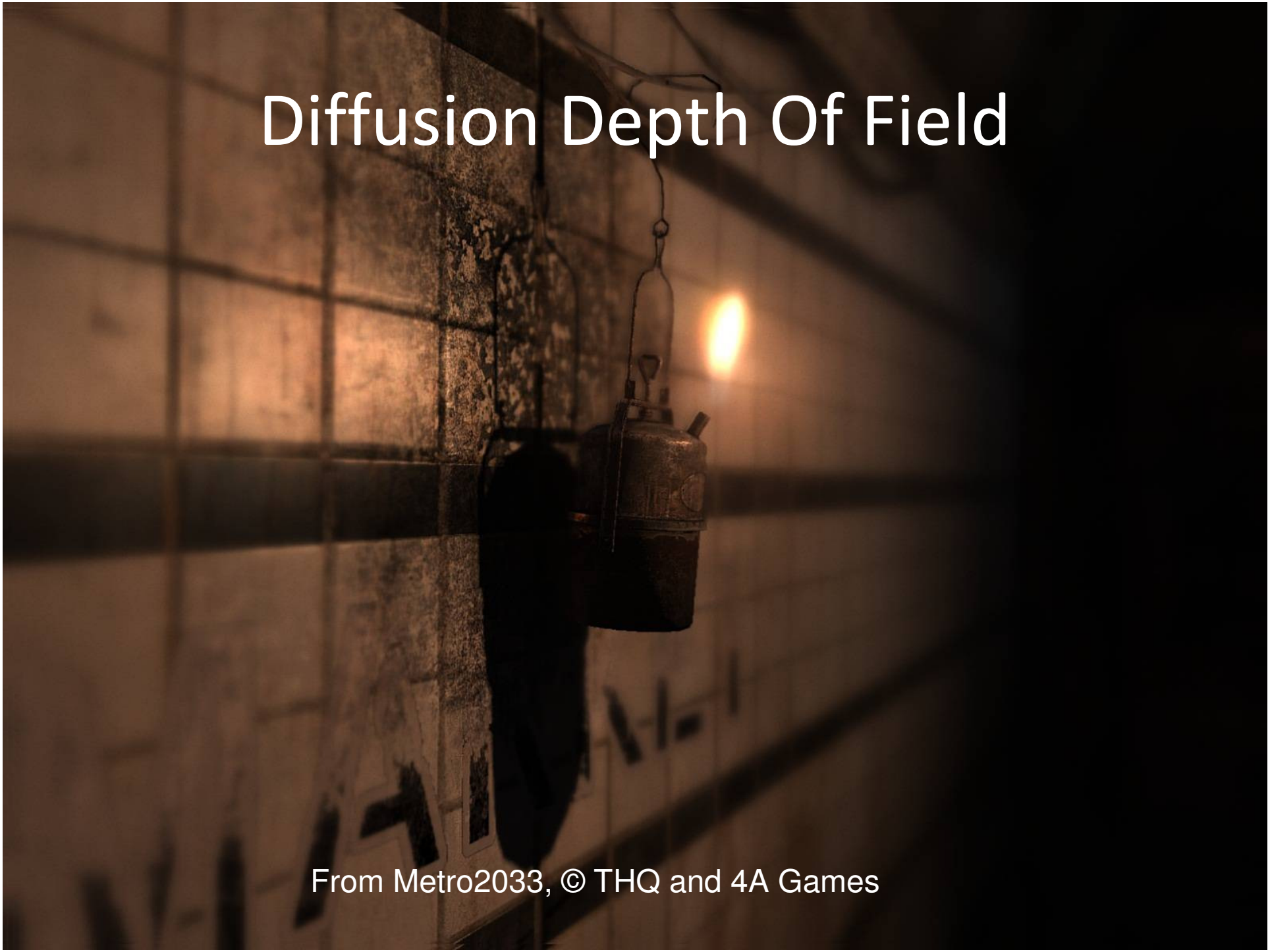


- Gather-based DOF
  - Variable blur radius
    - Function of depth and camera parameters
  - Filter colors using depth-dependent weights
    - To avoid bleeding across edges
  - Main drawback: limited blur radius

Diffusion Depth Of Field

- "Diffusion Depth Of Field"
  - Heat diffusion simulation
    - [Kass et al. 06]
  - Unlimited blur size
  - Used in Metro 2033
- Can use DirectCompute
  - For solving tri-diagonal systems in parallel
    - [Zhang et al. 10]
  - 8 ms / frame in 1600x1200 on GeForce GTX480
    - [Sakharnykh 10]

Diffusion Depth Of Field



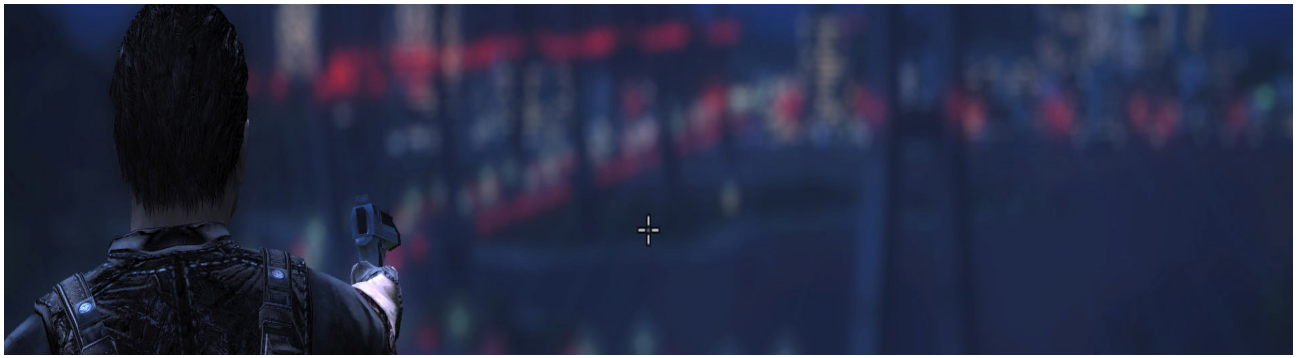
From Metro2033, © THQ and 4A Games

Bokeh Filters

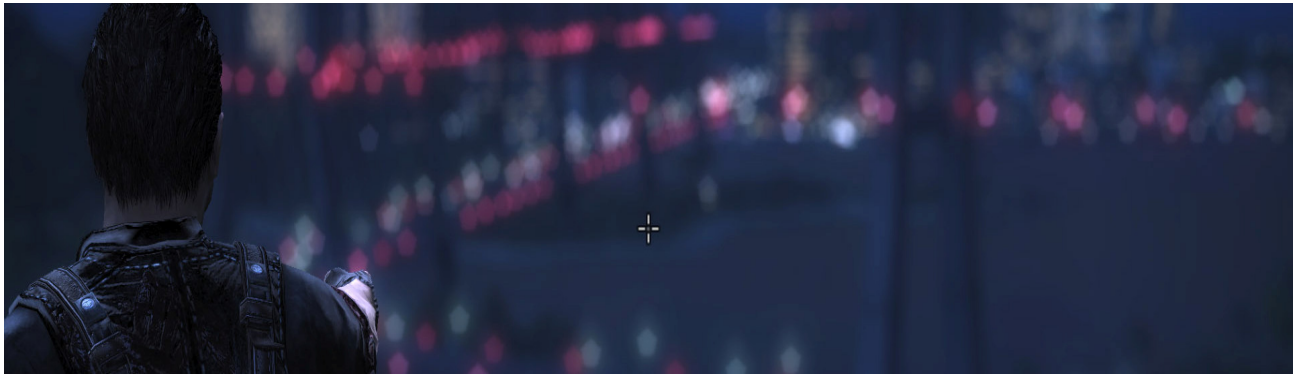
- The Bokeh effect
  - “Bokeh” is the Japanese word for blur
  - Refers to the blurred shape of light sources
  - Can take various shapes
  - It is a very well know effect used in films

- Bokeh filters require a non separable convolution for shapes other than circles
  - Can benefit from CUDA shared memory

CUDA Bokeh in Just Cause 2



Original



With Bokeh

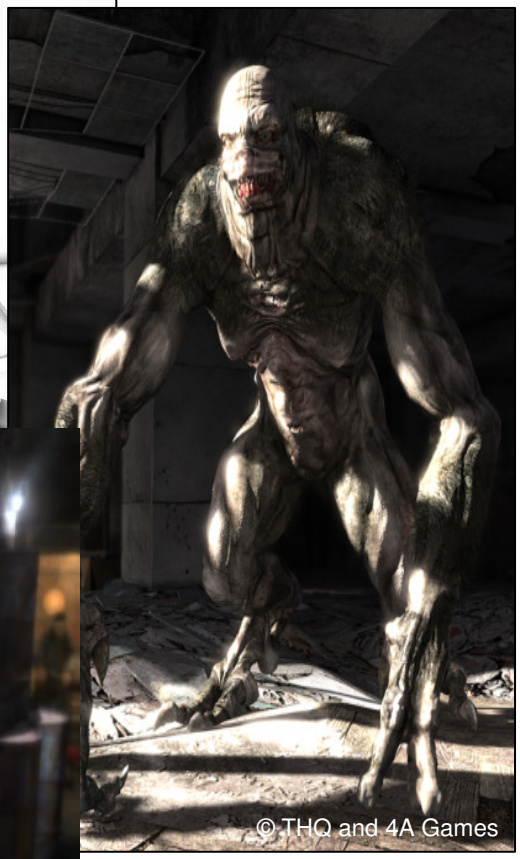
From Just Cause 2 © Eidos and Avalanche Studios

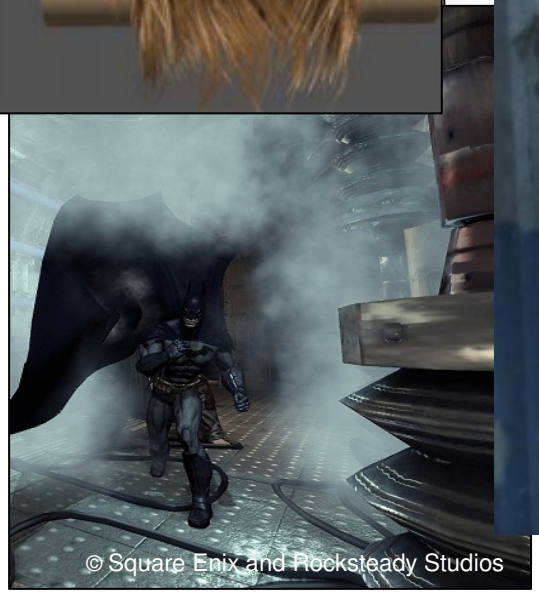
Conclusion

- Discussed variety of rendering techniques
  - Taking advantage of latest GPUs
  - Tessellation, transparency, and post-processing effects (SSAO, depth of field)

- Acknowledgments
  - EA / DICE, Square Enix / Rocksteady Studios, THQ / 4A Games, Eidos / Avalanche Studios, Unigine Corp
  - NVIDIA Developer Technology Group
  - Eric Enderton for OIT discussions
  - FGO organizers

Questions?





lbavoil@nvidia.com

References

- DirectX 11 Tessellation
  - [Tariq and Yuksel 10] S. Tariq, C. Yuksel, “Advanced Techniques in Real Time Hair Rendering and Simulation”, SIGGRAPH 2010 Course
  - [Ni 10] Tianyun Ni, “Enriching Details using Direct3D 11 Tessellation”, GDC 2010 Talk
  - [Boubekeur and Alexa 08] Tamy Boubekeur, Marc Alexa, “Phong Tessellation”, SIGGRAPH Asia 2008

- Particle Rendering
  - [Jansen and Bavoil 10] Jon Jansen, Louis Bavoil, “Fourier Opacity Mapping”, I3D 2010
  - [Shopf 09] Jeremy Shopf, “Mixed Resolution Rendering”, GDC 2009
  - [Kopf et al. 07] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, “Joint Bilateral Upsampling”, SIGGRAPH 2007
  - [Cantlay 07], Iain Cantlay, “High-Speed, Off-Screen Particles”, GPU Gems 3

References

- Order-Independent Transparency (OIT)
  - [Enderton et al. 10] E. Enderton, E. Sintorn, P. Shirley, D. Luebke, “Stochastic Transparency”, I3D 2010
  - [Gruen and Thibieroz 10] H. Gruen and N. Thibieroz, “OIT and Indirect Illumination using DX11 Linked Lists”, GDC 2010
  - [Liu et al. 10] Liu, Huang, Liu, Wu, “FreePipe: a programmable parallel rendering architecture for efficient multi-fragment effects”, I3D 2010
  - [Bavoil and Myers 08] L. Bavoil and K. Myers, “Order Independent Transparency with Dual Depth Peeling”, Technical report, NVIDIA, 2008
  - [Myers and Bavoil 07] "Stencil Routed A-Buffer", Kevin Myers, Louis Bavoil, ACM SIGGRAPH Technical Sketch Program, 2007
  - [Bavoil et al. 07] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. D. Comba, C. T. Silva, “Multi-Fragment Effects on the GPU using the k-Buffer”, I3D 2007
  - [Callahan et al. 05] S. P. Callahan, M. Ikits, J. L. D. Comba, C. T. Silva, “Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering”, IEEE TVCG, 2005
  - [Everitt 01] C. Everitt, “Interactive order-independent transparency”, Technical report, NVIDIA, 2001
  - [Carpenter 84] L. Carpenter, “The A-buffer, an antialiased hidden surface method”, SIGGRAPH 1984

References

- Screen Space Ambient Occlusion (SSAO)
  - [Andersson 10] J. Andersson, “Bending the Graphics Pipeline”, SIGGRAPH 2010 Course
  - [Loos and Sloan 10] Loos, Sloan, “Volumetric Obscurance”, I3D 2010
  - [Szirmay-Kalos et al. 10] Szirmay-Kalos, Umenhoffer, Tóth, Szécsi, Sbert, “Volumetric Ambient Occlusion”, Technical Report, 2010
  - [Ownby et al. 10] J-P. Ownby, R. Hall and C. Hall, “Rendering techniques in Toy Story 3”, SIGGRAPH 2010 Course
  - [Kajalin 09] V. Kajalin, “Screen Space Ambient Occlusion”, ShaderX 7, 2009
  - [Bavoil and Sainz 09] L. Bavoil, M. Sainz, “Image-Space Horizon-Based Ambient Occlusion”, ShaderX 7, 2009
  - [Bavoil and Sainz 08] L. Bavoil, M. Sainz, “Image-Space Horizon-Based Ambient Occlusion”, SIGGRAPH 2008 Talk
  - [Shanmugam and Arikan 07] P. Shanmugam, O. Arikan, “Hardware accelerated ambient occlusion techniques on GPUs”, I3D 2007
  - [Mittring 07] Mittring, “Finding next gen: Cry Engine 2”, SIGGRAPH 2007 Course

References

- Temporal Filtering
  - [Herzog et al. 10] Herzog, Eisemann, Myszkowski, Seidel, “Spatio-temporal upsampling on the GPU”, I3D 2010
  - [Soler et al. 10] C. Soler, O. Hoel, F. Rochet, N. Holzschuch, “A Deferred Shading Algorithm for Real-Time Indirect Illumination”, SIGGRAPH 2010 Talk
  - [Smedberg and Wright 09] N. Smedberg, D. Wright, “Rendering Techniques in Gears of War 2”, GDC 2009

- Depth Of Field
  - [Sakharnykh 10] N. Sakharnykh, “Depth of Field using DirectX Compute”, KRI Conference Talk, May 2010
  - [Zhang et al. 10] Y. Zhang, J. Cohen, J. D. Owens, “Fast Tridiagonal Solvers on the GPU”, Principles and Practice of Parallel Programming (PPoPP), 2010
  - [Kass et al. 06] M. Kass, A. Lefohn, J. Owens, “Interactive Depth of Field Using Simulated Diffusion”, Pixar Technical Report, 2006
  - [Demers 04] J. Demers, “Depth of Field: A Survey of Techniques”, GPU Gems, 2004