



Advances in Real-Time Rendering in Games

Dynamic Occlusion with Signed Distance Fields

Daniel Wright

Sr Graphics Programmer at Epic Games



Notes down here

About me

- Daniel Wright
 - Graphics programmer at Epic for 9 years now!
 - Gears of War series
 - Focus on lighting features and rendering optimization

Problem

- No good solutions for general purpose occlusion in dynamic production game scenes
 - Need soft, accurate and incoherent visibility queries
 - Useful for area shadows, sky occlusion, reflection shadowing
 - Solution: Ray Marching Signed Distance Fields



Some existing techniques for dynamic general purpose occlusion with their limitations:

Shadowmaps - inherently coherent, limited area shadows through things like PCSS

SSAO – missing information on edges and due to occlusion

Voxel cone tracing - low resolution, can't represent walls used in architecture with affordable voxel sizes in practice

Precomputed AO volumes - low resolution for self shadowing, difficulty avoiding over occlusion from multiple objects

Inspiration

- Iñigo Quílez's work
 - Ray marching procedural distance functions



- Fast Approximations for Global Illumination on Dynamic Scenes
 - Alex Evans 2006

UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

History

- Used SDF (Signed Distance Field) ray marching for reflection shadowing in 2011 Samaritan demo (Unreal Engine 3)
 - Single volume texture and static scene
 - Required high end hardware



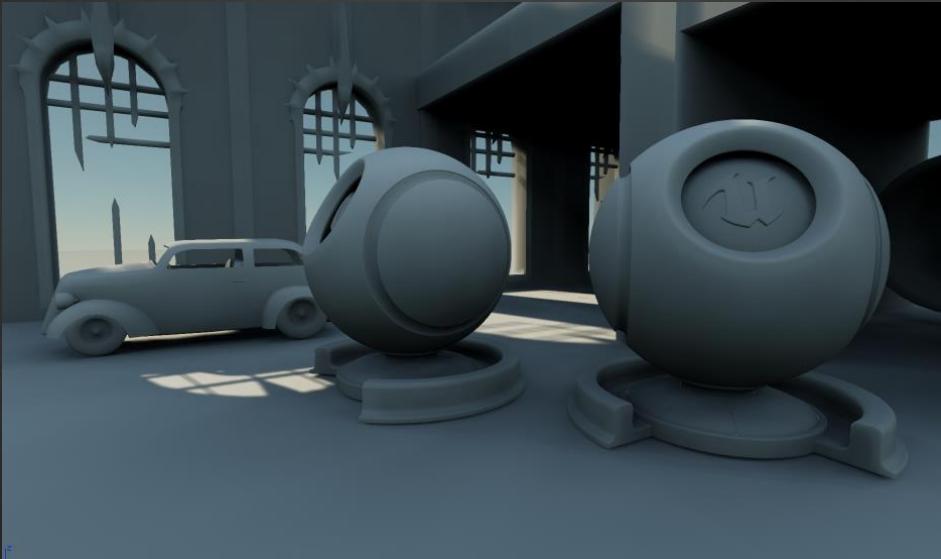
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

The reflection method made for the UE3 Samaritan demo used textured quads and analytical light sources, with SDF traced shadowing to handle geometry not represented by the textured quads. It handled varying roughness but the implementation was not general purpose as it precomputed a single volume texture for the whole scene.

History

- Early dynamic Sky Occlusion quality prototype
 - Per pixel cone march against every intersecting object
 - 27 cones
 - Ran at 2fps in a tiny scene!



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

History

- GDC 2015 Kite tech demo
 - Huge world with fully dynamic lighting using these methods
 - 980 GTX



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

The Kite open world demo uses SDF ray marching for distant shadows, sky occlusion and local occlusion of the dynamic GI method (not covered here). It needed a 980 to run but since then we've optimized the sky occlusion and shadows down to current gen consoles.

Present

- Fortnite - in development at Epic
 - Dynamic time of day
 - Procedurally generated levels with massive player building using these lighting methods
 - All supported on PlayStation 4-level hardware



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Outline

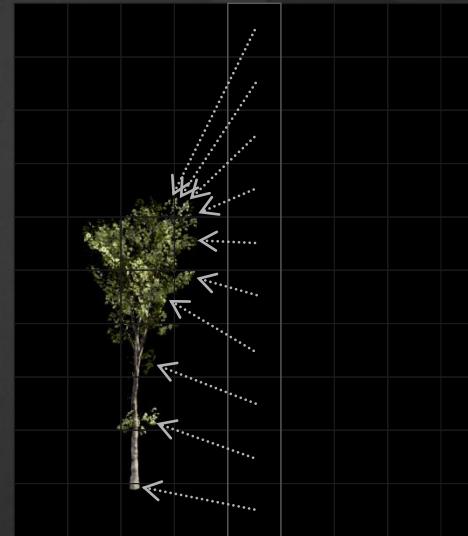
- SDF (Signed Distance Field) recap
- Scene representation
- Direct shadowing
- Sky Occlusion
- Problems and solutions

SDF Recap

- Stores distance to the nearest surface at every point
- Inside regions store negative distance (signed)
- Level set of $d=0$ is the surface



Mesh Distance Field



SDF Recap

- Signed distance interpolates linearly at surface
- Represents any surface orientation accurately, despite grid

Mesh Distance Field



UNREAL
ENGINE

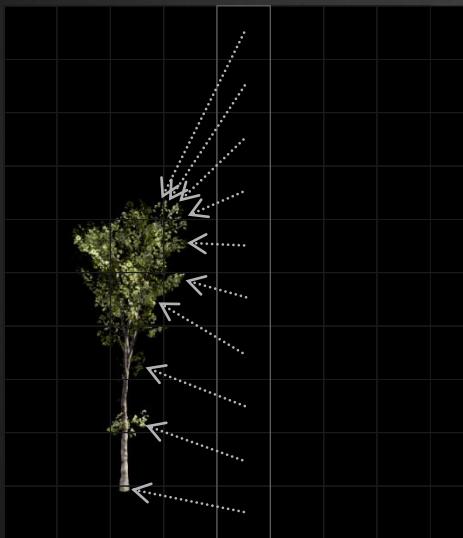
SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Effectively 4x higher resolution in each dimension to represent a surface as a SDF instead of voxels due to the property of interpolating linearly at surfaces, which allows use of hardware filtering.

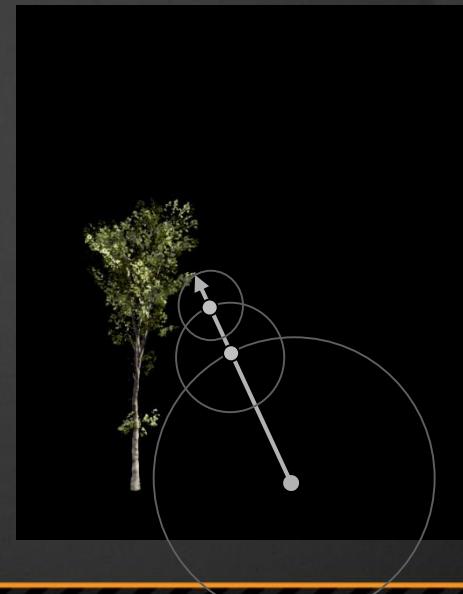
SDF Recap

- Ray intersection skips through empty space based on distance to surface!
 - If the ray intersects, light is shadowed

Mesh Distance Field



Ray march to light



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Sometimes called Sphere Tracing

SDF Recap

- Approximate cone intersection for free!
 - Area shadows

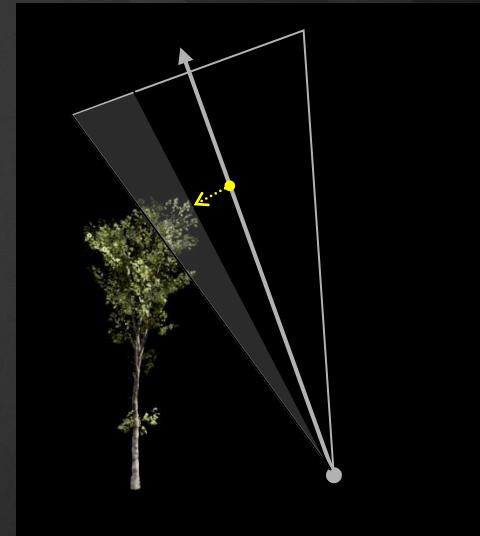
Mesh Distance Field



Ray march to light



Cone intersection



**UNREAL
ENGINE**

By tracking the closest distance a ray passed by an occluder, we can compute an approx cone intersection with no extra cost. This approximation works surprisingly well and is the basis for solving area shadowing and sky occlusion with distance fields.

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

SDF compared to Voxels

- + Represent any orientation surface equally
- + Interpolates linearly
 - + Higher effective resolution
 - + Less biasing to avoid self intersection



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Comparison of a SDF representation of a mesh to voxels. The voxel grid does better with axis aligned geometry and worse with diagonal, but it's all the same for SDF. The SDF has much higher effective resolution, which means you don't need to bias as much against incorrect self-occlusion.

SDF compared to Voxels

- + Skip empty space in tracing
- + Approximate cone intersection without prefiltering



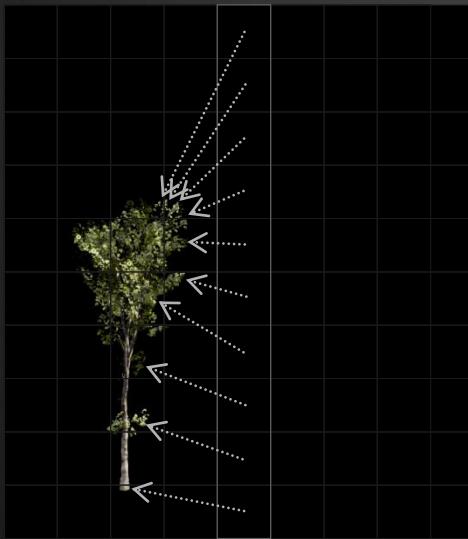
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

The SDF data allows us to skip empty space, while voxels require fixed steps. Voxel representations need to be prefiltered to handle cone intersections efficiently, which adds cost and complexity for scene updates.

SDF compared to Voxels

- Only represent surface position and normal
- Can only get visibility out of a trace



**UNREAL
ENGINE**

However, a SDF can only represent the surface, and therefore we can only get visibility out of a trace, whereas a voxel cone trace can gather any property (incoming lighting for GI)

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

SDF Recap

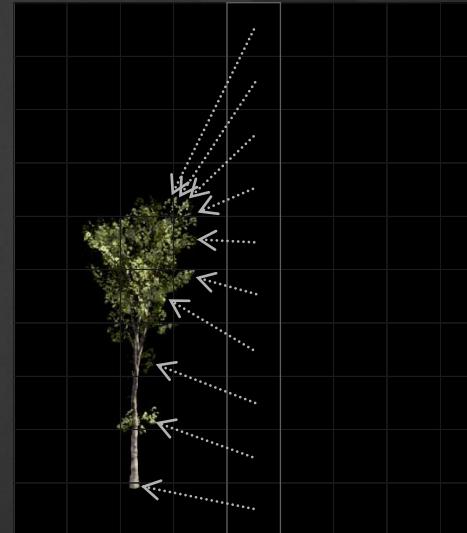
Fundamentally a surface representation instead of a volume

Can only leverage where incoming lighting and visibility can be decoupled

Provides soft, accurate visibility traces in
any direction!

Scene Representation

- Mesh SDF generated offline with brute force triangle raytracing
 - Stored in fp16 volume texture
 - 50^3 enough for average mesh (240KB)
- GPU methods are available for dynamic updates



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

In our implementation we just have an offline method to compute mesh SDFs, by tracing rays in all directions to find the nearest surface. This means it only supports rigid meshes. There are methods to compute SDFs fast enough to do it at runtime.

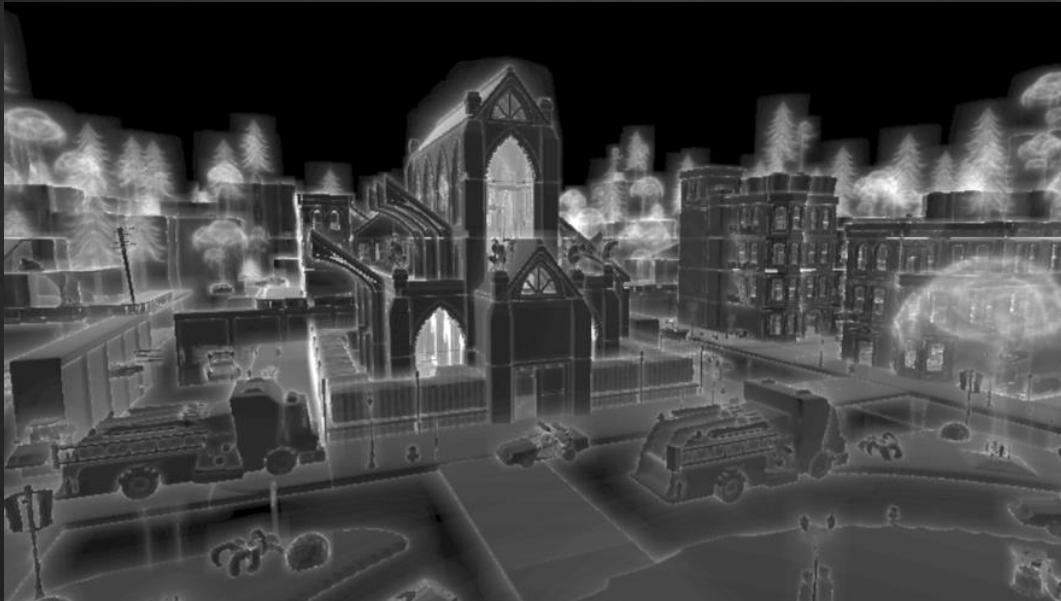
Scene Representation

- SDF needs to be defined everywhere
 - Border added around bounds which is guaranteed to be outside the surface
 - Samples outside the valid area are clamped
 - Composite distance gives approximation



Scene Representation

- Visualization
 - Trace camera rays and visualize the number of steps



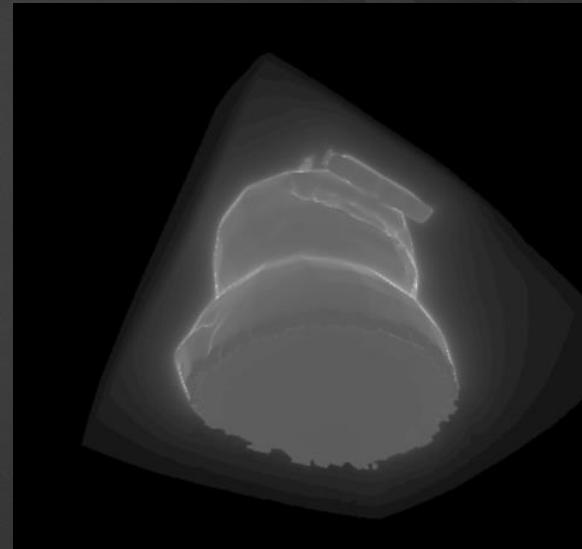
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

This visualization mode is useful for inspecting the SDF version of the scene. White means many steps needed to find intersection. Note that rays at grazing angles to surfaces took many more steps to intersect.

Scene Representation

- Inside / outside determined by backfacing ray hits
 - Doesn't require closed meshes



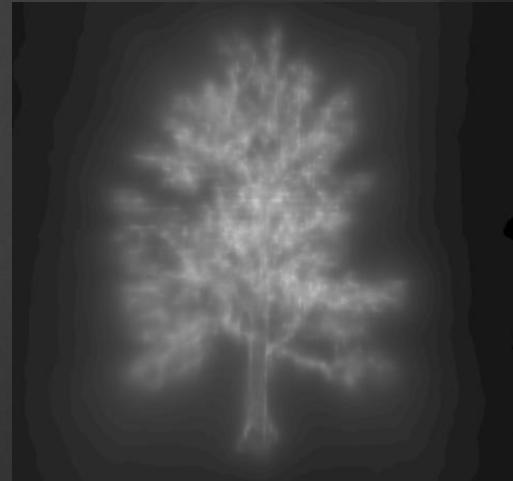
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Computing the sign can be challenging for arbitrary unclosed meshes. A simple heuristic that works well for determining 'inside': after tracing rays to find the closest surface, if the number of backface hits is $> 50\%$ of the rays traced, that position is inside.

Scene Representation

- Partial occluders semi-supported
 - Treated as two-sided surface, which never causes full intersection
 - Works great for foliage
 - Hefty ray marching cost though



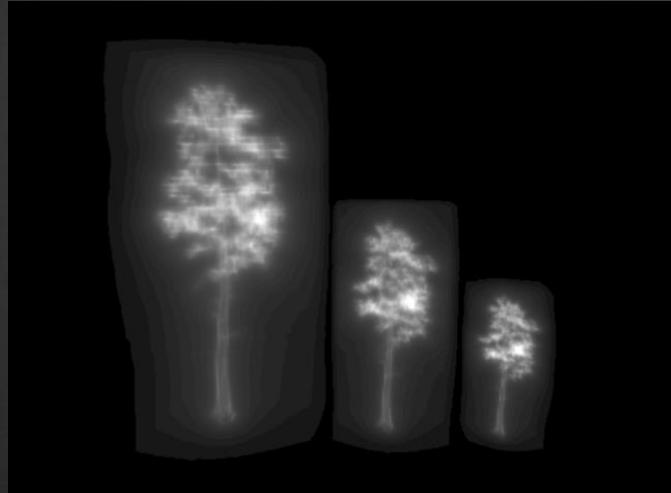
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Partial occluder here means anything with such small holes that it can't be represented as a solid surface.

Scene Representation

- Instancing supported
 - Same SDF data, different transform
 - Only uniform scale in current implementation
 - Default resolution assignment based on asset scale



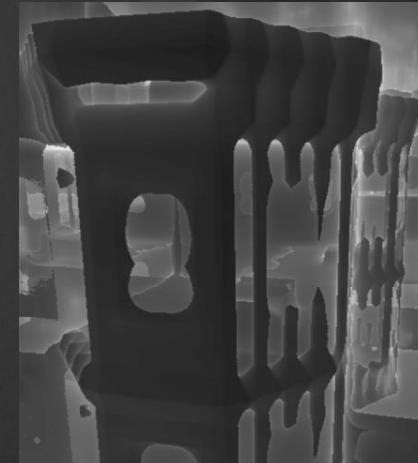
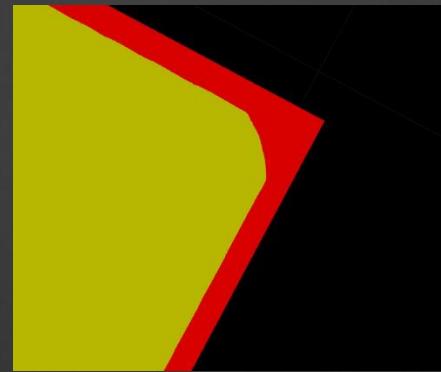
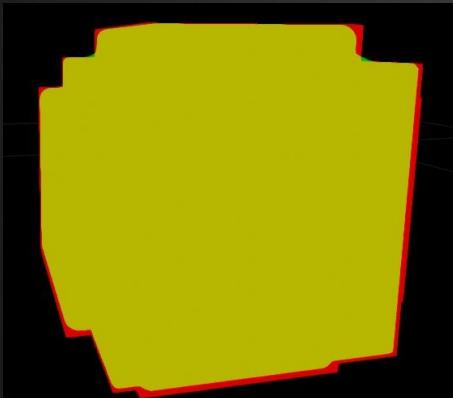
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Because the SDF is computed in local space, it can only handle transforms with uniform scale or mirroring. Perhaps non-uniform scaling could be handled by storing a vector to the nearest surface in local space instead, but that doesn't interpolate properly near surfaces. The SDF is computed for assets, with resolution assigned assuming no scaling per-instance. Artists can increase the asset SDF resolution to handle instances scaling up.

Scene Representation

- Corners are rounded off based on resolution
- Thin surfaces can only be represented with a negative texel inside
 - Necessary for root finding



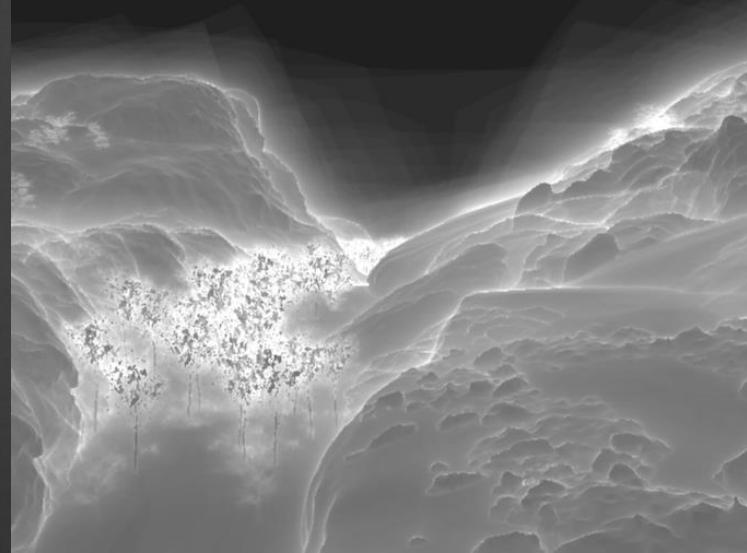
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Even if thin surfaces aren't represented properly, occlusion further from the surface will be accurate, so it's often not noticeable with sky occlusion.

Scene Representation

- Distance field textures are atlased for global scene access while ray marching
 - 300 Mb typical usage for level – 512^3



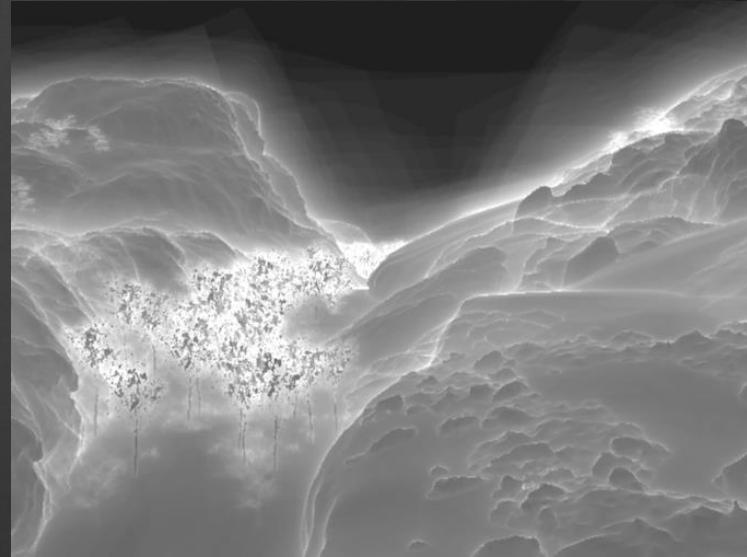
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

The reuse of SDF data through instancing along with only storing it around tight object bounds are the main factors toward keeping the memory usage reasonable in large scenes.

Scene Representation

- Scene objects managed entirely on GPU
 - CPU sends update deltas
 - Moving an object just requires updating a matrix



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

A compute shader is dispatched to handle all object adds, removes and transforms for a frame, which are just operations on a buffer.

Scene Representation

- Culling of objects to view 100x faster than CPU
 - 2 million tree instances in Kite demo -> 50k on screen @ .1ms on PS4
- Possible because all operations on objects down the pipeline are on the GPU

UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

No graphics API calls are used on objects, because no triangles are involved when ray marching the SDF. This allows the object management and culling to be done on the GPU (along with the rest of the technique).

Scene Representation

- Heightfields for terrain
 - Approx cone intersection computed against heightfield
 - Reuses high resolution heightmap
 - Separate pass with results composited



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Scene Representation

- Current representation handles many game scenes
 - Except non-uniformly scaled meshes
 - And skinned / dynamically deformed meshes
 - And large organic meshes / volumetric terrain
- Alternate primitives possible
 - Analytical distance functions
 - Sparse volumetric SDF?

Anything that can be cone traced can be integrated



In the case of direct shadowing, unsupported objects like skeletal meshes can have their shadowing composited into the screenspace shadow mask.

Direct Shadowing

- Sufficient resolution for direct shadowing!



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Direct Shadowing

- Radial lights with sphere source shape (area shadows)
 - Intersecting objects culled to light bounds
 - Then to screen tiles
 - Cone from light containing tile vs object bounds



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Direct Shadowing

- Radial lights with sphere source shape

Foreach intersecting object

 Foreach sample step

 DistanceToSurface = Sample SDF

 Track min visibility (DistanceToSurface / ConeRadius)

 Step along ray toward light by abs(DistanceToSurface)

 Terminate if inside or exceeded max step count

 Shadow = 1 - MinVisibility



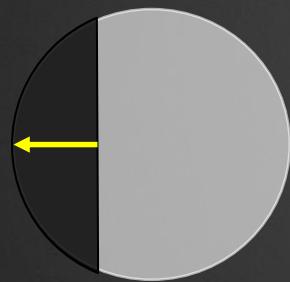
SIGGRAPH 2015

Advances in Real-Time Rendering in Games

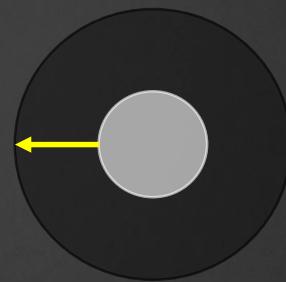
This is the inner loop of the ray marching of object SDFs - it's very simple and the complexity lies in doing as few traces as possible against as few objects as possible. Full source available at <https://www.unrealengine.com/ue4-on-github>

Direct Shadowing

- Cone occlusion heuristics
 - $\text{DistanceToSurface} / \text{ConeRadius} = 1\text{d}$ intersection
 - Found that smarter heuristics gave visually identical results to $(\text{DistanceToOccluder} / \text{SphereRadius})^X$



Cross section



Cross section

**UNREAL
ENGINE**

We stuck with the cheap 1d intersection and applied contrast controls later to make up for the difference to a more accurate intersection. Note that we're only creating the penumbra, not the umbra. This is to avoid using the negative regions of the SDF for occlusion, which would mean solid objects caused more occlusion than thin ones. Small objects on the cone center would cause full occlusion, so these have their occlusion contribution clamped.

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Direct Shadowing

- Directional Lights
 - Intersecting objects culled to max view distance
 - Then into light space grid
 - Same tracing kernel



Direct Shadowing

- Directional Lights
 - Vertex animation can't be represented, use Cascaded Shadow Maps where noticeable



CSM →

UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Direct Shadowing

- Directional Lights

SDF
Traced
Shadows →

CSM →



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Direct Shadowing

- Triangle LODs (billboard) don't match SDF
 - Use conservative depth writes



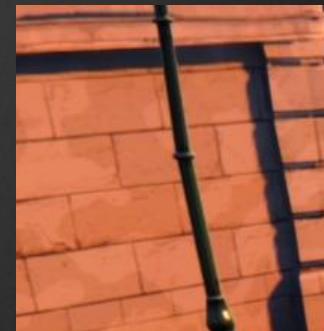
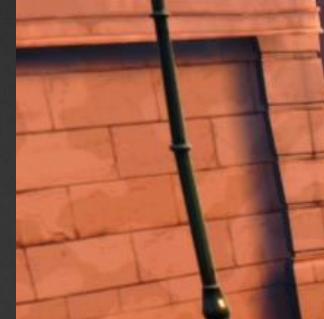
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Triangle LODs become a worse representation for the 3d object in the distance, but the SDF is still accurate. The result is incorrect self shadowing as rays start inside the object and immediately self-intersect against the SDF. To solve this we used conservative depth writes (barely documented d3d11 feature) to add per-pixel depth information to billboards, which maintains HiZ. The per-pixel billboard depth is rendered out along with other attributes like BaseColor. This also improves billboard intersections and other deferred effects like SSAO which operate on the GBuffer depth.

Direct Shadowing

- Why?
 - Area shadows with sharp contacts
 - Controllable self-shadowing artifacts - world space bias
 - No shadowmap aliasing
 - Performance independent of triangle count
 - Dependent on object density and resolution (num traces)
 - Can downsample fairly easily
 - Huge view ranges supported – no CPU cost
 - 30-50% faster than traditional shadowmaps (cubemap / CSM)
 - This depends on how efficient the triangle LODs are



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

The image is a zoom in on a distant building. Top image is CSM, which had to use a bias based on the texel size, losing contact shadows. Bottom image is SDF traced shadows, which only needed a world space bias.

Sky Occlusion Problem

- SSAO sufficient for small scale only, screen space artifacts
- Want general purpose solution for medium-range occlusion (10's of meters)
- Architecture has thin walls

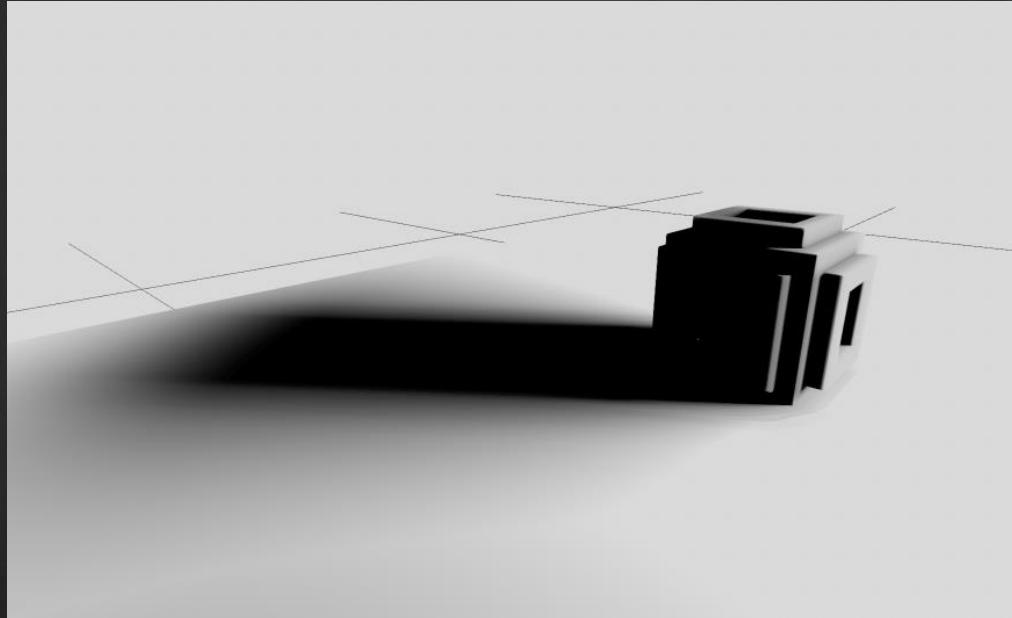


**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Sky Occlusion

- Single SDF cone trace can be soft

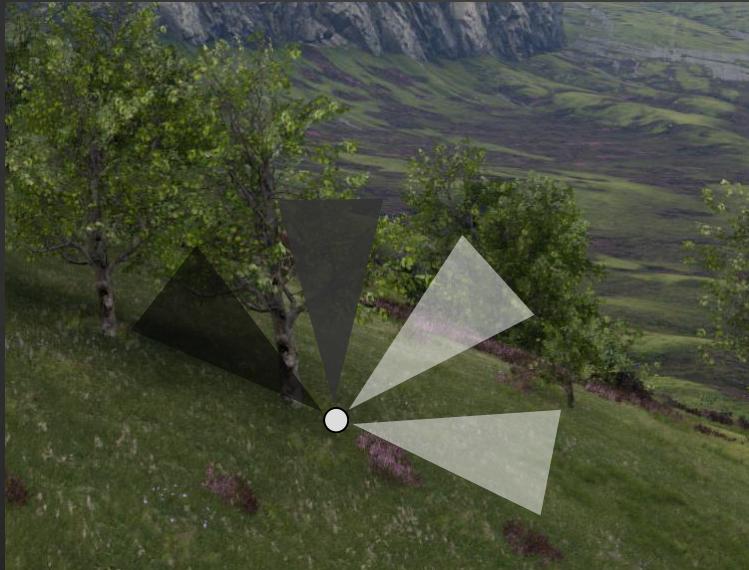


UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Sky Occlusion

- Multiple cones cover the hemisphere – used 9
 - Oriented in hemisphere of the normal
- Occlusion from multiple objects handled with min()



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

9 cones is the lowest we could go and still get good quality sky occlusion. With larger cones, the cone occlusion heuristic breaks down, resulting in over-occlusion from small features near the center of the cone and multiple occluders along the same cone. Cones are oriented in the hemisphere of the normal, which drastically increases quality over world space directions where a flat surface would always be partially self-occluding. This is only possible because we can support incoherent traces.

Sky Occlusion

- Multiple cones cover the hemisphere
 - Oriented in hemisphere of the normal
- Occlusion distance is limited – default 10m



**UNREAL
ENGINE**

MaxOcclusionDistance enforced for performance. Artists can change it based on the frequency of occlusion generally found in the scene.

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Sky Occlusion

- Cone of least occlusion produced (bent normal)
 - Other directional occlusion representations are possible
- Applied to sky light's SH diffuse lighting



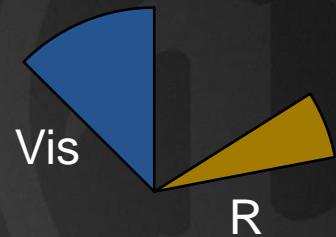
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Combining the cone directions * visibility along each cone produces the cone of least occlusion.

Sky Occlusion

- Cone of least occlusion produced (bent normal)
- Also applied to sky specular using approx cone/cone intersection



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

The limited directionality of the resulting occlusion is sufficient for specular occlusion on contacts.

Sky Occlusion Optimizations

- Objects influence bounds culled to screen tiles
 - Two depth bounds per tile and two culled lists
- Object SDF leveraged to further cull vs tile bounds



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Enforcing the MaxOcclusionDistance allows us to cull objects based on their influence distance.

The bounding sphere of the tile is intersected with DistanceToSurface + MaxOcclusionDistance with a single SDF sample, to determine if the object can affect any pixels in the tile.

Sky Occlusion Optimizations

- Tile culling with the rasterizer found to be faster than tiled compute shader on GCN
 - Biggest win when 10's of thousands of objects
 - Probably applicable to many other use cases (lights, decals, reflection probes)
 - .4ms -> .2ms on GCN
- Build draw arguments buffer – output from Frustum culling
- Draw bounding geometry – DrawIndexedInstancedIndirect
- Pixel shader builds tile intersection lists in UAVs

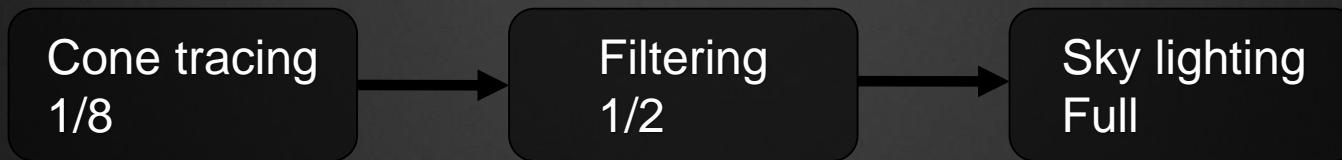
UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

This only matters if not using async compute. Note the bounding geometry has to be expanded to achieve a conservative coverage of the object being culled. Also, the pixel shader should do a more accurate intersection before adding the object to the intersection list.

Sky Occlusion Optimizations

- Cone tracing computed at 1/8th in each dim
 - Severe aliasing without further filtering
- Filtering runs at half res
- Geometry aware upsamples (bilateral)



Sky Occlusion Optimizations

- 4x temporal supersampling (4 jitter positions)
 - Reuse accurate pixel velocities from Temporal Anti Aliasing for reprojection
 - Depth rejection, mark rejected pixels as unstable
 - Spatial fill of temporally unstable areas



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

This is essentially filling new, unstable areas with their stable, blurred surroundings, which is highly preferable to flickering. It's hard to show in a screenshot or video, but works amazingly well in practice for just the cost of a small screenspace gather (5x5 used at half res). Heavy reliance on this temporal filter causes stale occlusion behind moving objects, which could be improved by increasing the convergence speed in those areas.

Sky Occlusion Optimizations

- Sampling object distance fields is expensive

White = 200 object influences



Solution: Composite into a global distance field

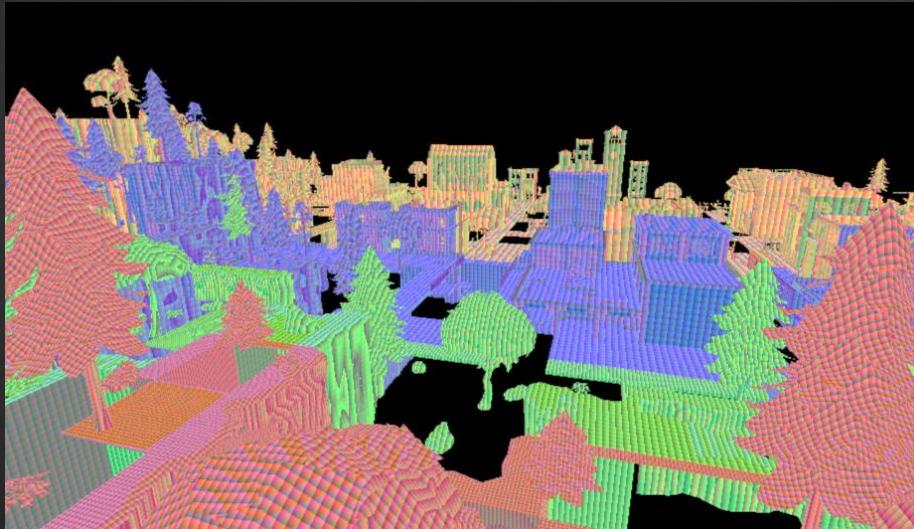
UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

200 overlapping objects = 200x the tracing cost

Global SDF

- Stored in clipmaps centered around the camera – used 4 clipmaps with 128^3 each
- Clipmaps are scrolled with movement
- New slices composited from object SDFs as needed
- Distant clipmaps updated less frequently



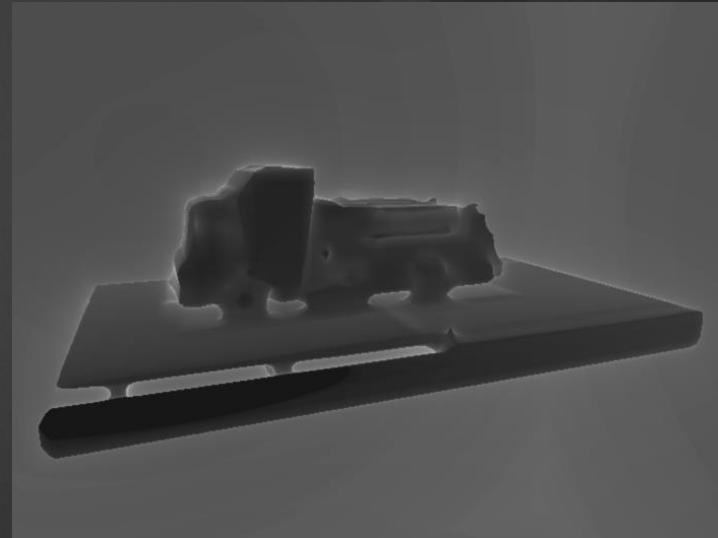
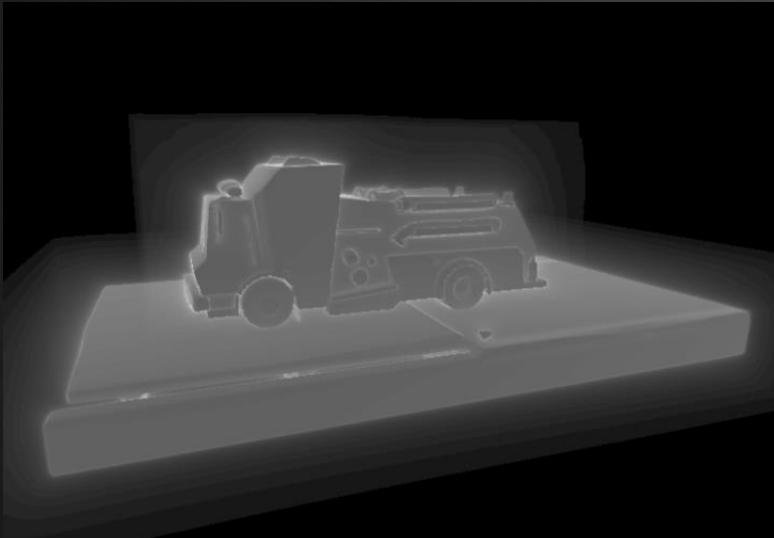
**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

This is essentially a cache of the SDF around the camera, which is updated only where needed (camera movement revealed a new slice, or an object was moved and its influence bounds are dirty), such that average cost of maintaining is close to 0. Worst case update costs are ~7ms on GCN and occur on teleports.

Global SDF

- Global SDF is inaccurate near surfaces
- Sample object SDFs near start of cone, global SDF for the rest



**UNREAL
ENGINE**

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

On the left are the accurate object SDFs, right shows the low res global SDF. By sampling the object SDFs near the start of an occlusion cone, we get the best of both (accurate self-occlusion, efficient long distance traces).

Global SDF

- Massively reduces object influence overlap!

Full influence range



Reduced influence range



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

The effective max object influence distance is massively decreased, this results in performance gains of ~5x for sky occlusion.

Sky Occlusion

- Performance breakdown on Radeon 7870 for 1080p (actual PS4 cost was 3.7ms)
 - 3.44ms DistanceFieldLighting
 - 0.03ms ObjectFrustumCulling
 - 0.23ms ComputeNormal
 - 0.29ms BuildTileList
 - 1.68ms ConeTraceObjects
 - 0.29ms ConeTraceGlobal
 - 0.14ms CombineCones
 - 0.17ms GeometryAwareUpsample
 - 0.31ms UpdateHistory
 - 0.31ms UpsampleAO



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

No heightfields were present in this scene, otherwise their occlusion would be composited in CombineCones. ComputeNormal is just downsampling the GBuffer normal + depth to half res. Object cone tracing continues to be the most expensive part of the algorithm, which is highly dependent on the distance along a cone at which to switch to sampling the global SDF.

Future Research

- Interior occlusion problem
 - Happens when distance falloff intersects with penumbra falloff
 - Individual cones become visible



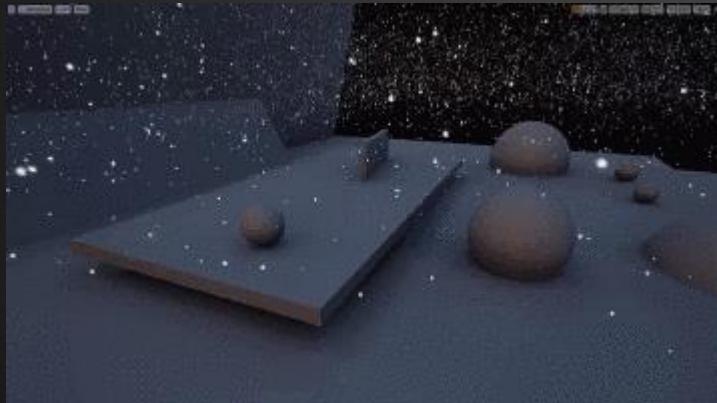
UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

Exposure increased to show the problem clearly

Bonus

- Full scene particle collision
 - Extract candidate collision plane efficiently from SDF
 - Gradient of SDF provides plane normal
 - Same performance as scene depth collision



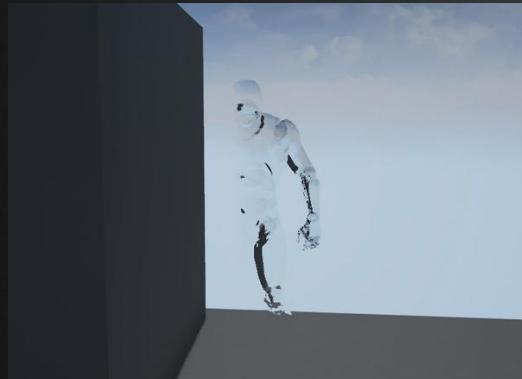
UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

This method of particle collision has a huge advantage over colliding against the depth buffer because it's not limited to what's on your screen, allowing stable effects that accumulate.

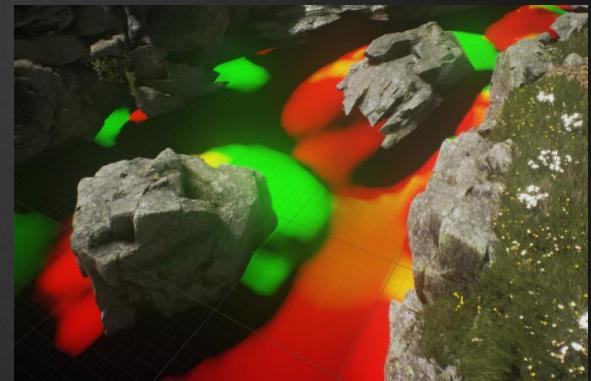
Other uses?

Soft body
approximation



Gameplay
effects
(stealth)

Procedural
flow map
generation



UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

We exposed the global distance field to our data driven material editor in UE4 4.9, and users are already making cool effects.
(images and effects by Roel Bartstra)

Finally

- All of this is available with source in Unreal Engine 4!
 - <https://www.unrealengine.com/ue4-on-github>
- Ray Traced Distance Field shadows doc
 - <https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/RayTracedDistanceFieldShadowing/index.html>
- Distance Field Ambient Occlusion doc
 - <https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/DistanceFieldAmbientOcclusion/index.html>
- Thanks to rendering team at Epic
- Epic is hiring!
 - <http://epicgames.com/>

Any Questions?

UNREAL
ENGINE

SIGGRAPH 2015
Advances in Real-Time Rendering in Games

References

- Ray Marching procedural distance functions: <http://www.iquirezles.org/www/articles/raymarchingdf/raymarchingdf.htm>
- Fast Approximations for Global Illumination on Dynamic Scenes:
http://developer.amd.com/wordpress/media/2012/10/Course_26_SIGGRAPH_2006.pdf
- Unreal Engine 3 'Samaritan' Demo: <http://www.nvidia.com/content/PDF/GDC2011/Epic.pdf>
- Sphere tracing: <http://graphics.cs.illinois.edu/sites/default/files/zeno.pdf>