Game Developer Conference 2011

The Technology Behind the DirectX 11 Unreal Engine "Samaritan" Demo

POWERED BY

UNREAL TECHNOLOGY

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

Bryan Dudash
Developer Technology
bdudash@nvidia.com
NVIDIA

NVIDIA

Overview

- About
- Real-time demo

- Technical Part:
  - Tessellation (NVIDIA)
  - Hair
  - Deferred + MSAA
  - Subsurface Scattering
  - Reflections
  - Depth of Field

Demo Goals

- Ready for GDC 2011
- Real-time on High-end PC (off the shelf hardware)
- Engine improvements:
  - Add Direct3D 11 support in Unreal Engine 3
  - Implement features needed for next-gen quality
- Research:
  - New hardware features like Tessellation
  - Advanced render techniques
  - Content creation / workflow

Storyboard to define the scope

=> Near shots, faces, hair, harsh lighting, rain

Derived Technology needs

Direct3D 11

- Tessellation (NVIDIA)

Filmic look

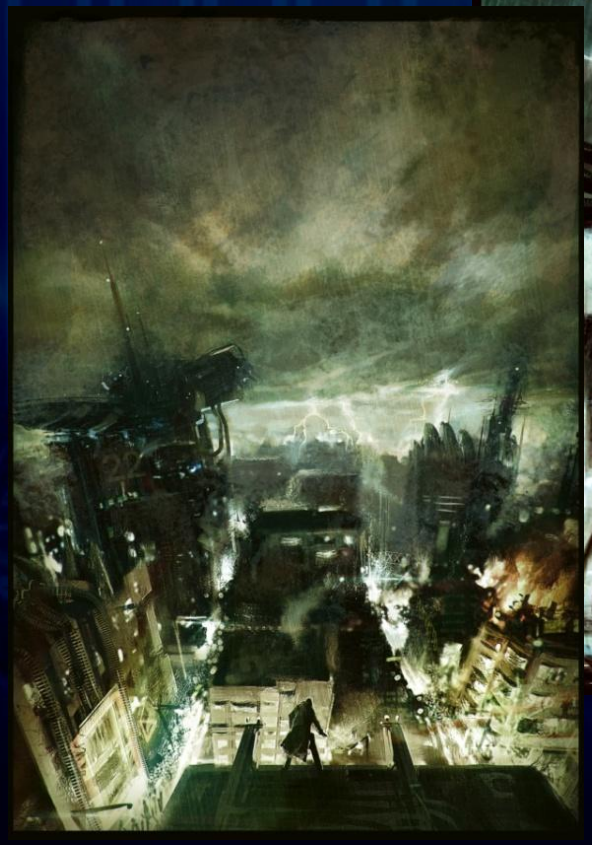
- Quality

Harsh lighting, night scene

- Dynamic Shadows

Rain

- Reflections
- Particles
- Animated water surface
- Wet material shading



Derived Technology needs

Close ups

- Depth of Field
- Facial expressions

Short scalp hair and beard

- Hair
- Simple animation
- Rather simple shading

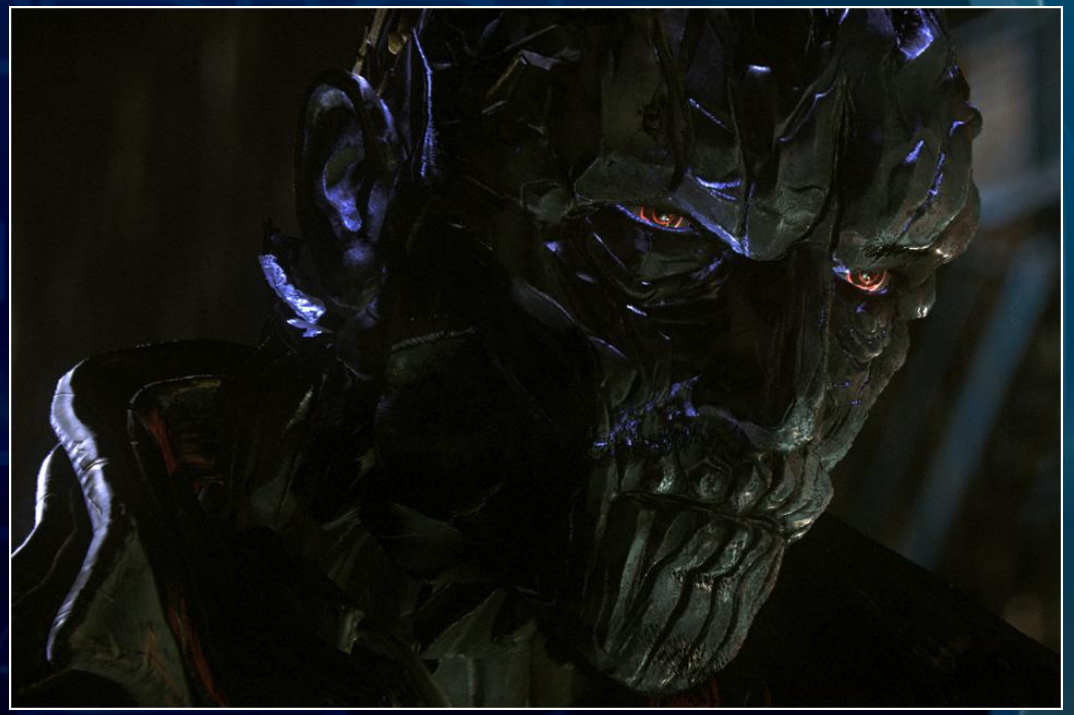
Coat

”Realistic and Interactive Clothing in Epic Games Samaritan Demo Using NVIDIA APEX” Thursday 4:30- 5:30 Room 110, North Hall

Video / Real-time demo

Rendering

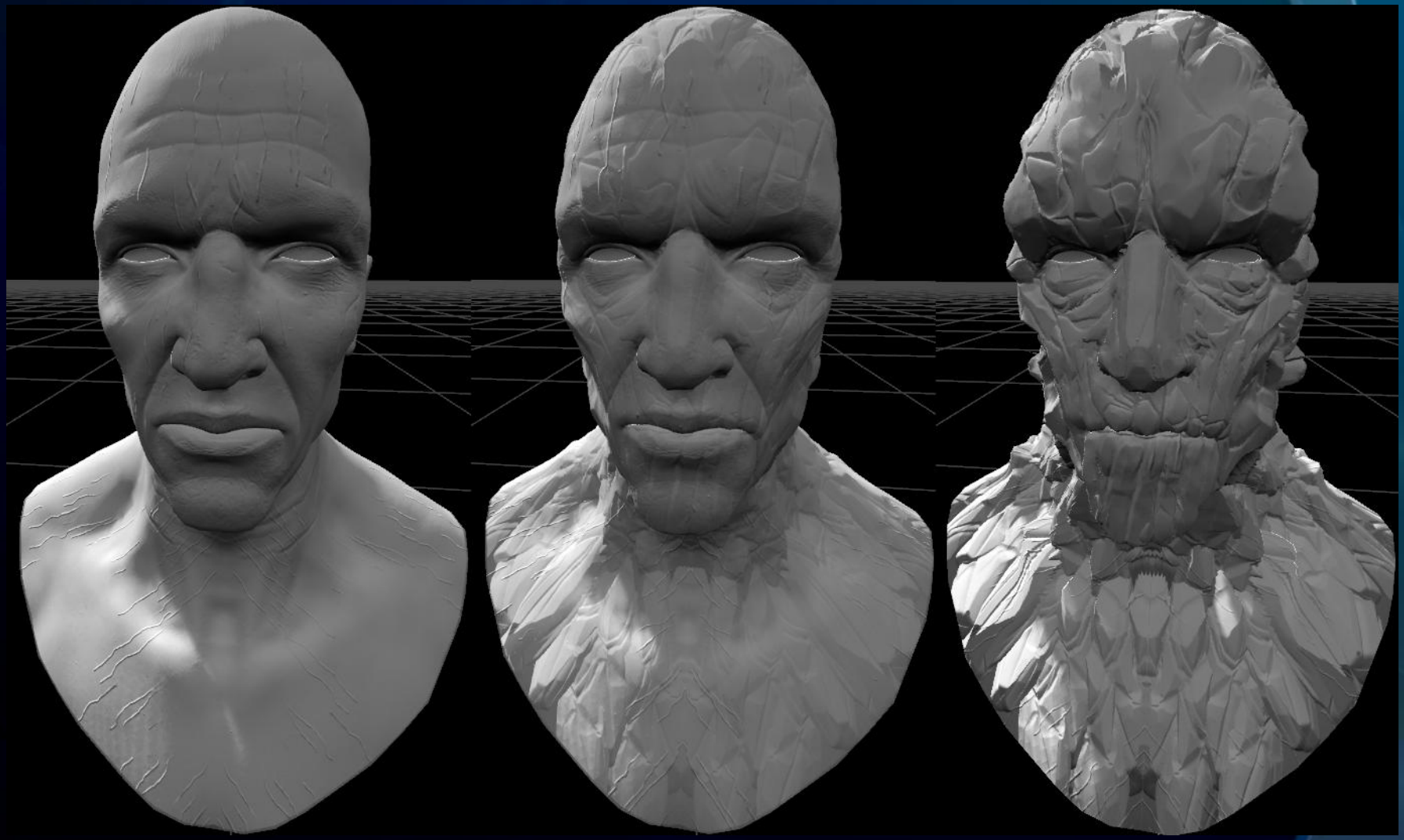
- Tessellation
- Hair
- Deferred + MSAA
- Subsurface Scattering
- Reflections
- Depth of Field



Samaritan Demo Tessellation Breakdown

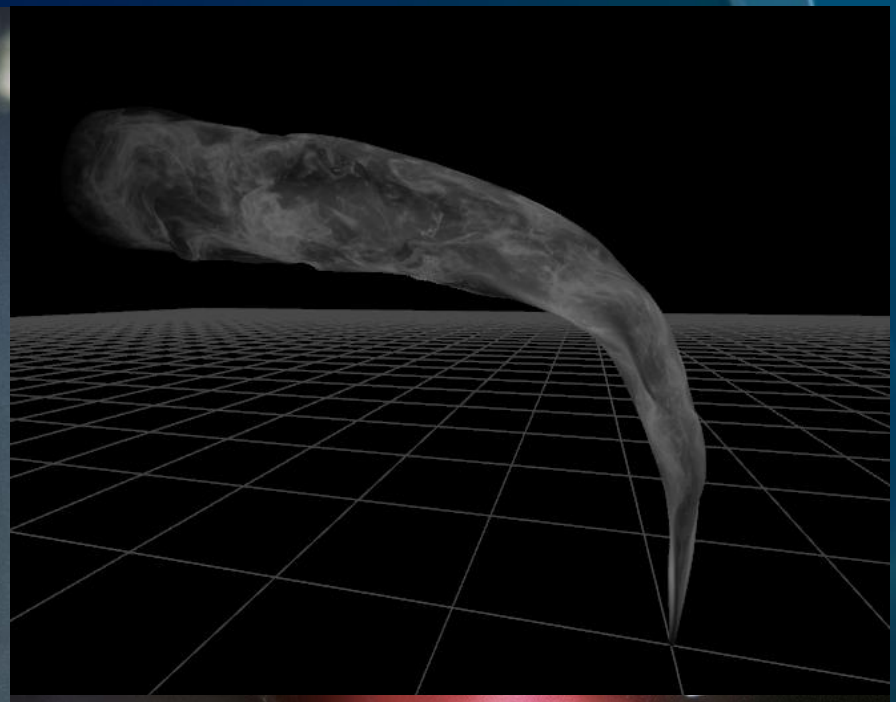
- Uses UE3 October 2010 release code
  - D3D11 RHI with “Tessellation Preview”
- Flat tessellation implementation
  - Texture seams positioned “off camera”
- Material based tessellation factors
  - Controlled via matinee scripting
- “Densing”
  - Animated displacement maps
  - Painted gradient defines flow of transition
- Cigarette Smoke
  - Low poly base mesh for volume
  - Animated world displacement

Mesh Change w/ Tessellation



NVIDIA

Smoke



NVIDIA

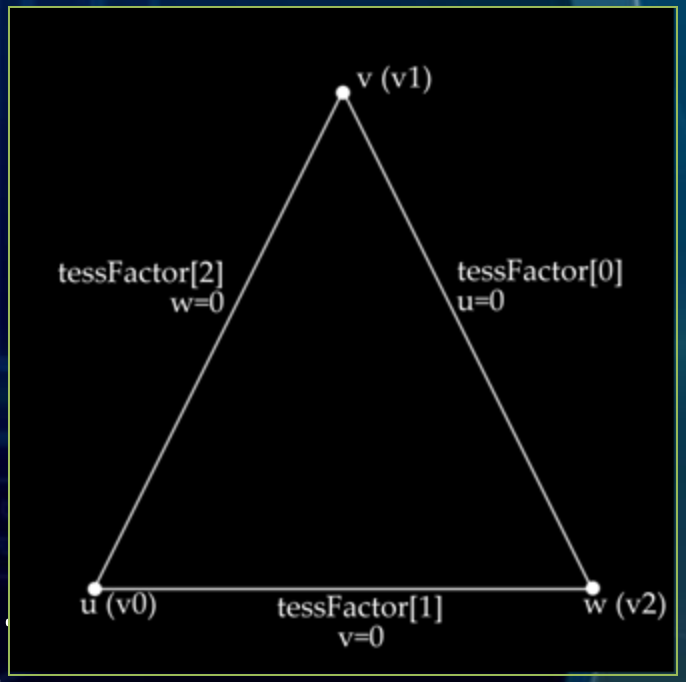
Tessellation Pipeline Review

- Input vertices a list of control points
  - VS runs normally
- Hull shader processes the primitive to be tessellated
  - HS Constant decides how much tessellation
- Tessellator is fixed function
- Domain shader turns tessellated coordinates into positioned vertices

NVIDIA

Edge Tessellation Factors Explained

- float[3] edge tessellation factors
  - tessFactor[0] => Opposite U=1
  - tessFactor[1] => Opposite V=1
  - tessFactor[2] => Opposite W=1
- Control points to barycentric is arbitrary
  - Most common maps V0 to U, etc.
  - Must be consistent between Hull and Domain shaders



A common barycentric mapping

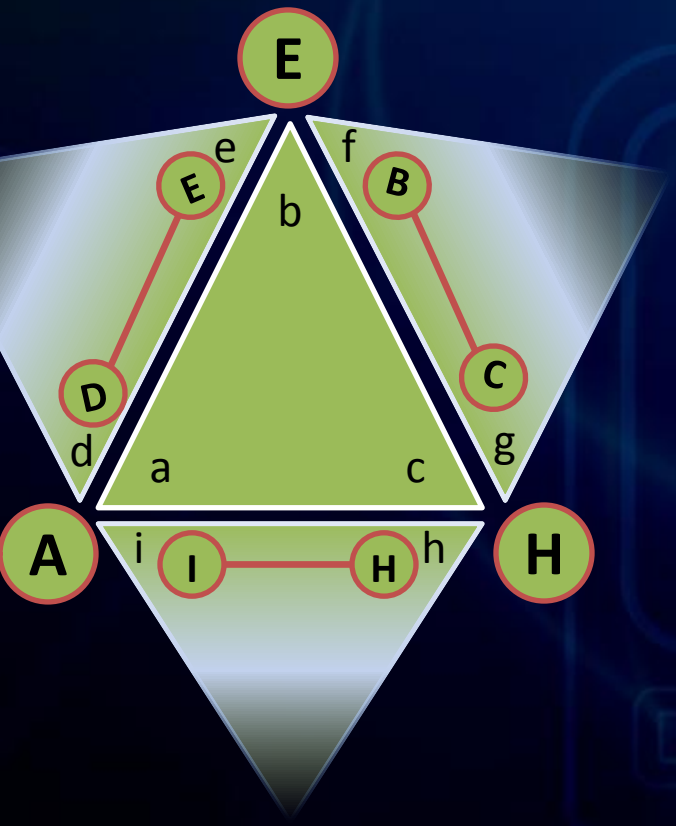
Coming soon to Unreal Engine
(tessellation related)

- Dominant edge/vertex solution
  - For displacement map cracking
- Dynamic tessellation
  - Efficient and smooth LOD
- Mip mapped displacement maps
  - Reduce mesh swimming from dynamic tessellation
- PN-AEN Tessellation
  - Divergent normals cracking

Displacement Maps and Cracks

- Texture space discontinuities
  - From normal texture unwrapping
    - Can hide this with content tweaks
  - Can also be in different spaces due to texture scaling
- Potential filtering discrepancies
  - Along discontinuous edges
- Shows up as cracks along seams

Dominant Edge Vertex



Index Buffer

a	b	c	D	E	B	C	H	I	A	E	H

- Preprocess mesh
  - Library available soon (Google code)
- Decide which edge/vertex is dominant
  - Arbitrary but consistent heuristic
- On Corners, use dominant vertex
  - Pins corner vertices in UV space
- Along edges, interpolate dominant edge
- One extra index per vertex in the primitive
- Two extra indices per edge in the primitive

NVIDIA

Dynamic Tessellation

- Define Easy perf knob => Pixels/triangle
  - Good value is about 32-64
- Convert to a user constant factor with simple heuristic
  - Tessellated Edges per RenderTarget Width
  - Passed as shader constant

```
RenderTargetWidth / ( sqrt( 2 * PixelsPerTri ) )
```

- Issues
  - Long thin triangles will over tessellate
- Calculate a sphere bounds on primitive edge
  - Orientation independent
  - Transform to post projection space
    - Scales with Z distance
    - Equates to Percentage of RenderTarget width

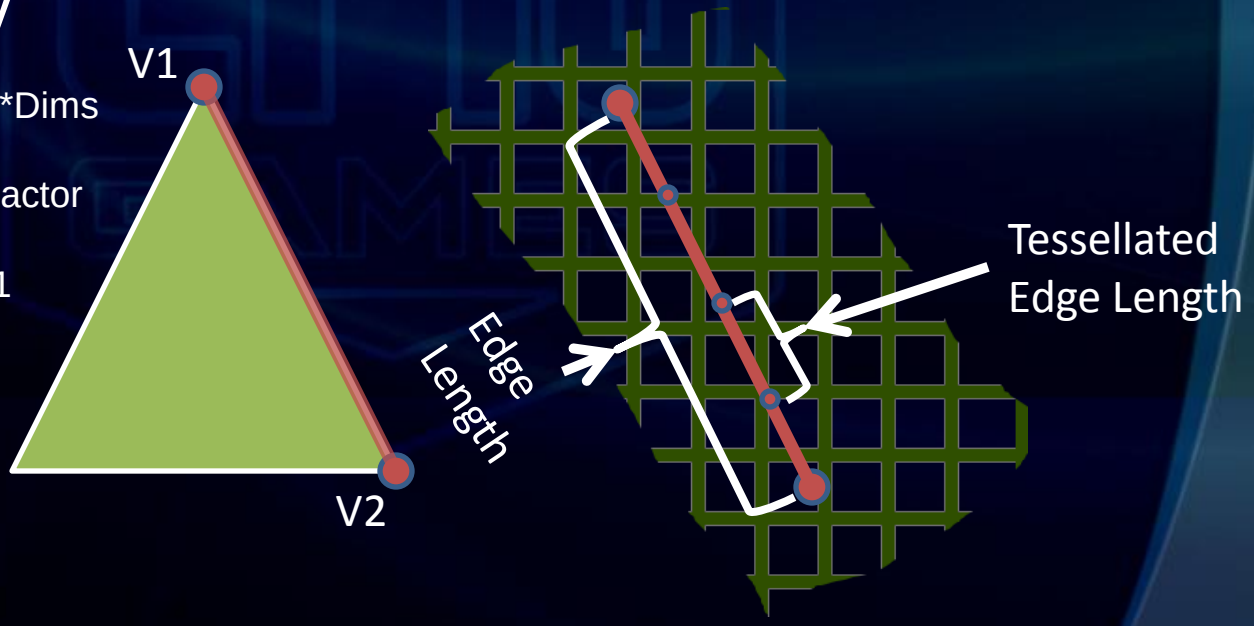
Mesh Swimming & Popping

- fractional_odd or fractional_even
  - Barycentric coords move tangentially on the primitive
  - Slide around on the displacement map
  - Mesh displacements “swim” around
- pow2 or integer
  - Coords “pop” in
  - New coordinate may pick up new detail

Displacement Map MIPing

- Generate MIP maps for the displacement map
- Select mip level manually
  - tex2dlod
  - Align frequency of edge tessellation to texel frequency

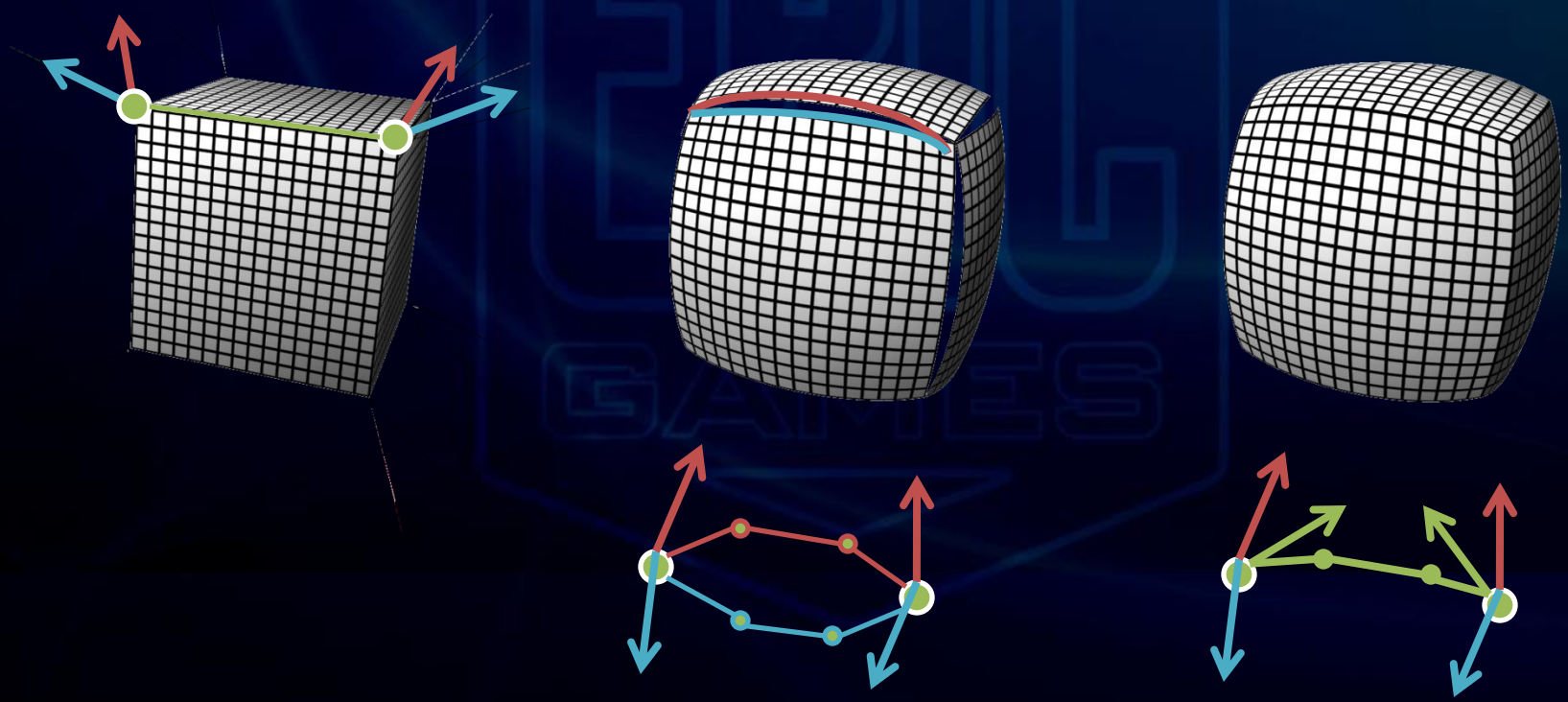
1. L = len(V2.UV – V1.UV)*Dims
2. SubL = L / TessellationFactor
3. mipLOD = Log2(SubL)-1



NVIDIA

PN triangles and cracking

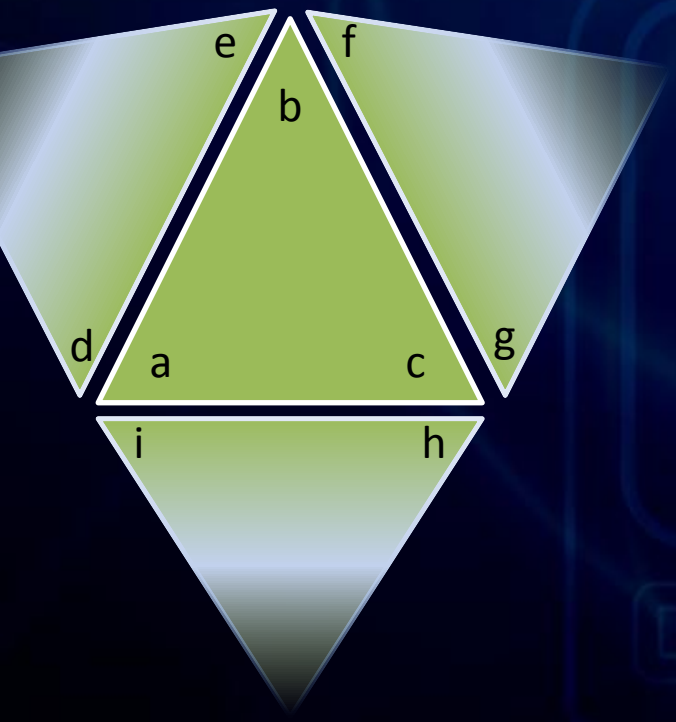
- Non “smooth” normals result in cracks
- Calculated spline control points diverge



PN Triangles

PN Triangles w/ AEN

PN-AEN



Index Buffer

a	b	c	d	e	f	g	h	i

- AEN = Adjacent Edge Normals
- New Index Buffer
  - Contains edge neighbors
- Average normals across edges
  - Ensures crack-free along edges
- Drop in fix
  - No change to vertex buffers
  - Automated preprocessing
    - No artist time
  - Library available soon
    - Plan to release on Google code

NVIDIA

Summary

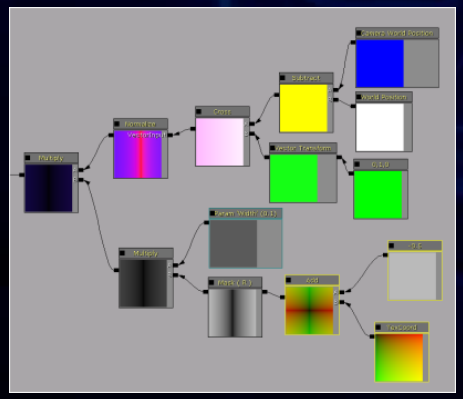
- Tessellation can be used to provide vibrant dynamic effects and smooth mesh LOD
- Dominant Edge/Vertex fixes cracks from divergent UV coordinates
- MIP mapping displacement map can mute mesh swimming
- PN-AEN avoids cracks from divergent normals

Rendering

- Tessellation
- Hair
- Deferred + MSAA
- Subsurface Scattering
- Reflections
- Depth of Field

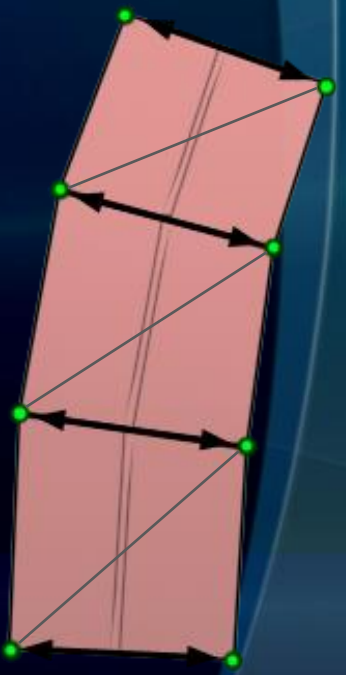
Short Hair / Beard

- Considered many methods
  [Tariq08] [Neulander98] [Assarson09] [Nguyen06] [Neulander01]
- Ended up with camera aligned triangle strips
  - Reuse of existing code (e.g. mesh skinning)
  - Reuse of existing art pipeline
  - Move Vertices in the Vertex Shader (VS)



VS “code” to move the vertices

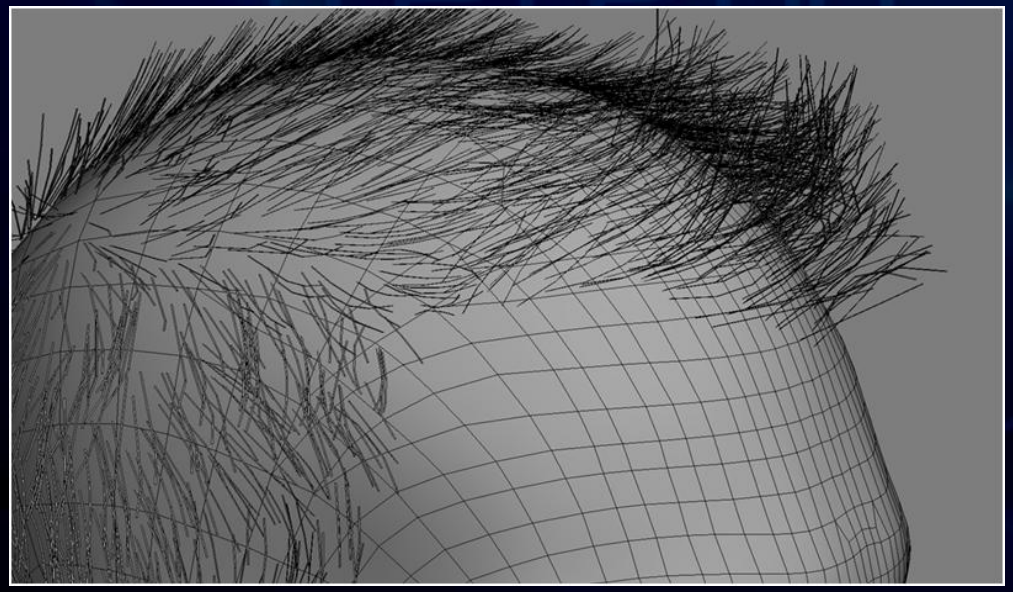
Before VS



After VS

Hair creation

- Generate very thin triangle strips in 3ds Max (Plug-in “Hair Farm”)
- ~5000 splines -> ~16000 triangles
- Texture contains 36 individual hairs



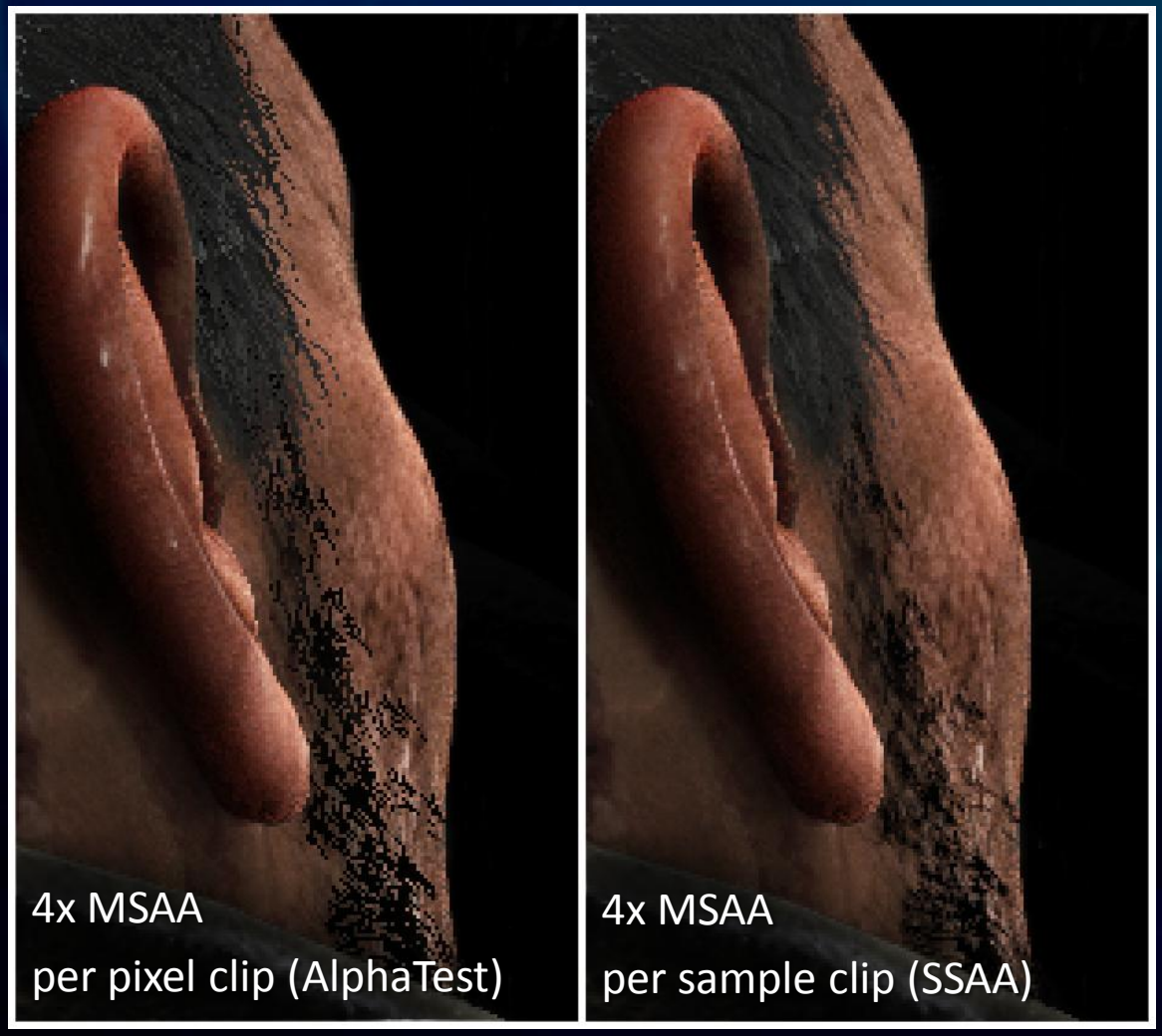
Hair and head mesh in 3ds Max

Texture

Rendering hair

- How to shade the pixel?
  - Alpha Test / clip -> Problems with Aliasing
  - Alpha Blend -> Problems with Sorting, fogging, Depth of Field
  - Alpha To Coverage (A2C) -> Problems with many layers
  - Order Independent Transparency [Gruen10] -> Too many layers?
- Our choice:
  - Render to MSAA buffer
    -> Depth for DOF/Fog/Shadow receiving
  - Stick to binary occlusion (per MSAA sample)
  - SSAA (Alpha Test per MSAA sample)
    -> Anti-aliasing for individual hairs

SSAA


4x MSAA
per pixel clip (AlphaTest)
4x MSAA
per sample clip (SSAA)

Rendering

- Tessellation
- Hair
- Deferred + MSAA
- Subsurface Scattering
- Reflections
- Depth of Field

Deferred Rendering [Hargreaves04]

- UnrealEngine 3 is primarily a forward renderer
- Geometry detail * MSAA * Complex shaders * Many lights
  -> too slow in forward, too many shader permutations
- Added more GBuffer properties
  - Albedo + Specular color, Specular Power
  - Spec + Diffuse normal (Wet material is 2 layered)
  - Subsurface scattering
- Some forward rendering remains (skin, hair and translucency)

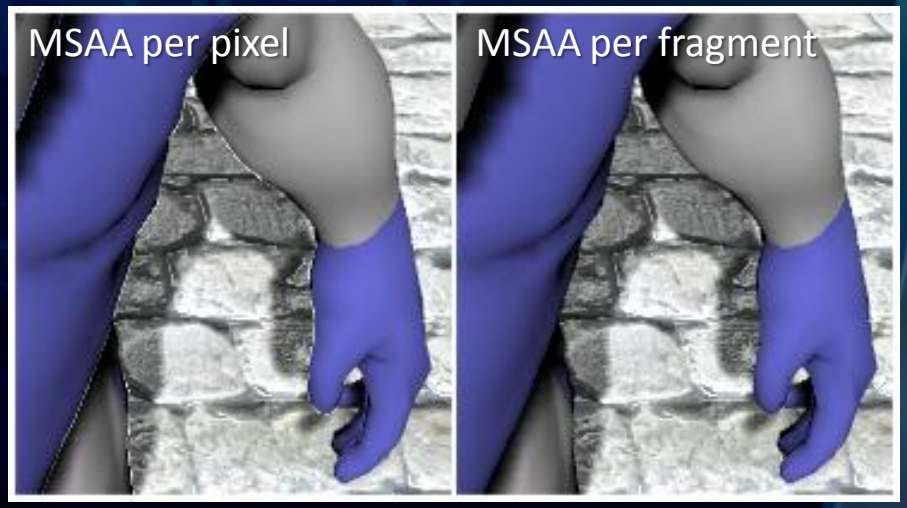
Anti-Aliasing

- 4x MSAA for forward rendering
- Deferred rendering requires special attention
- Per fragment shading only where needed:
  1. Clear stencil, Set stencil write
  2. Pass 1:

     if heuristic(depth/normal) do discard
     otherwise shade per pixel

  3. Activate stencil test
  4. Pass 2:

     shade per fragment



Deferred shadows (notice the bright halo)

Rendering

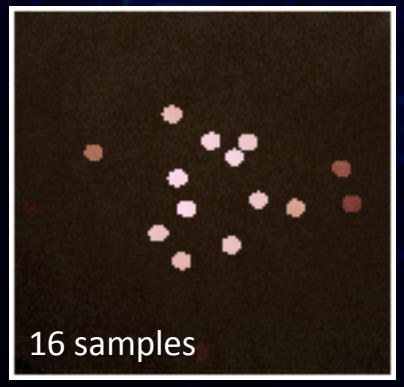
- Tessellation
- Hair
- Deferred + MSAA
- Subsurface Scattering
- Reflections
- Depth of Field

Human skin Subsurface Scattering (SSS)

- Important effect to render believable faces in dynamic lighting
- Many skin layers contribute to the final look
- Human eye is trained to recognize details in faces
- Human skin is a special case that allows approximations

Screen Space Subsurface Scattering (SSSSS)

- Idea is to gather lighting contributions in screen space [Mikkelsen10] [Jimenez09]
- Gather 16 samples in a disc, randomize per pixel and in time
- Artist can define SSS color and world space scatter radius
- Takes Depth and Normal input into account
- Hides shadow sampling artifacts
- Doesn’t work with ear



Rendering

- Hair
- Deferred + MSAA
- Subsurface Scattering
- Reflections
- Depth of Field

Image Based Lighting (IBL)

- To compute incoming light at given position and direction
- How it works?
  - Function that maps position and direction to an image point (5D->2D)
  - Image with HDR content representing all incoming light
- Complex lighting
- Blurry reflections
- Diffuse lighting

Cubemaps

- only far reflections

Planar reflection

- fixed camera position
- fixed reflection plane
- good for dynamic ground reflection

••• [Buerger07]

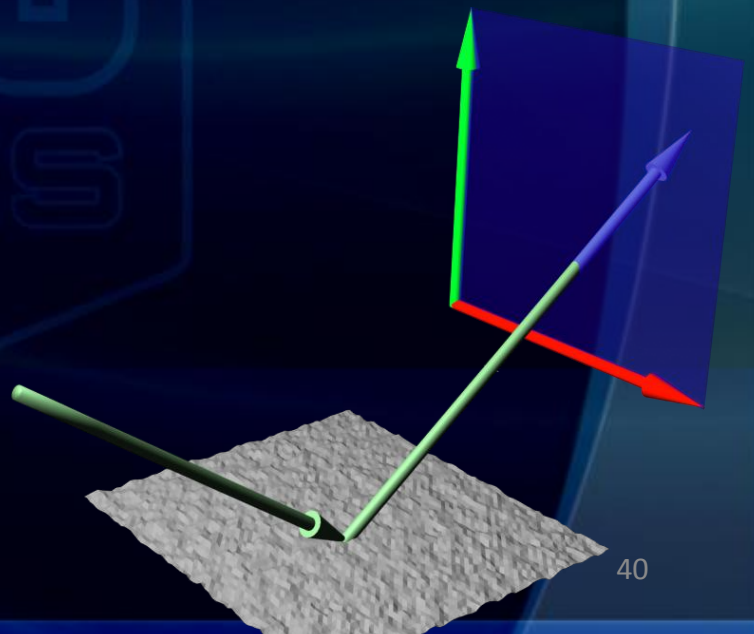
“Billboard reflections”

- Many textured quads (billboards)
- Placement like any other static object
- Can move/rotate/scale dynamically
- No limitations on the reflecting surface

One Billboard reflection

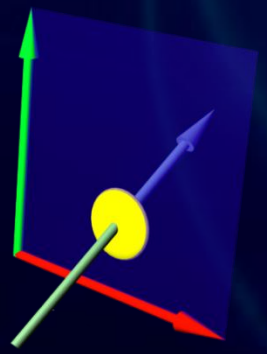
- Each billboard is textured (Color and Alpha for Opacity)
- Ray / quad intersection is simple math
  - Ray start position: surface point we want to shade
  - Ray direction: reflected eye vector

Notice the reflection on the right

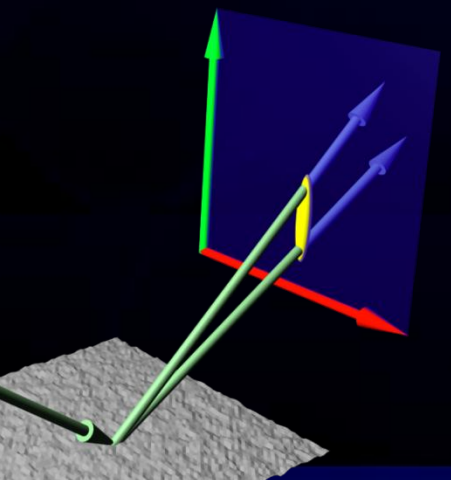


Glossy reflections

- Isotropic reflections



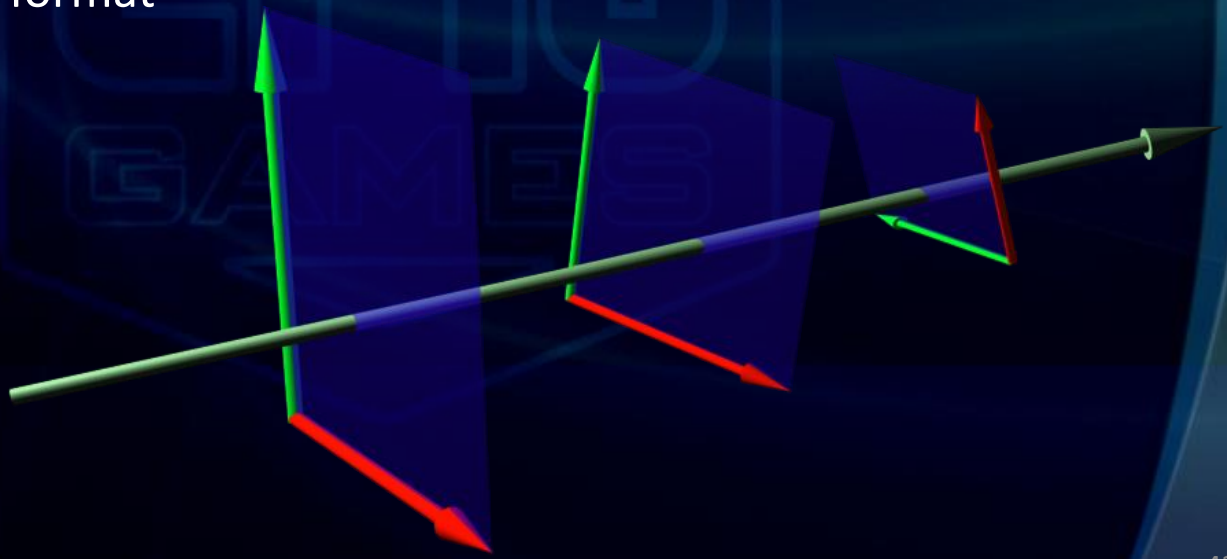
- Anisotropic “lengthy” reflections



Many Billboard reflections

- Many Billboard can occlude each other
- Iterate through all billboards
- Store n (~3) nearest hits (z, color, opacity)
- Composite n layers with alpha blending
- TextureArrays to index a texture in the shader

  -> Same size and format

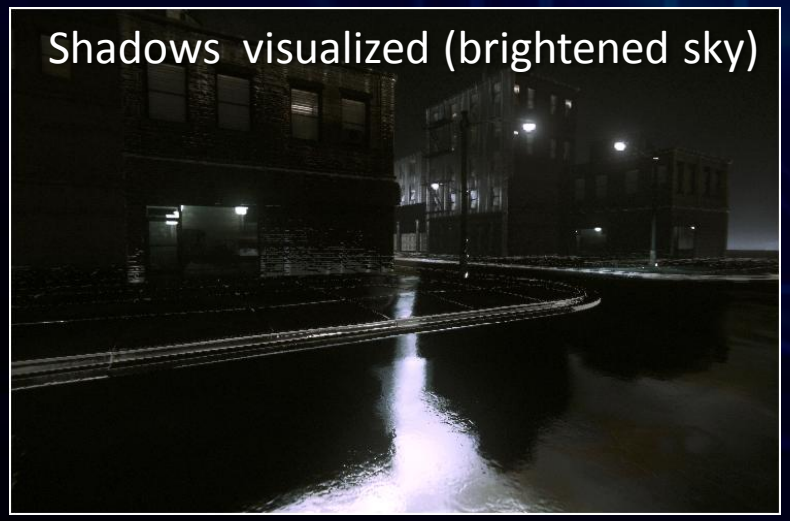


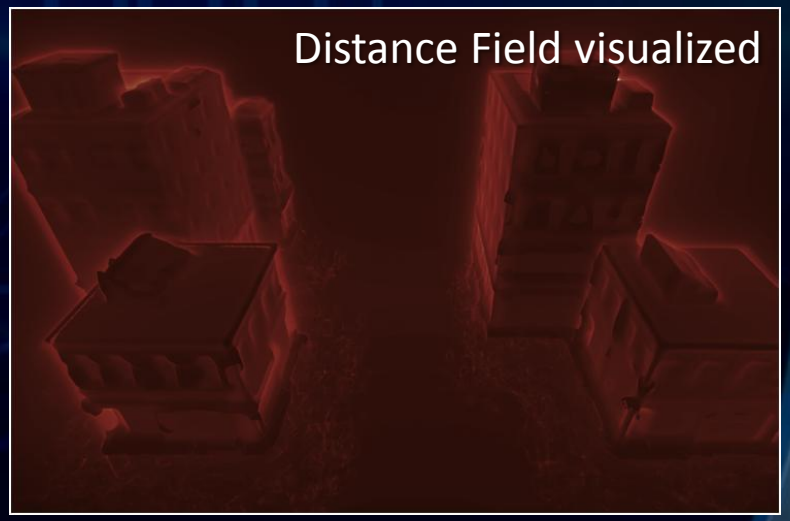
Reflection Shadows

Notice that without shadows light leaks through the building

Static Reflection Shadows

- Ray-casting a distance field
  - Jump over empty areas
  - Stored in a volume texture
  - Distance also allows arbitrary blurred shadows
  - Half resolution (bilateral up-sampling) [Shopf09] [Tomasi98]





Dynamic Reflection Shadows

- Crucial for grounding objects
- Method assumes single plane reflection (ground)
- We generate an image from the reflected eye position (similar to planar reflections), storing depth
- Final mask is generated by rendering quads for each occluding Texel
- The quad size is computed from the stored depth

Point Light Reflections

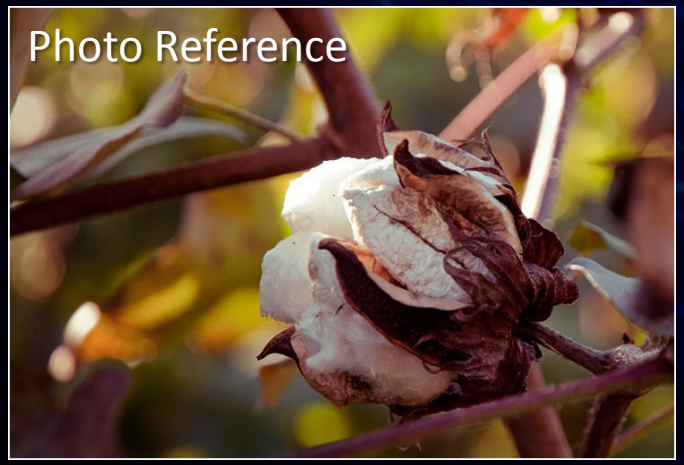
- Phong or Blinn-Phong specular wasn't giving the look we wanted
- Anyone have a "wet street BRDF" ?
- We added a new specular type
  - More "lengthy"
  - Shadowed like Billboard reflections
  - Energy preserving [CodeItNow09]
  - Distance attenuated but not distance bound

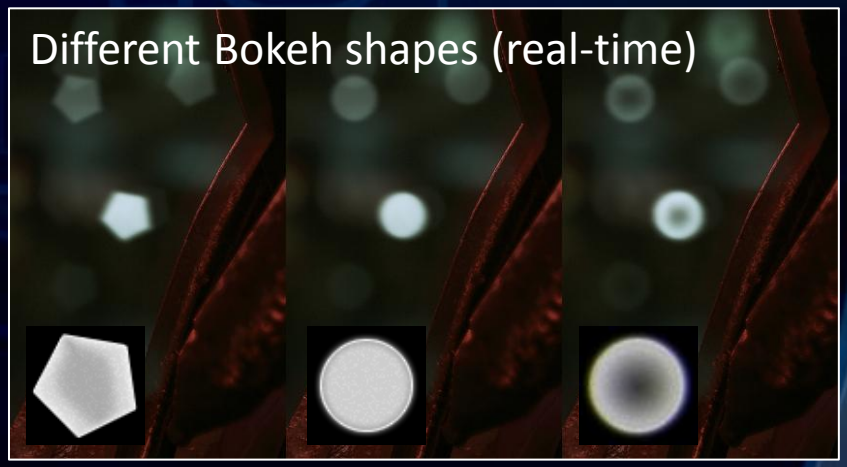
Rendering

- Tessellation
- Hair
- Deferred + MSAA
- Subsurface Scattering
- Reflections
- Depth of Field

What is Bokeh?

- Bokeh is the name of the shape that can be seen in photos or movies when objects are out of focus.
- Contributes to the filmic look
- Shape depends on the camera and lens
- Many Depth of Field algorithms blur objects out of focus without the desired shape. [Lefohn10]





Bokeh Depth of Field

Render a Bokeh textured quad for each pixel
[LostPlanetD3D10][3DMark]

- Quad size and opacity depends on the Circle of Confusion (CoC) radius
- CoC radius is computed from the pixel depth
- Accumulate pixel color and opacity weighted by the Bokeh texture
- Splitting the content into layers avoids occlusion artifacts

Foreground (blurred)

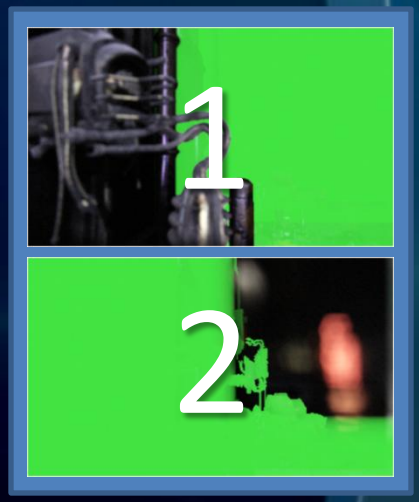
In Focus (Full Resolution)

Background (blurred)

Bokeh Depth of Field Rendering

Scatter pass (Geometry Shader):

- Setup viewports to render to background/foreground layers
- For each pixel:
  - Compute the Circle of Confusion (CoC) radius
  - Compute viewport (foreground / backgound)
  - Setup a quad with the Bokeh texture (RGB: Bokeh*scene color, A: Bokeh)
  - Render quad with additive blending



Render Target with two viewports

Resolve pass:

- Reconstruct the layer color (RGB divided by A)
- Blend layers by the accumulated occlusion (background, in focus, foreground)

Bokeh Depth of Field Optimizations

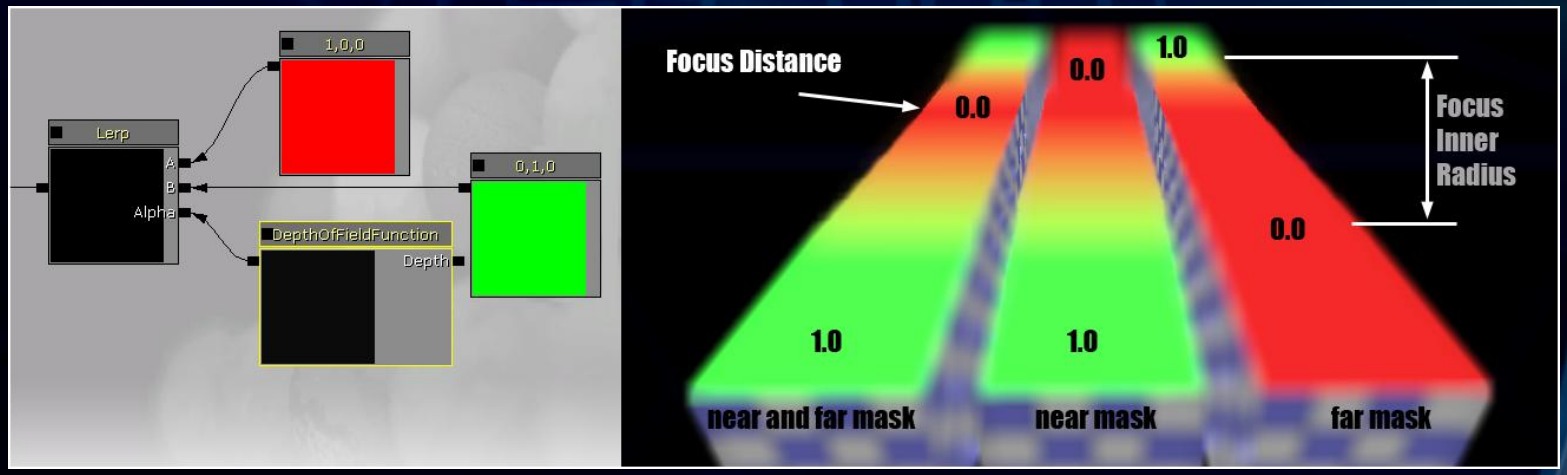
- Vertex / Triangle count:
  - Input image is the half resolution scene (Color + Depth)
- Fill rate:
  - Input image is the half resolution scene (Color + Depth)
  - Output image is half resolution and recombined later with full resolution
  - For each 2x2 input block:
    depending on heuristic (CoC radius, color and depth difference),
    spawn 1 or 4 quads (GS)

Red: 4 quads

Green: 1 quad

Bokeh Depth of Field: Translucency

- Problem: Fog / particles / smoke / lens flares
- Make some effects not affected by Depth of Field
  - Artists can specify which material
  - Composed after/without Depth of Field
- New Shader graph node
  - To give artist control (fade out or blend to blurry version)



References 1/2

- [Buerger07] GPU Rendering of Secondary Effects
  http://wwwcg.in.tum.de/Research/data/Publications/vmv07.pdf
- [Tariq08] Real-Time Hair Rendering on the GPU
  http://developer.nvidia.com/object/siggraph-2008-hair.html
- [Lauritzen10] Deferred Rendering for Current and Future Rendering Pipelines
  http://visual-computing.intel-research.net/art/publications/deferred_rendering/
- [CodeItNow09] Energy Conservation In Games
  http://www.rorydriscoll.com/2009/01/25/energy-conservation-in-games
- [Lefohn10] Advanced Real-Time Depth of Field Techniques
  https://graphics.stanford.edu/wikis/cs448s-10/FrontPage?action=AttachFile&do=get&target=CS448s-10-10-depthOfFieldForWeb.pdf
- [Jimenez09] Screen-Space Perceptual Rendering of Human Skin
  http://giga.cps.unizar.es/~diegog/ficheros/pdf_papers/TAP_Jimenez_LR.pdf
- [Shopf09] Mixed Resolution Rendering
  http://developer.amd.com/gpu_assets/ShopfMixedResolutionRendering.pdf
- [Tomasi98] Bilateral Filtering for Gray and Color Images
  http://www.cs.duke.edu/~tomasi/papers/tomasi/tomasiIccv98.pdf
- [Hargreaves] Deferred Shading
  http://read.pudn.com/downloads160/sourcecode/game/724029/DeferredShading.pdf

References 2/2

- [Gruen10] OIT and Indirect Illumination using DX11 Linked Lists http://developer.amd.com/gpu_assets/OIT%20and%20Indirect%20Illumination%20using%20DX11%20Linked%20Lists_forweb.ppsx
- Robust Multiple Specular Reflections and Refractions http://http.developer.nvidia.com/GPUGems3/gpugems3_ch17.html
- [Neulander01] Hair Rendering (Ivan Neulander, Rhythm & Hues Studios) http://www.rhythm.com/~ivan/hairRender.html
- [Nguyen06] GPU Gems2: Chapter 23. Hair Animation and Rendering in the Nalu Demo http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter23.html
- [Assarson09] Siggraph 2009: GPU Primitives-Case Study: Hair Rendering http://s09.idav.ucdavis.edu/talks/07-Ulf-GPU-Prims-and-Hair-course-slides.pdf
- [Neulander98] Rendering Generalized Cylinders with Paintstrokes http://www.rhythm.com/~ivan/pdfs/gi98.pdf
- [LostPlanetD3D10] Lost Planet D3D10 Parallel Rendering http://meshula.net/wordpress/?p=124
- [3DMark] 3DMark11 Whitepaper http://www.3dmark.com/wp-content/uploads/2010/12/3DMark11_Whitepaper.pdf
- [Buerger07] GPU Rendering of Secondary Effects http://wwwcg.in.tum.de/Research/data/Publications/vmv07.pdf
- [Mikkelsen10] Cross Bilateral Filters for Skin Shading http://jbit.net/~sparky/subsurf/cbf_skin.pdf

Thanks

- Our partner: NVIDIA
- NVIDIA:

  Johnny Costello, Bryan Dudash, Jon Jansen, Ignacio Llamas, John McDonald, David Schoemehl
- Entire Epic team
- Everyone that contributed to the demo
- Epic:

  Daniel Wright, Andrew Scheidecker, Jordan Walker


EPIC
GAMES
We Are Hiring
www.EpicGames.com/jobs

Questions?

- Is this a game?

  No. This is just a technology demo.

- Is that in UnrealEngine 3?

  These features are available now to UE3 licensees and will be in the March UDK.

”Realistic and Interactive Clothing in Epic Games Samaritan Demo Using NVIDIA APEX” Thursday 4:30- 5:30 Room 110, North Hall

NVIDIA @ GDC 2011

CAN'T GET ENOUGH? MORE WAYS TO LEARN:

NVIDIA GAME TECHNOLOGY THEATER

Fri, March 4th @ NVIDIA Booth

Open to all attendees. Featuring talks and demos from leading developers at game studios and more, covering a wide range of topics on the latest in GPU game technology.

MORE DEVELOPER TOOLS & RESOURCES

Available online 24/7 @ developer.nvidia.com

NVIDIA Booth
South Hall #1802

Details on schedule and to download copies of presentations visit

www.nvidia.com/gdc2011

Bonus Slide

- Barely documented but very useful:
  - HLSL Semantic `SV_SampleIndex`
    Used as input causes the shader to run per MSAA sample.
    Can be used in `texture.Load(float2(u,v),SampleIndex)`
    or `EvaluateAttributeAtSample(Interpolator,SampleIndex)`
  - HLSL Semantic `SV_Coverage`
    uint, MSAA bit mask, PS input and output
- How to index a texture in the shader?
  - 2D Texture Atlas -> Size limits, Border and Precision issues
  - Sample array (D3D9/10/11) -> Only for constant index / unroll able loops
  - Dynamic branching -> Slow
  - Texture array (D3D10/11) -> Same size and format, CPU update performance?
- Energy preserving Specular images (material varies Glossiness):