
Advanced Graphics

Modern Graphics

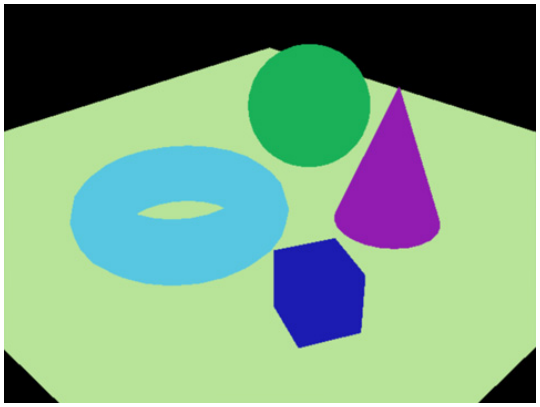
- Today is a high-level survey of techniques
- Focus: detail from shaders
  - Not detail from geometry, computation power increasing faster than fill rate
  - Old multipass algorithms make more sense now as a single pass
- Modern techniques
  - Screen-space computation
    - Reuse other parts of the scene
  - Hybrid raytracing and rasterization
    - Best of both worlds

Forward Shading

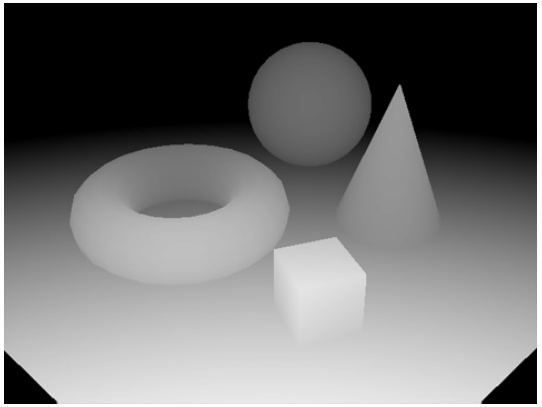
- Forward Shading
  - Traditional rendering method
  - Discover any light interactions with every model
  - Render the model by looping over each light in the pixel shader
- Problems
  - A light touching a small part of a model makes rendering the entire model more expensive
  - Light interactions potentially become $O(n^2)$, causing uneven performance

Deferred Shading

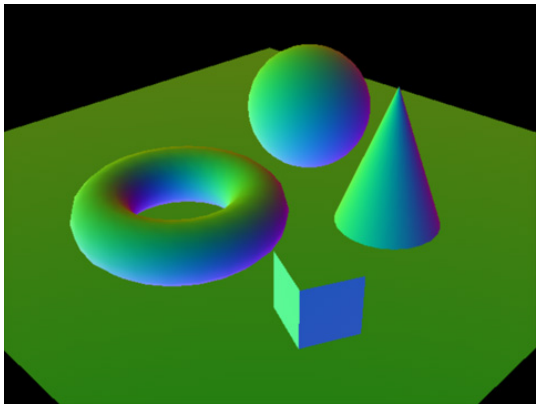
- Idea: decouple lighting and geometry
- Render scene once into multiple buffers
  - Per pixel material, normal, depth, ...
  - Collectively called the G-buffer
  - Can render to multiple render targets at once with modern GPUs



Material color



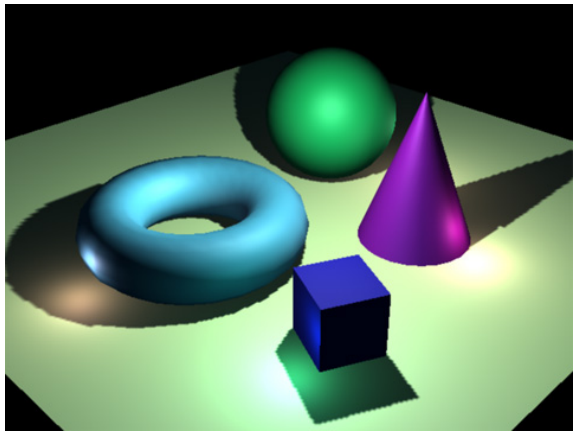
Depth



Surface normal

Deferred Shading

- Layer lighting onto scene by rendering light shapes (i.e. sphere for point light)
  - Each light only rendered to relevant pixels
  - Can recover 3D position from depth and pixel coordinate
  - Use additive blending



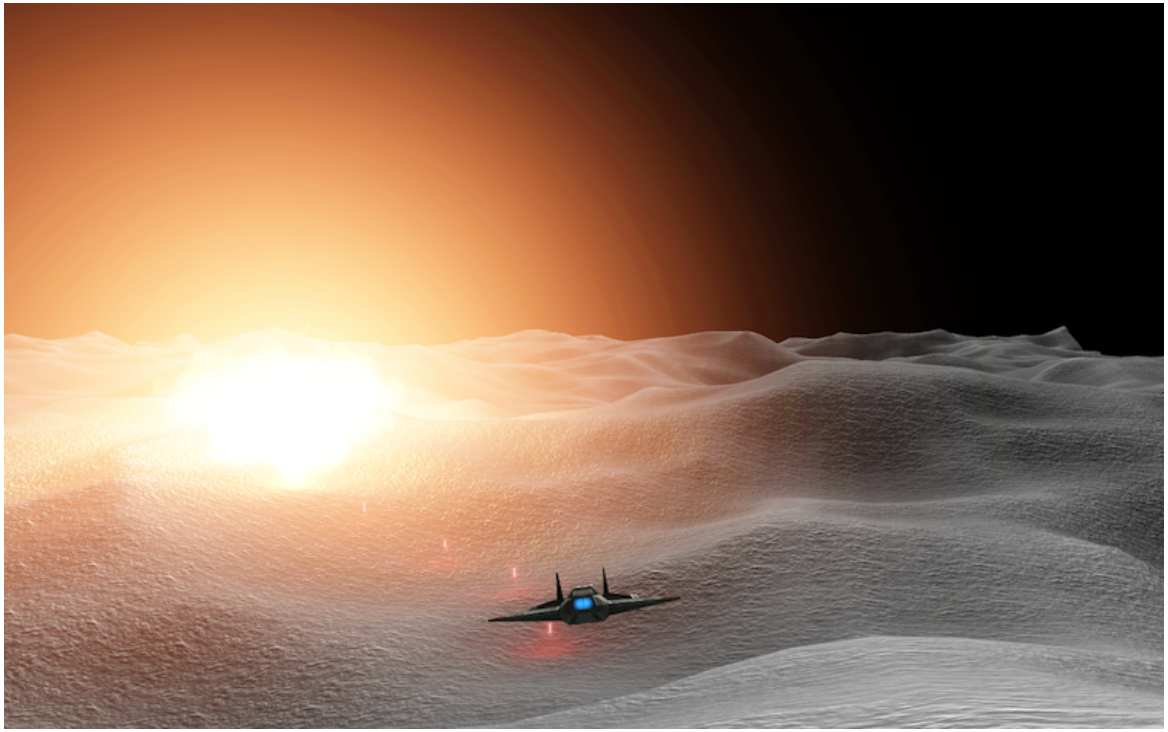
Final render

Deferred Shading

- Benefits
  - Scales much better to large numbers of lights
  - Screen-space effects give consistent performance
  - Allows for many awesome screen-space effects
- Problems
  - Cannot handle transparency (G-buffer can only store information about one pixel)
  - Scenes with many material parameters need large G-buffers, memory bandwidth suffers
  - Cannot be used with hardware anti-aliasing, although modern post-processing edge-smoothing techniques help (MLAA)

Volumetric Effects

- Volumetric glow (fake scattering)

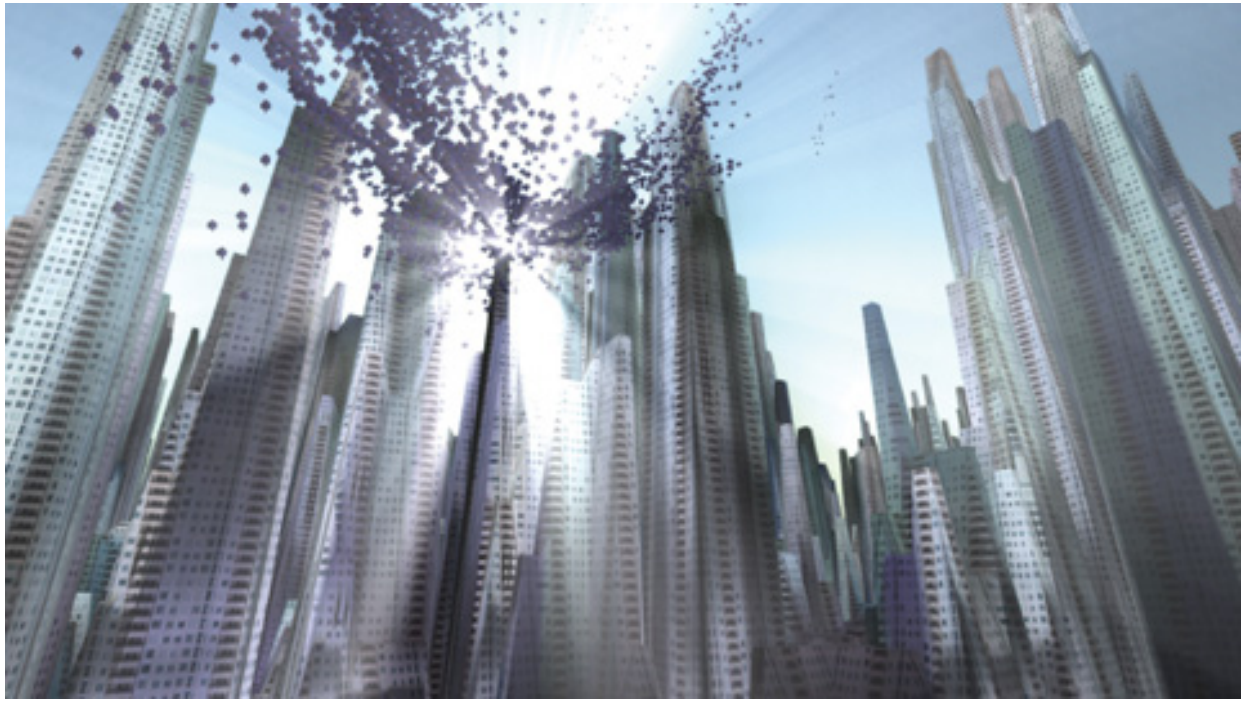


Volumetric Effects

- Volumetric glow (fake scattering)
  - Blend glow color over every pixel
  - Fade off using closest distance from light source to line segment starting from eye and ending at object under pixel
  - Requires deferred shading for position of object
- Rendering to entire screen is expensive
  - Fade off to zero at some radius
  - Only need to draw pixels within that radius in world space, will be cheap for a far away effect
  - Render using inside-out sphere with that radius

Volumetric Effects

- Fake light shaft post process

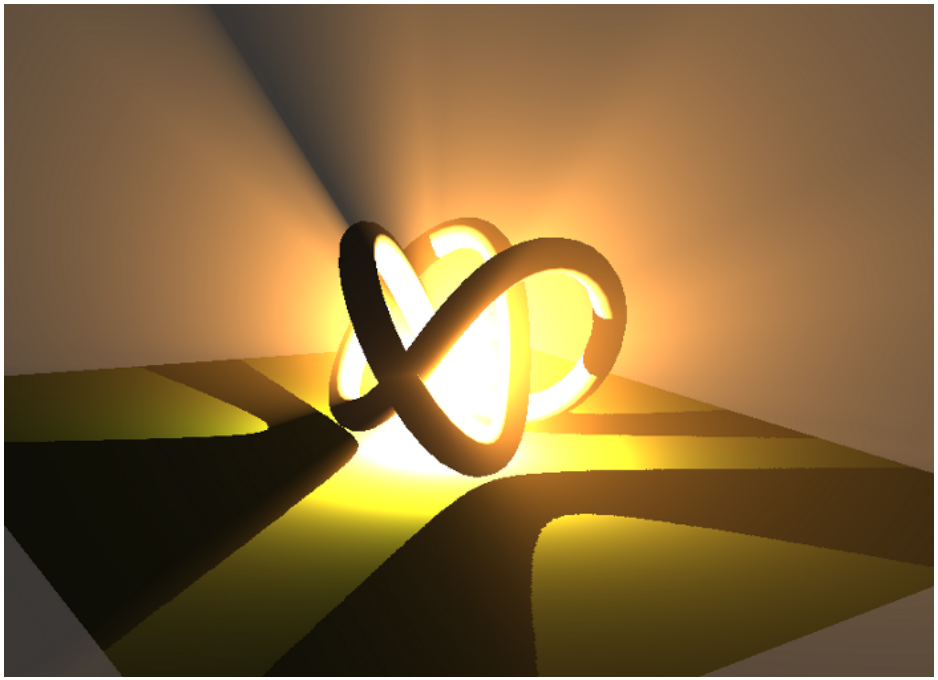


Volumetric Effects

- Fake light shaft post process
  - Render glow around light into separate buffer
  - Render occluding objects on top in black
  - Screen-space zoom blur centered at light
  - Blur with fixed linear steps from pixel to light center
  - Additive blend result on top of regular scene render
- Sample pattern will be visible
  - Random offset to hide sample pattern
  - Or repeat blur multiple times to blur out pattern

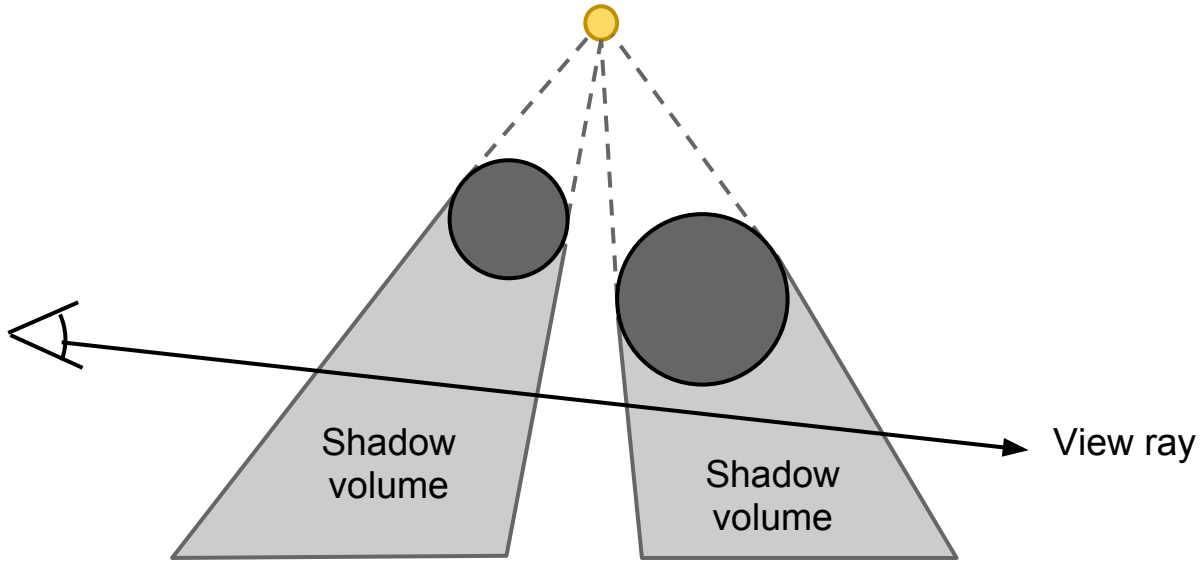
Volumetric Effects

- Real Time volumetric shadows
  - "Real Time Volumetric Shadows using Polygonal Light Volumes" (Billeter, Sintorn, Assarsson)



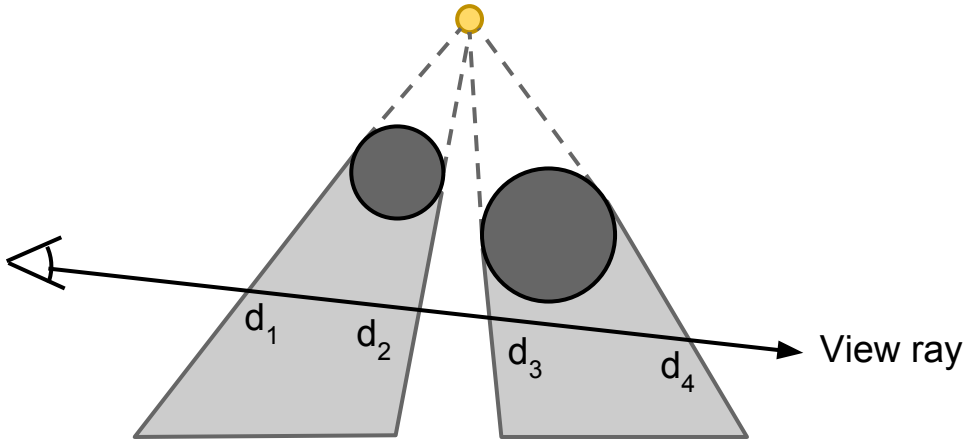
Volumetric Effects

- Real Time volumetric shadows
  - Measure shadow thickness using polygons enclosing shadow volumes
  - Render shadow on top of scene using shadow thickness map



Volumetric Effects

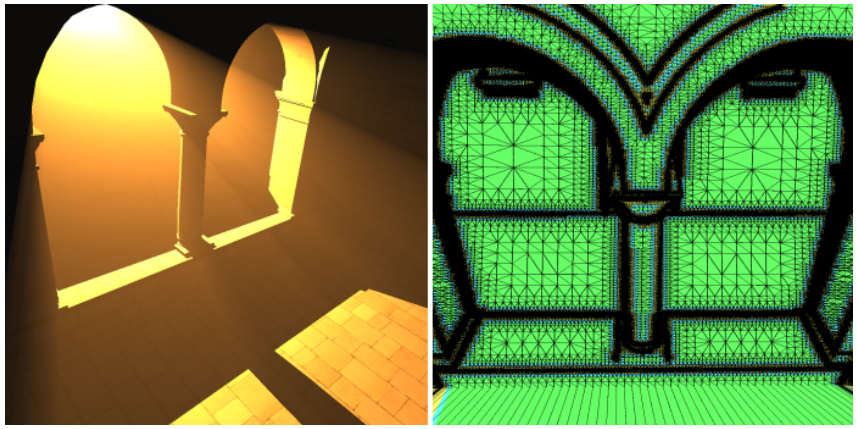
- Real Time volumetric shadows
  - Need two floating-point render buffers, one for front faces and one for back faces
  - Abuse additive blending mode to compute thickness
  - Render depth of front volume faces to first buffer
  - Render depth of back volume faces to second buffer
  - Thickness is second buffer minus first buffer



First buffer: $d_1 + d_3$
Second buffer: $d_2 + d_4$
Thickness: $(d_2 + d_4) - (d_1 - d_3)$

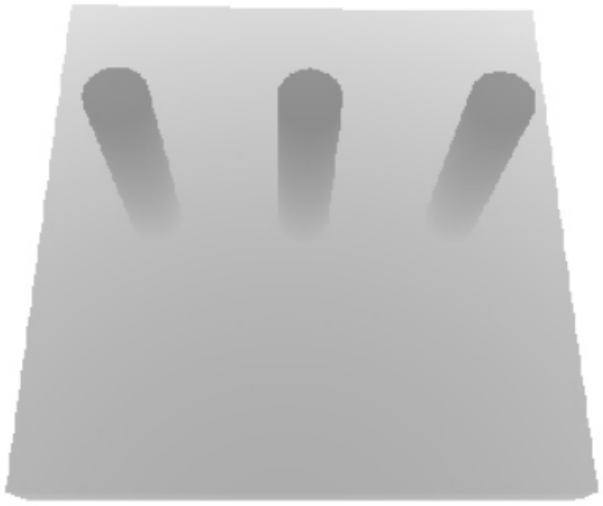
Volumetric Effects

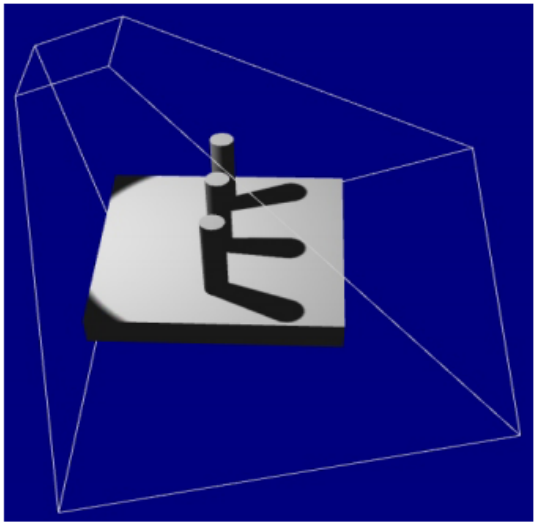
- Thickness is double-counted in overlapping shadow volumes
  - Construct shadow volumes from shadow map every frame, ensures no shadow volumes overlap
  - Can be done efficiently with adaptive tesselation and geometry shaders



Shadow Mapping

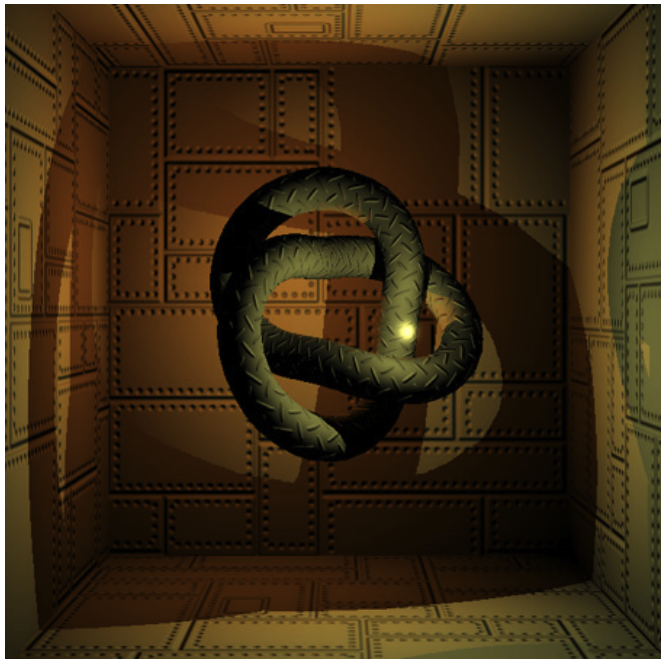
- Need to test whether a pixel is in shadow
  - Render scene from light's point of view
  - Depth map stores closest point to light
  - Render the scene from the camera, projecting each point back into the light's view frustum
  - Shadow if depth of projected point > depth map





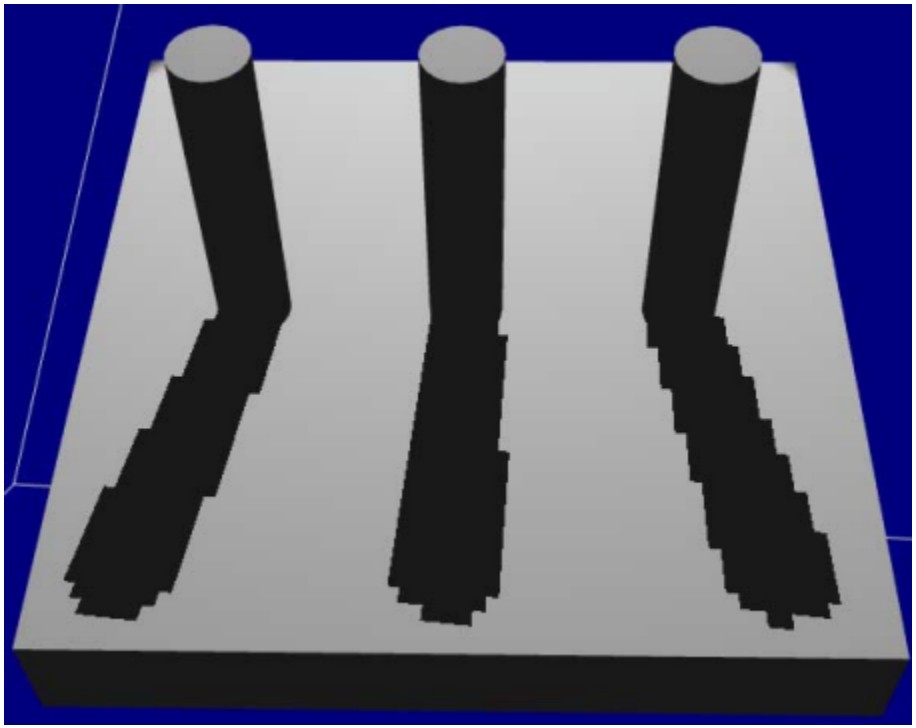
Shadow Mapping

- Need to fit frustum to light
  - Directional light => parallel rays => orthographic
  - Spot light => frustum => perspective
  - Point light => rays in all directions => use cube map



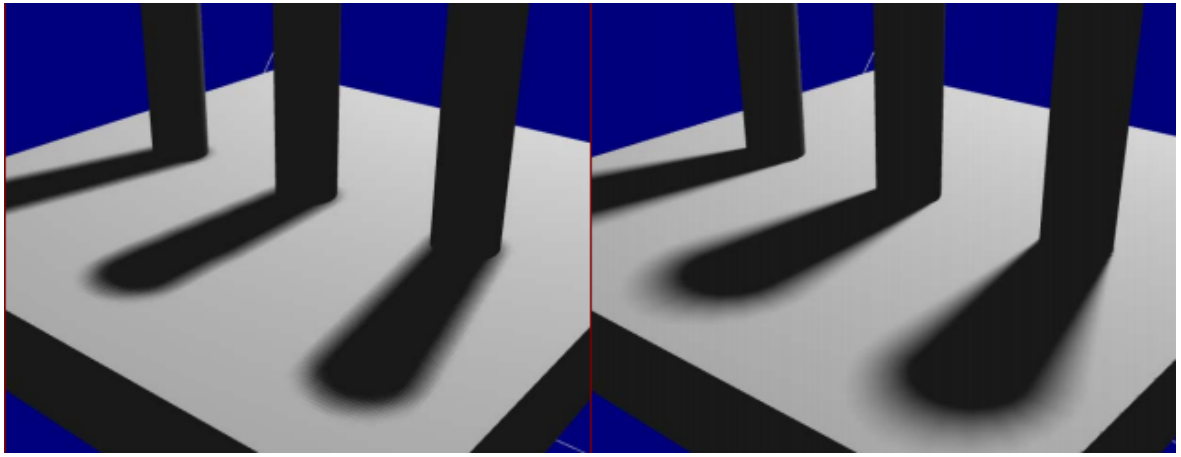
Shadow Mapping

- Problem: Jagged edges
  - Shadow map resolution varies across scene
  - Increasing resolution helps, but uses more memory



Shadow Mapping

- Fix: blur or fuzz out boundaries
  - Multiple nearby shadow tests are made per pixel and are averaged together
  - Called PCF: Percentage Closer Filtering
    - May use randomized sample patterns
    - May use variable blur size since shadows get more blurry away from caster and area lights

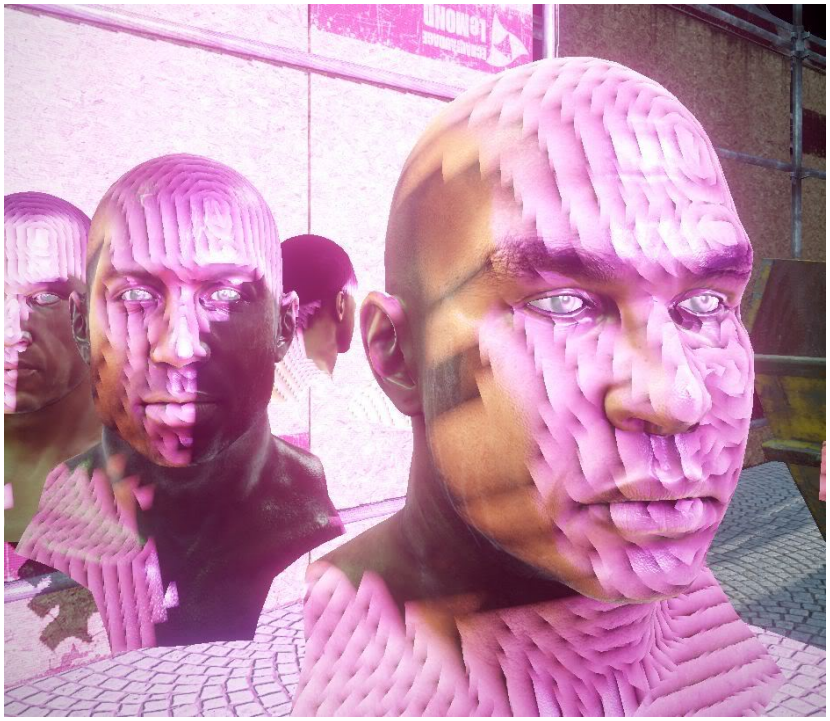


Shadow Mapping

- Fix: Average out over multiple frames
  - Reproject previous frame (if moving camera)
  - Jitter shadow map per frame for more samples
  - Weight by confidence (distance to texel center)
  - "Pixel-Correct Shadow Maps with Temporal Reprojection and Shadow Test Confidence"

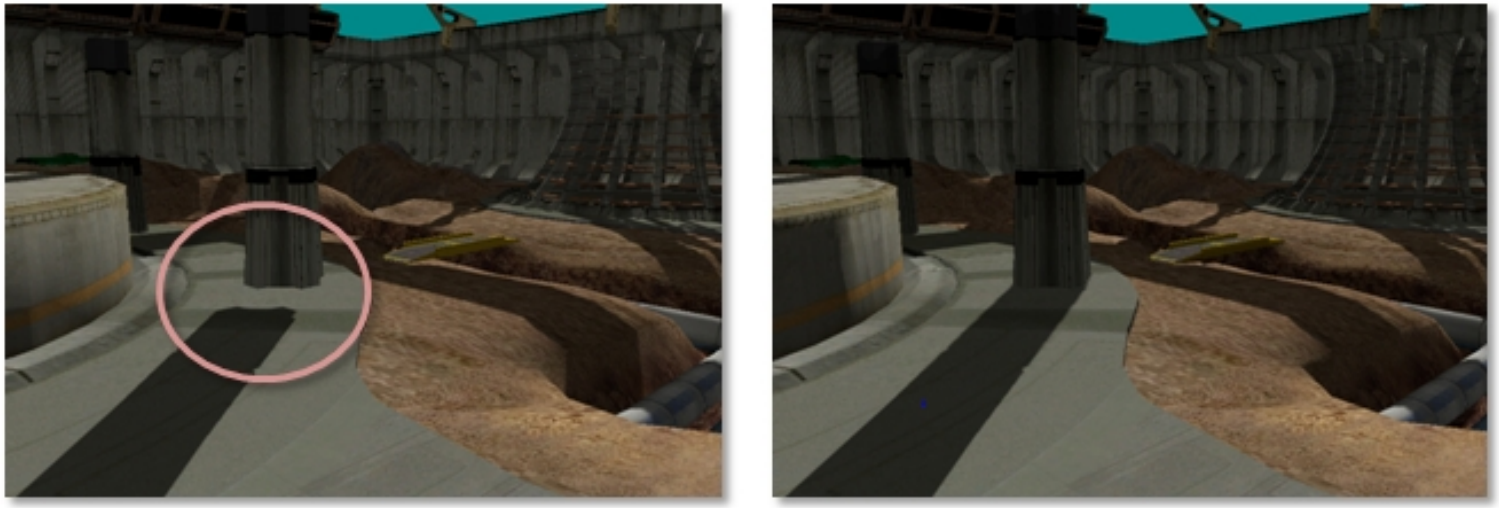
Shadow Mapping

- Problem: Shadow acne
  - False positives due to discrete grid of shadow map samples



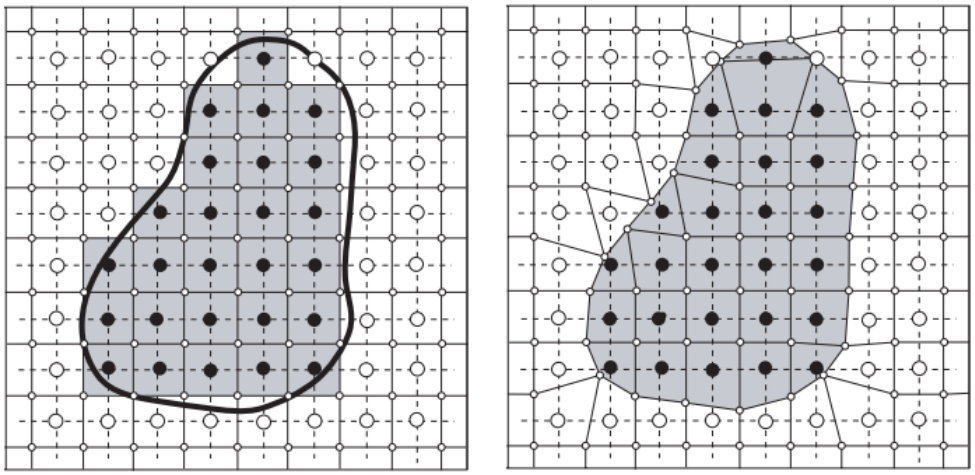
Shadow Mapping

- Fix: Add small bias value
  - Too large and shadows start "floating"
  - Better to scale with slope of polygon relative to light
  - Tight near and far planes help too



Silhouette Shadow Maps

- Gives non-jagged hard shadows
  - Idea: instead of filtering, warp shadow map lookups to fit the real edge
  - Store warp lookups in silhouette map
  - Sen, Cammarano, Hanrahan (Stanford, 2004)



Before warp

After warp

Silhouette Shadow Maps

- Algorithm
  - Render shadow map normally
  - Render silhouette map
    - Render thick edges as 3 pixel wide quads
    - Fragment shader computes (x, y) offset to edge
  - Render shadowed scene from camera
    - Apply shadow mapping normally
    - If on boundary (~1% of pixels), look up offset to edge from silhouette map
    - Use that to determine where to sample
    - See presentation for details
      - http://people.csail.mit.edu/ericchan/talks/sig2004-silmap.web/

Cascaded Shadow Maps

- Shadow mapping has problems
  - Resolution varies across scene
  - One shadow map per object doesn't scale
- Idea: fit several shadow maps to camera
  - Want uniform shadow map density in screen-space
  - Objects near eye require higher world-space density than objects far away
  - Use a cascade of 4 or 5 shadow maps
  - Each one is for a certain depth range of the scene
- Used in almost all modern games

Cascaded Shadow Maps

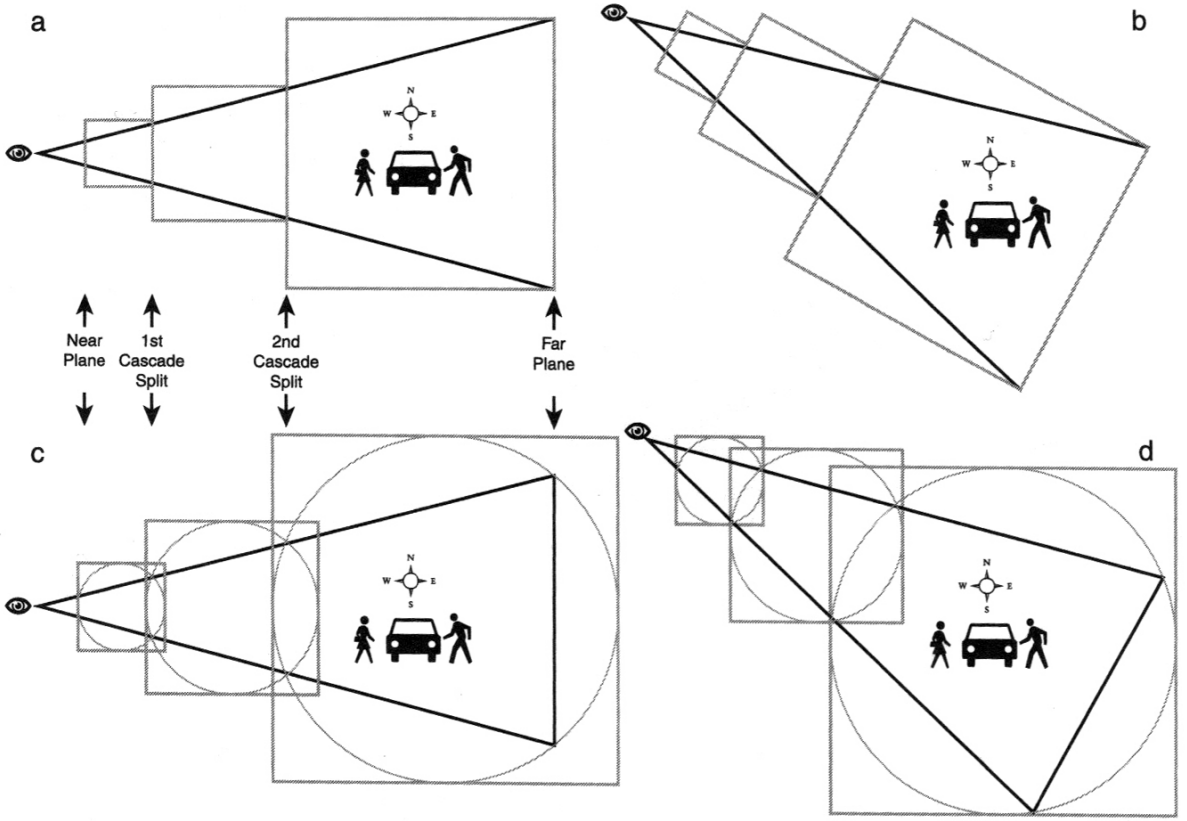
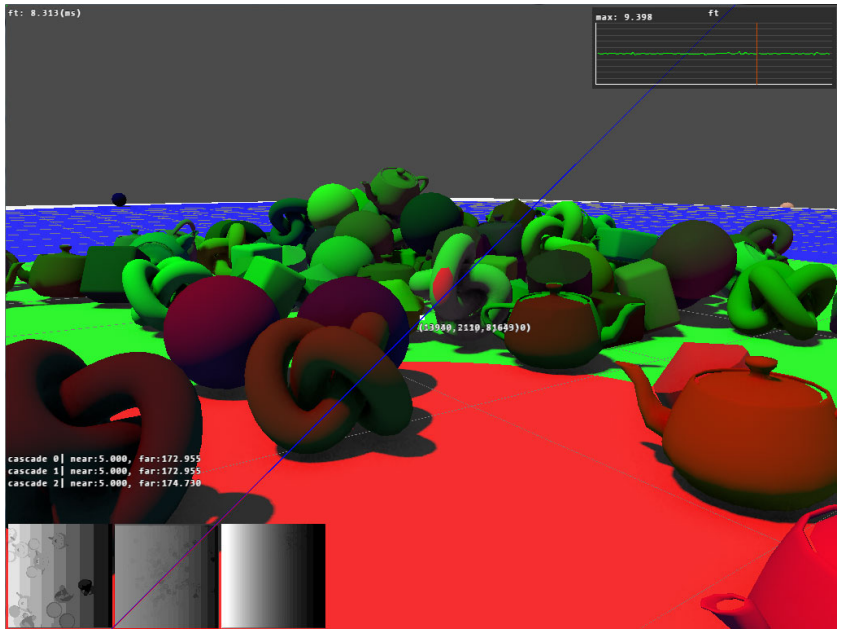


FIGURE 4.1.2 The view frustum in world space split into three cascade frustums and their corresponding shadow map coverage. We use a top view with the light direction pointing straight down the horizontal world plane.

Cascaded Shadow Maps

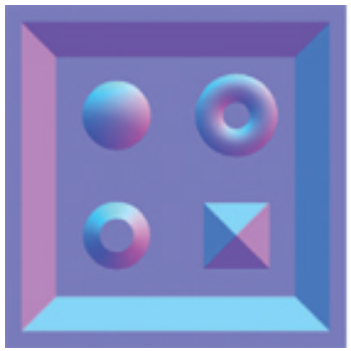
- Use depth in shader to choose cascade
  - Can blend between two closest cascades to smooth out discontinuities



Scene with a cascade of 3

Relief Mapping

- Make a 2D surface look 3D
  - Raytrace through surface along view ray in shader
  - Optionally also trace a shadow ray to light
  - Get detailed geometry with very few vertices
  - Automatic level of detail



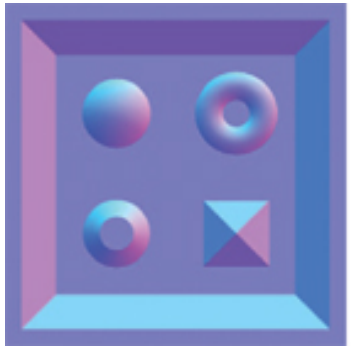
Normal map

Depth map

Flat surface lit by normal map

Relief Mapping

- Make a 2D surface look 3D
  - Raytrace through surface along view ray in shader
  - Optionally also trace a shadow ray to light
  - Get detailed geometry with very few vertices
  - Automatic level of detail



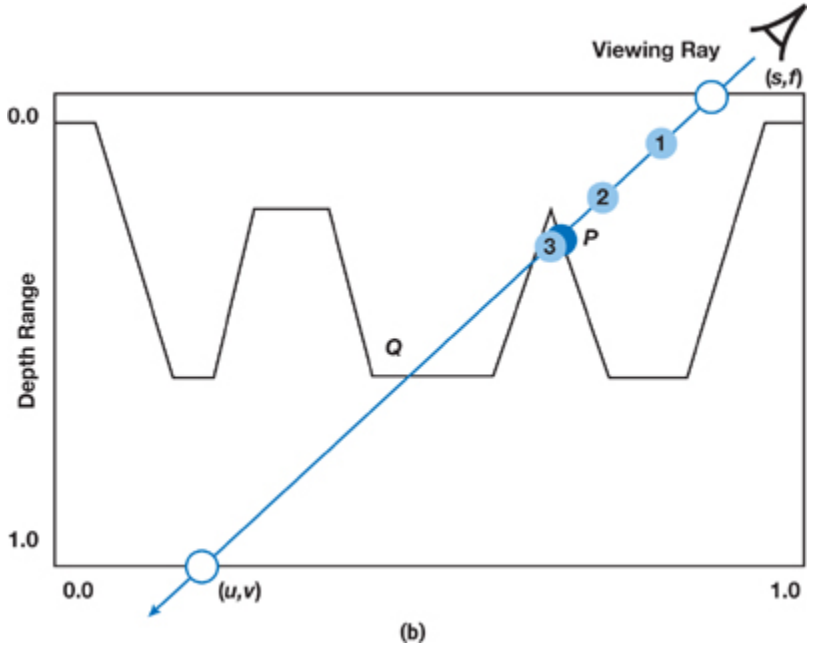
Normal map

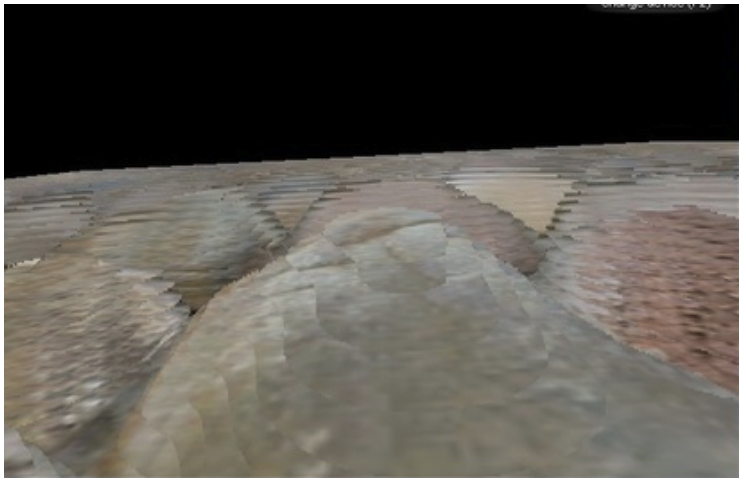
Depth map

Relief mapping with shadows

Relief Mapping

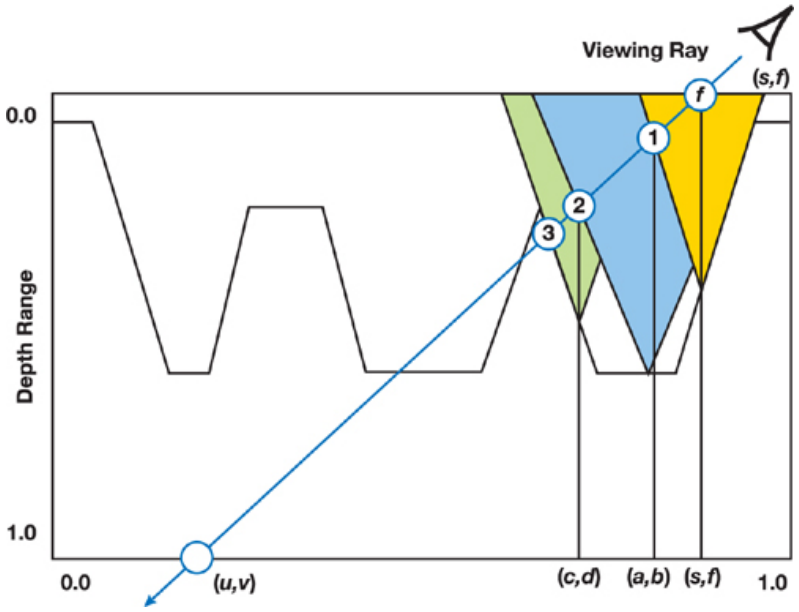
- Raytracing done using loop in shader
  - Take periodic samples and compare depth values, stop when inside surface (called raymarching)
  - Uses too many samples, also has stepping artifacts





Relief Mapping

- Different methods for faster raytracing
  - Linear search followed by binary search
  - Cone stepping
    - Each texel stores a cone containing no geometry
    - We know ray can safely move to edge of cone



Screen-Space Ambient Occlusion

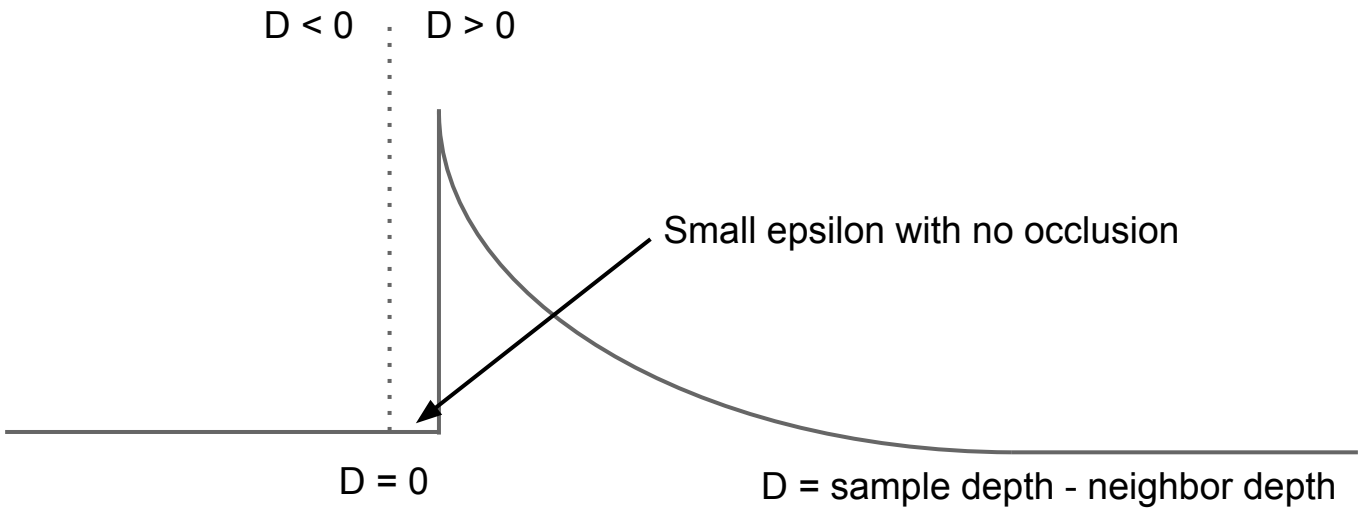
- Darken ambient term in occluded areas
  - Approximates indirect lighting

Screen-Space Ambient Occlusion

- Calculating occlusion
  - Probe nearby geometry using raycasting
  - Shoot rays in a hemisphere out of surface
  - Objects in close proximity cause darkening
- Idea: per-pixel occlusion approximation
  - Flatten raycasting to 2D in the image plane
  - Sample the depth of 8 to 32 neighboring pixels (requires deferred shading)
  - Don't count off-image samples as occlusions
  - Compare neighbor depth to 3D sample depth
  - If neighbor is in front, occlusion may be occurring

Screen-Space Ambient Occlusion

- Occlusion falloff function
  - Blockers closer to the sample should occlude more
  - Blockers far from the sample don’t occlude at all
  - Blockers behind don’t occlude at all



Real Time Local Reflections

- Used in Crysis 2

Real Time Local Reflections

- Used in Crysis 2

Real Time Local Reflections

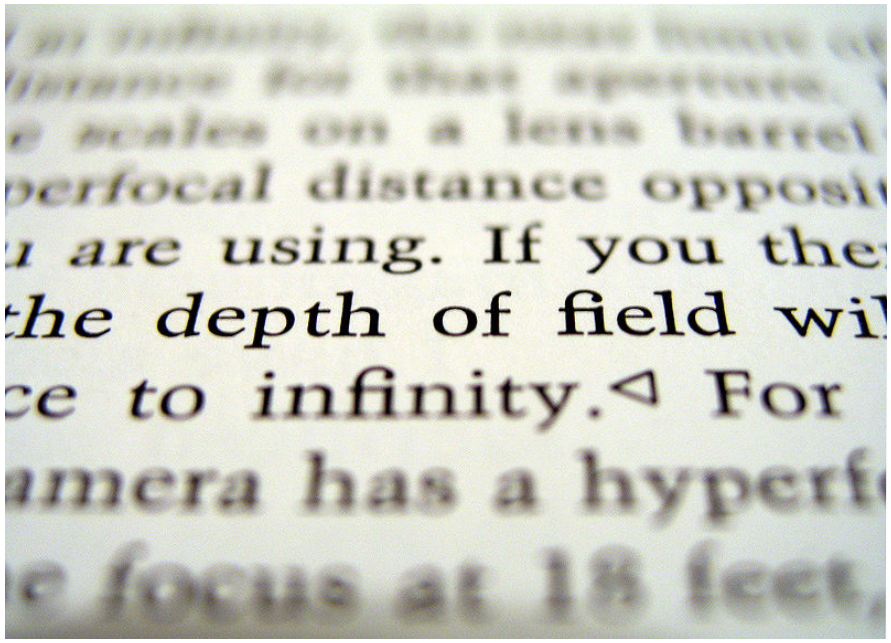
- Existing reflection methods (rasterization)
  - Render scene flipped about a plane
    - Use rendered scene as reflection
    - Only works for planar surfaces
  - Render scene from a point of view into cube map
    - Look up into cube map using reflection vector
    - Only works for small objects
- Reflections are expensive with rasterization
  - Need to render scene once per planar surface or per reflective object
  - Raytracing is much more straightforward

Real Time Local Reflections

- Raytrace reflections in screen space
  - Compute reflection vector per pixel using depth and normal from G-buffer
  - Raymarch along reflection vector
  - Project ray into 2D and check if scene depth is within threshold of ray depth
  - If so, use color from previous frame as reflection
- Edge cases (no data)
  - Fade out as reflection faces the camera
  - Fade out reflections off screen edge

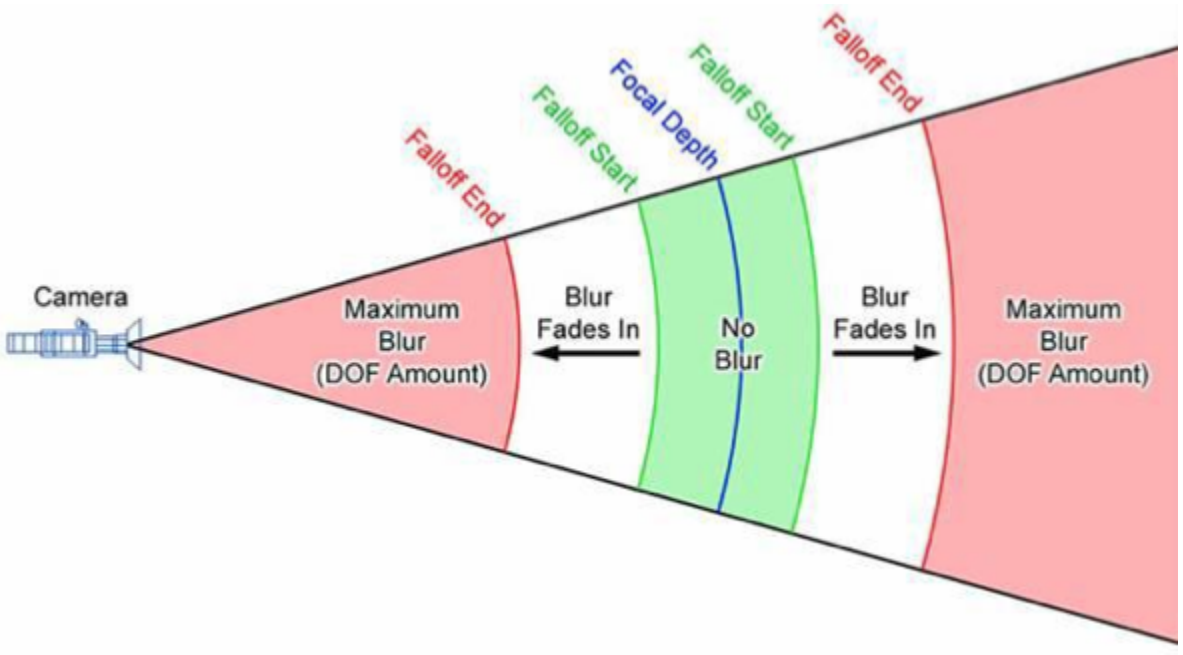
Depth of Field

- Out of focus blur in real cameras
  - Only one depth where objects are in focus
  - Focal blur increases in both directions away from that depth



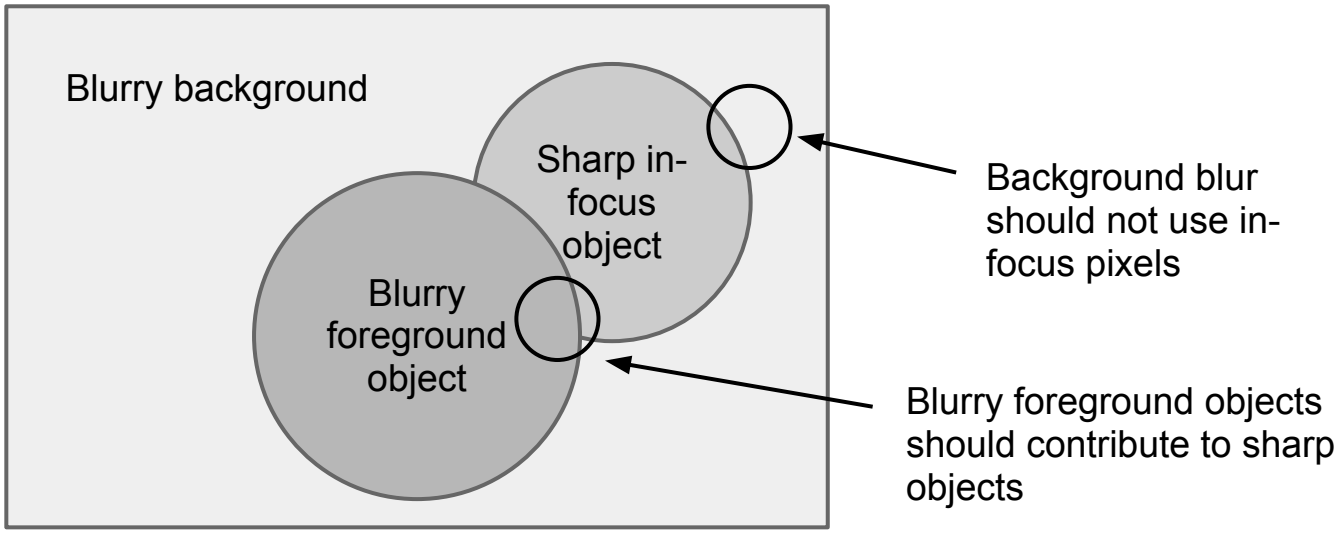
Depth of Field

- Model with post-process blur
  - Vary blur radius based on scene depth: slow
  - Interpolate between 3 images blurred with different radii: fast



Depth of Field

- Discontinuities are problematic (halos)
  - Don't use sharp objects in background blur
  - Blur over sharp objects for foreground objects



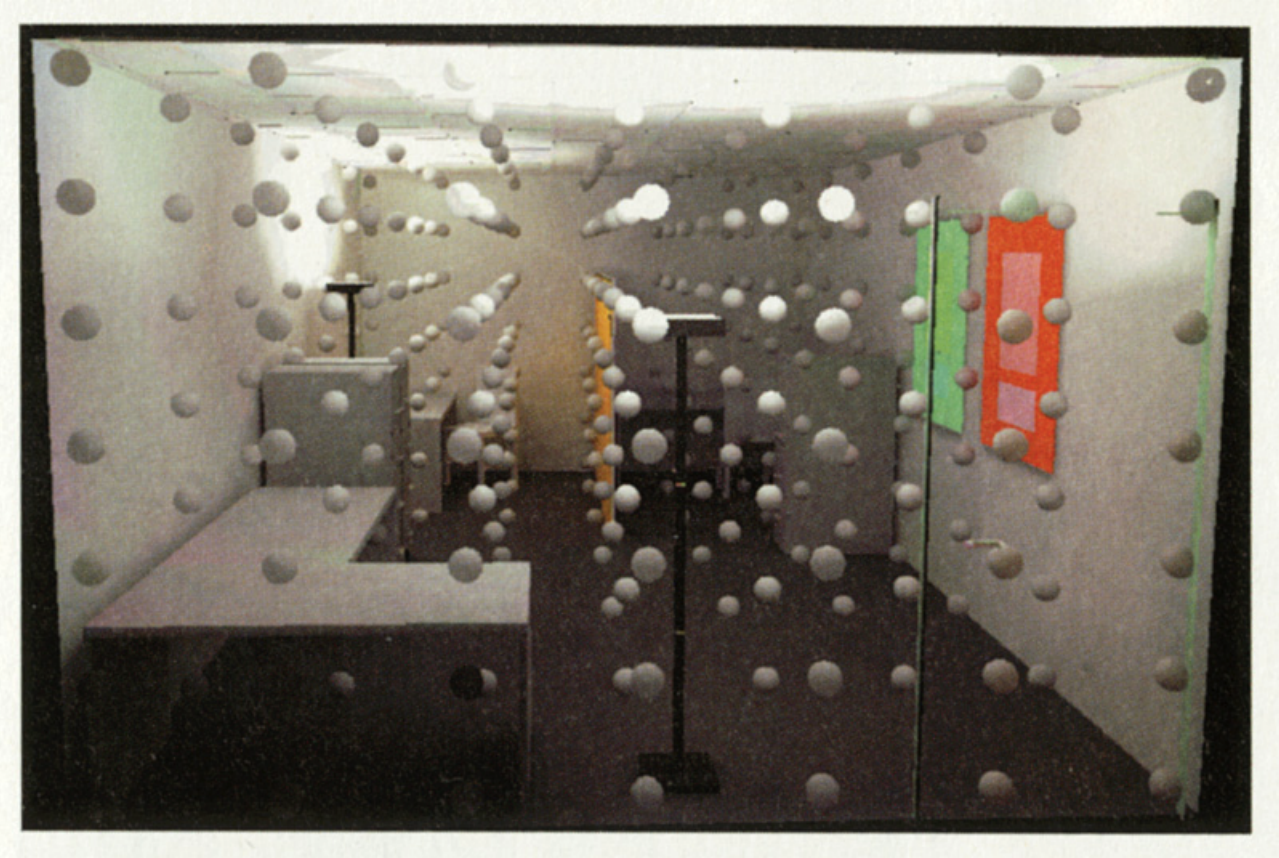
Depth of Field in Starcraft II

- Avoid sharp halos
  - Buffer of per-pixel blur radius
  - Weigh blur samples by radius buffer at sample point
  - Renormalize to sum to 1 again
- Halos around blurry objects
  - Compute blurred radius buffer
  - Compute blurred depth buffer (local average depth)
  - If average depth < current depth, use radius from blurred buffer, otherwise use radius from sharp buffer

DOF in Starcraft II cutscene

Real Time Global Illumination

- Precomputed with irradiance cache

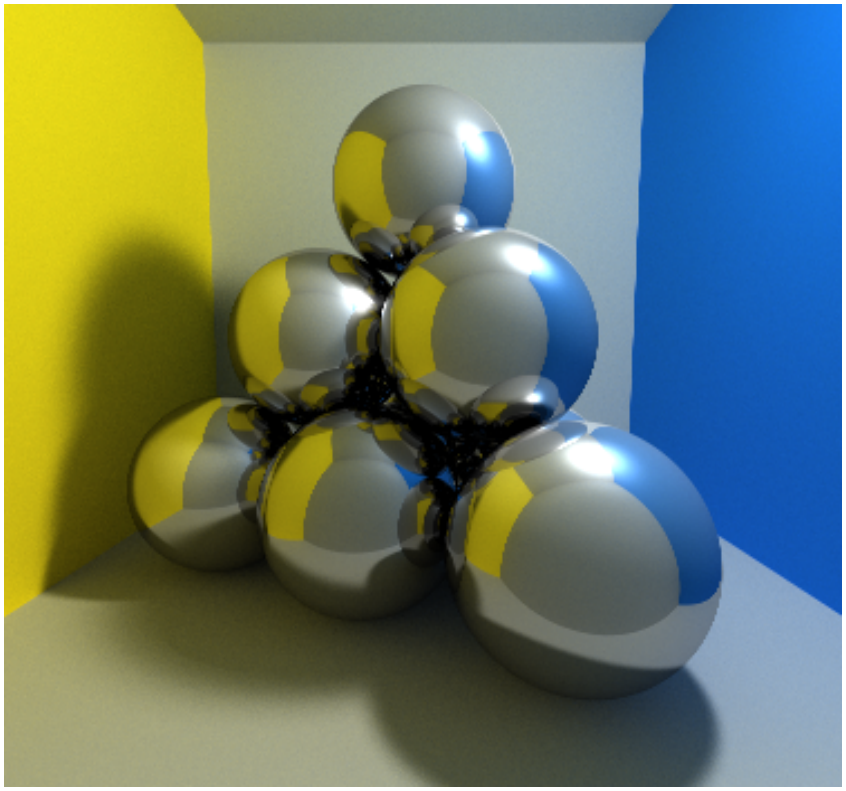


Real Time Global Illumination

- Precomputed with irradiance cache
  - Lightmaps for static objects
  - 3D grid of irradiance samples for dynamic objects
    - Each sample is snapshot of all light coming into a point
    - Think cube map, usually compressed using spherical harmonics
- Animated lightmaps
  - Static scene with moving light restricted to a path
- Precomputed Radiance Transfer (PRT)
  - Lightmap that can be queried by incident light angle
  - Lighting solution stored compressed using SH

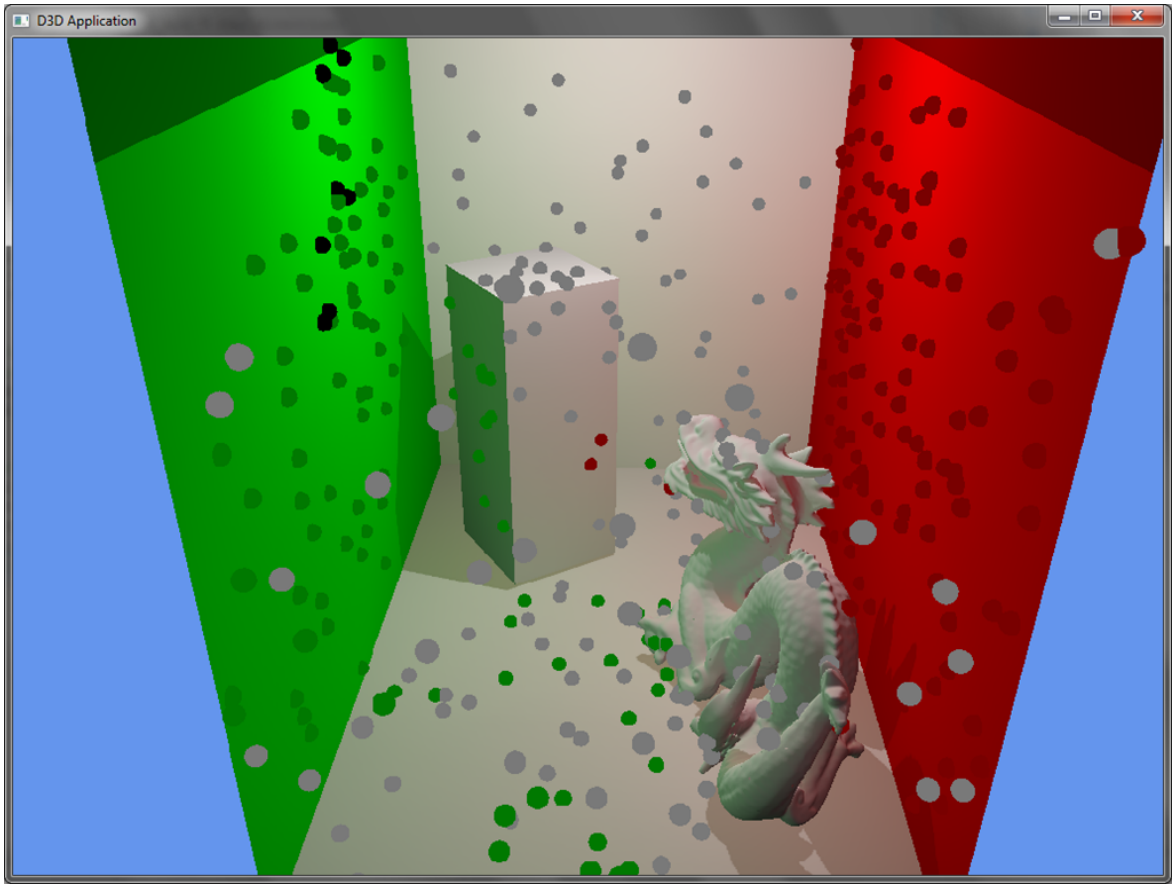
Real Time Global Illumination

- Path tracing directly
  - Not used much in games, technology still advancing



Real Time Global Illumination

- Instant radiosity



Real Time Global Illumination

- Instant radiosity
  - Shoot some photons into the scene (~200)
  - Only do one bounce
  - Virtual Point Light (VPL) where they land
- Rendering
  - Direct lighting: as usual (shadow maps)
  - Indirect lighting: each VPL becomes point light
- Updating
  - Randomized nature means lots of noise
  - Cache valid VPLs between frames

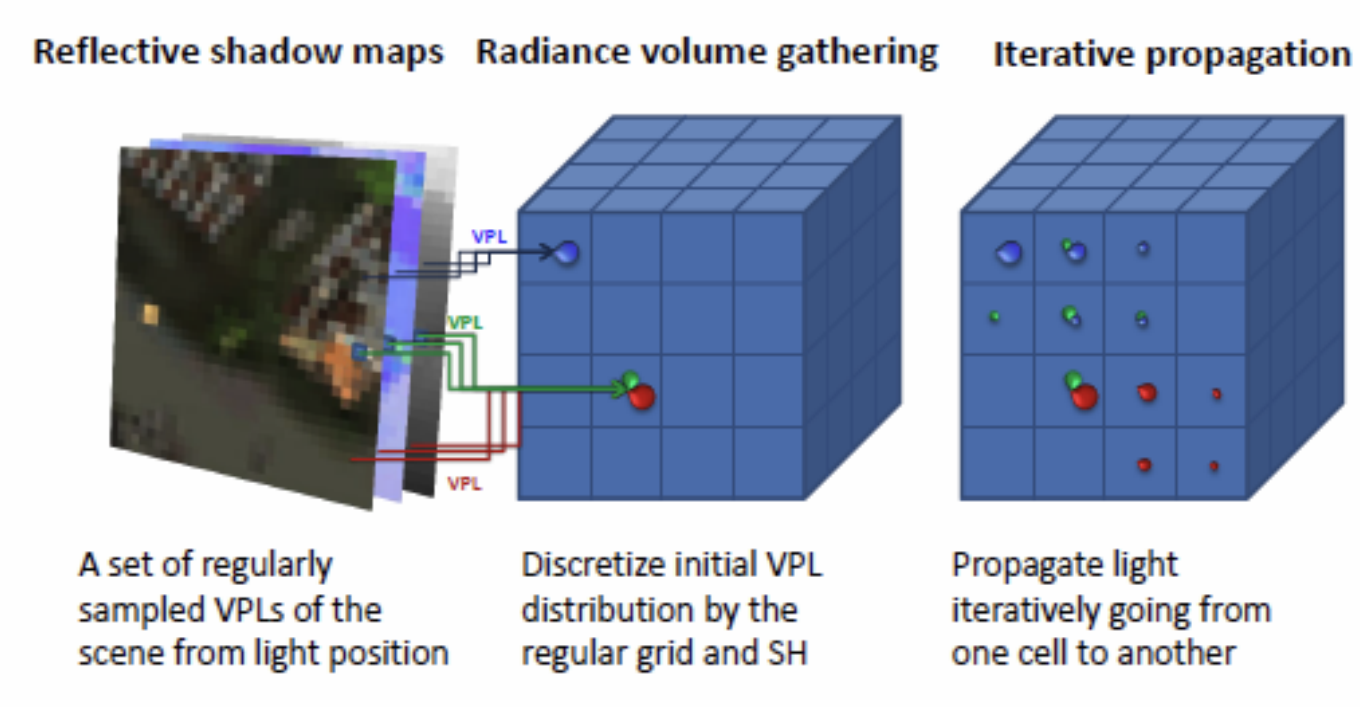
Real Time Global Illumination

- Light Propagation Volumes

http://www.youtube.com/watch?v=vPQ3BbuYVh8

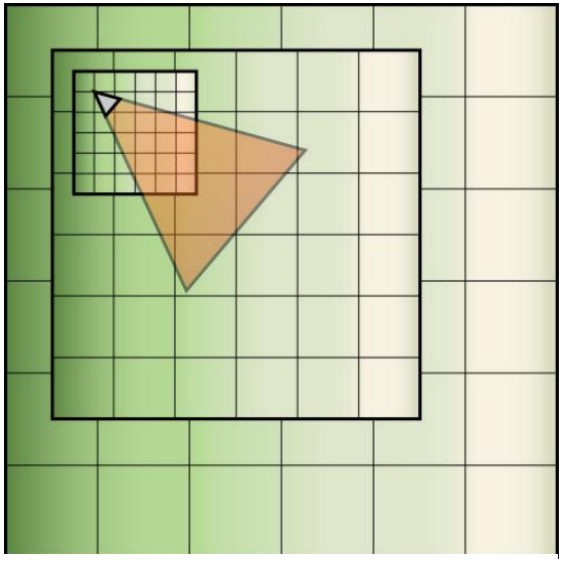
Real Time Global Illumination

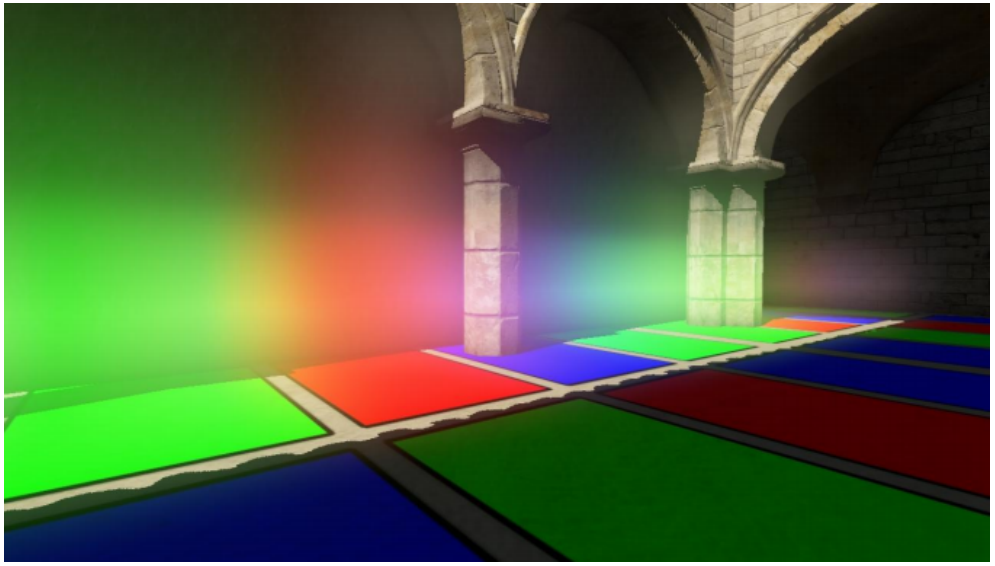
- Light Propagation Volumes



Real Time Global Illumination

- Light Propagation Volumes
  - Used in CryEngine (Crysis)
  - Scales better with more VPLs
  - Use cascade for details close to camera
  - Raytrace LPV for volumetric effects





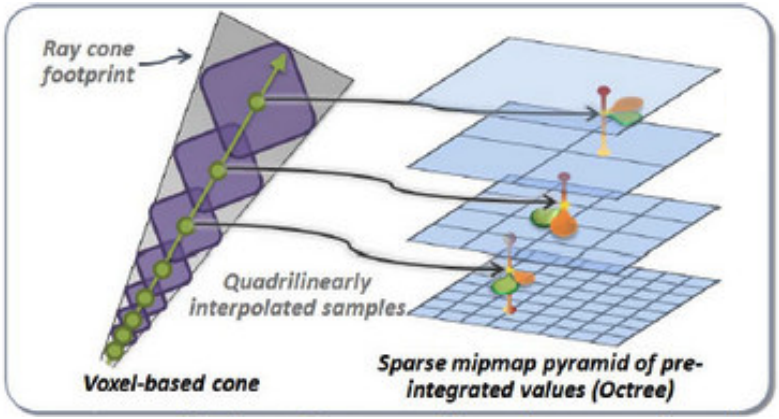
Real Time Global Illumination

- Voxel Cone Tracing

http://www.youtube.com/watch?v=fAsg_xNzhcQ

Real Time Global Illumination

- Voxel Cone Tracing
  - Store scene in octree
  - Rasterize new octnode data on the fly
  - Trace cones, not rays, using radius to pick tree level
- Makes hard things easy
  - Depth of field, soft shadows, glossy reflections
  - Blurry samples are cheaper than sharp samples!



C++ Tip of the Week

- Circumventing protections in C++
  - Need to access private members of existing library
  - Don't want to edit library source (allows for upgrades)
  - Technically illegal C++ (tokens can't be defined)

```
#define private public
#define protected public
#define class struct

#include "library.h"

#undef private
#undef protected
#undef class
```

Resources

http://developer.amd.com/media/gpu_assets/Isidoro-ShadowMapping.pdf
http://developer.amd.com/gpu_assets/S2008-Filion-McNaughton-StarCraftII.pdf
http://developer.amd.com/media/gpu_assets/Course_26_SIGGRAPH_2006.pdf
http://www.crytek.com/cryengine/presentations/secrets-of-cryengine-3-graphics-technology