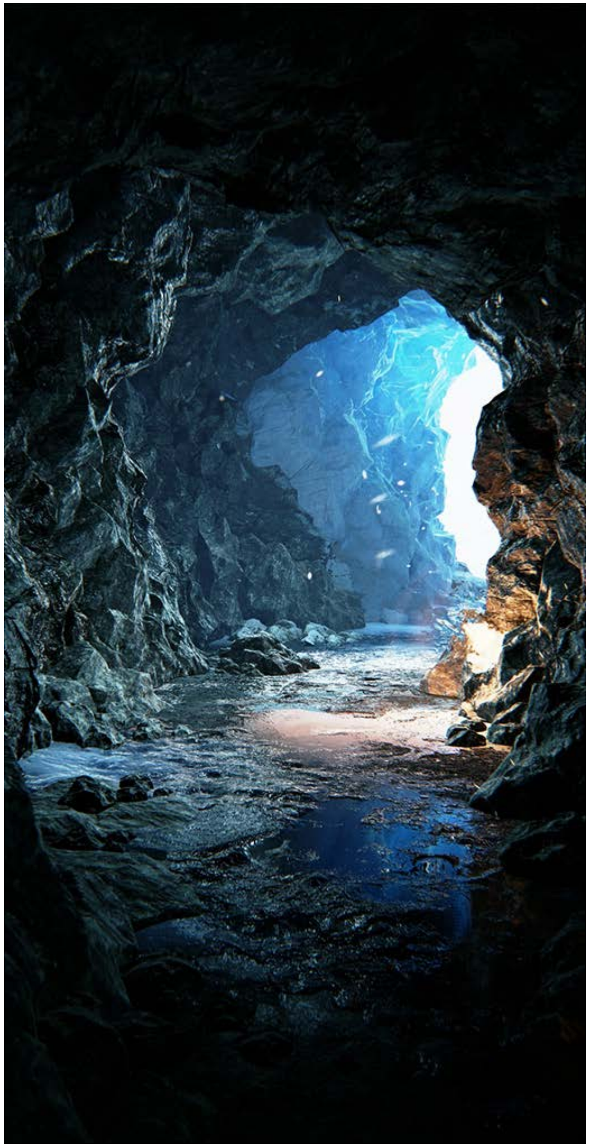
COMPUTER GRAPHICS COURSE

Game Graphics Techniques

PART II



Georgios Papaioannou - 2020

REAL-TIME DYNAMIC GLOBAL ILLUMINATION

Indirect Illumination

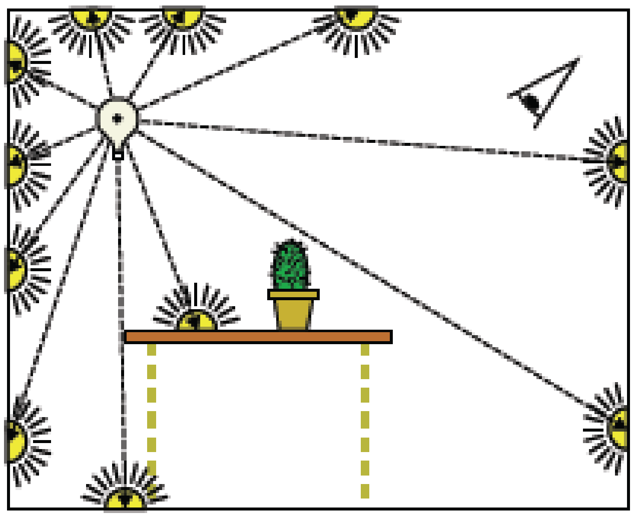
- Dynamic GI: Changes and adapts to follow:
  - the direct illumination in the scene
  - Optionally, changes to geometry and other dynamic aspects of the environment (particles, participating media, etc.)
- We typically treat the different BRDF response to incident illumination with different tools and methods in real time graphics:
  - Wide scattering – rough surfaces
  - Focused scattering – glossy and mirror-like surfaces

Instant Radiosity

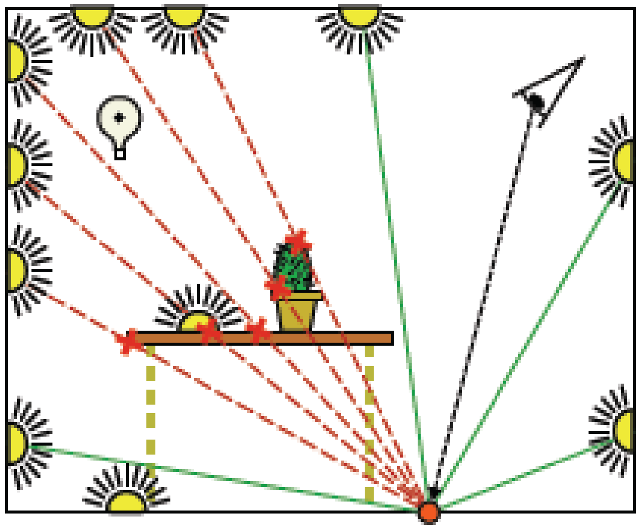
- Covers a wide range of methods, both interactive and off-line
- The concept is to replace indirect light bounces with direct illumination produced by virtual point lights (VPLs)
- VPLs (complete with visibility information) are placed at the intersection of photons from the light source with the geometry
- VPLs model the radiosity emitted from those intersection points
- VPLs are not limited to the first bounce only

Instant Radiosity

VPL placement

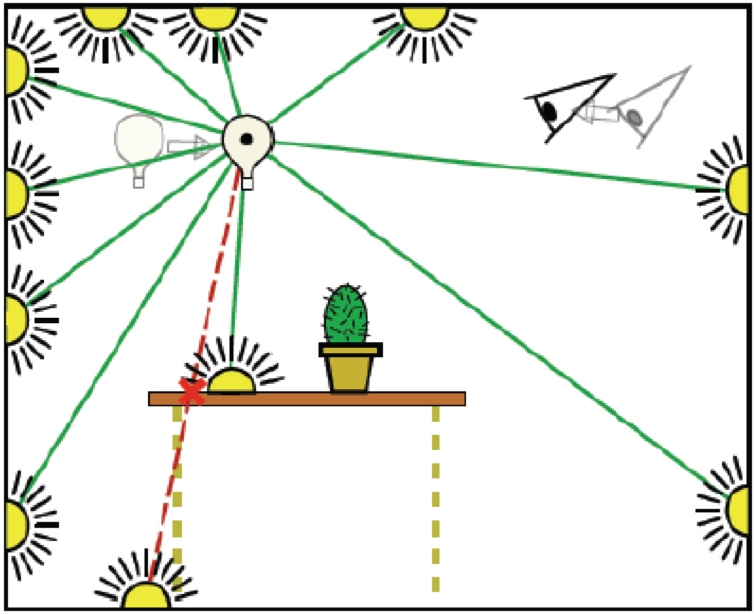


”Indirect” illumination from VPLs



Instant Radiosity – Dynamic VPL Update

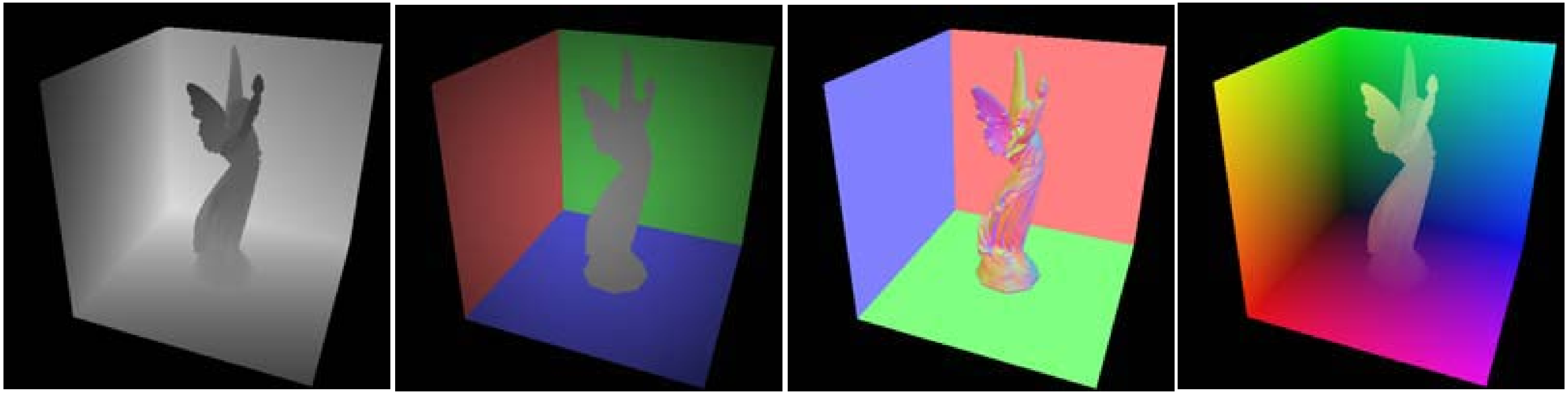
- Original CPU technique supported VPL updates
- When the scene changes, VPLs are updated:
  - Test VPL against shadow map
  - If invisible (beyond SM), discard VPL and add a new one



Reflective Shadow Maps

Reflective Shadow Maps

- Is a fast indirect lighting technique using:
- Shadow maps (depth maps) extended to also store VPL data:
  - Normals at visible points
  - Illumination (VPL power) at visible points
  - Optionally, location of VPLs and other data



Reflective Shadow Maps

- Essentially, an RSM replaces the tracing of VPLs in the scene:
- Each SM texel is considered a VPL
- The shadow map contains the nearest scene points to the light source
- The extra data completely describe the power distribution of each VPL (shadow map texel)
- The extended SM storage is used by other GI techniques → RSM now also refers to the multi-channel shadow map storage.

Reflective Shadow Maps

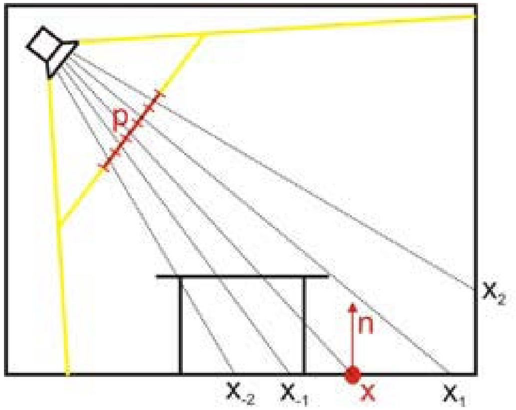
- What the RSM does NOT provide is visibility information for each VPL
- Therefore, the light from each VPL is considered unoccluded → no secondary bounce occlusion
- Also, RSM provides first-bounce (near field) GI only

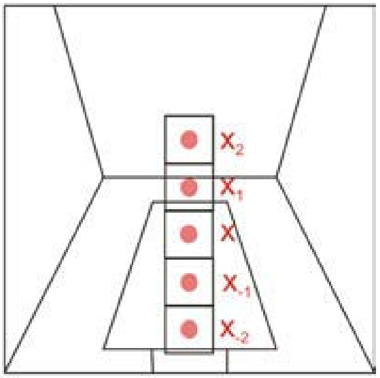
Using the RSM for Global Illumination

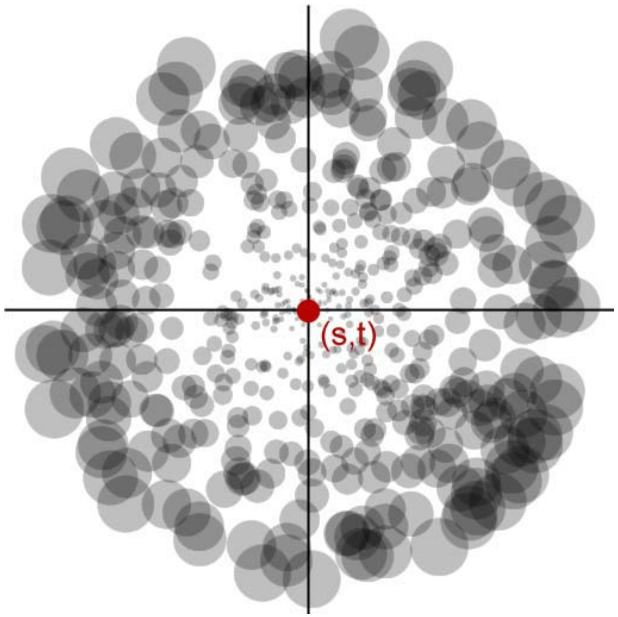
- RSM texels are sampled in the same manner as VPLs
- Light transfer can be estimated between each RSM virtual area light (or point light, depending on model) and the illuminated point
- Caution: Light transfer does not evaluate visibility between RSM samples and the receiving point

Using the RSM for Global Illumination

- Practical RSM sampling:
  - Project receiving point on RSM
  - Determine an area around projected point in RSM parametric space to sample
  - Accumulate RSM sample contribution





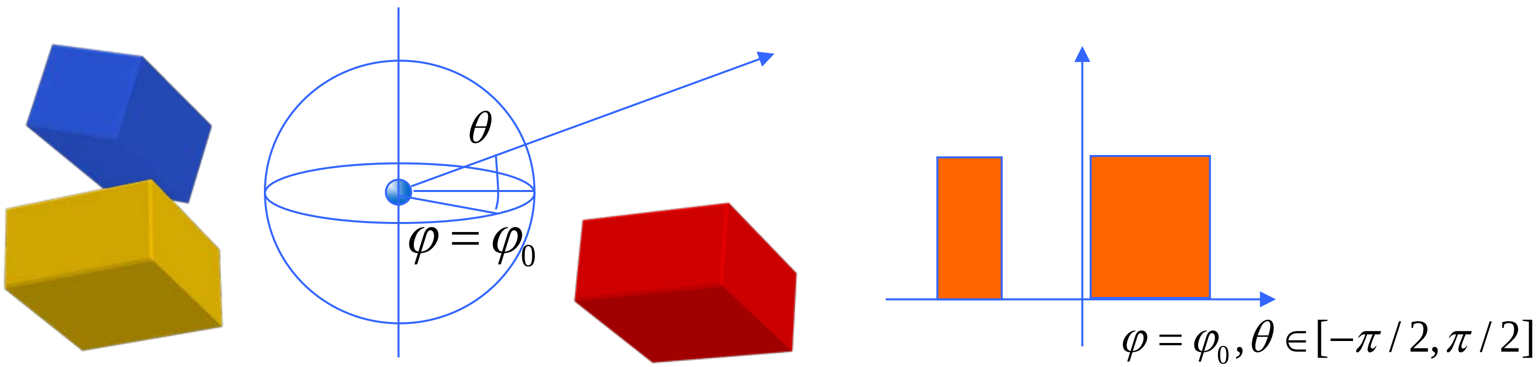


Precomputed Radiance Transfer

- It is the pre-calculation of the light transport operator on or near surfaces
- It is typically compressed and stored as a (hemi)spherical function (dependence on input or output, not both)
- During runtime, the PRT function is multiplied with a similarly coded illumination field to yield the resulting bounced energy

Frequency Analysis of Radiance Field

- Similar to radiance, we can encode visibility as a 5D field:
  - What is the visibility (how open is the environment) at a point (x,y,z) in space in a direction (θ,ϕ)?
  - Encodes the ability of the specific point to receive light from an incident direction (θ,ϕ)



- What are the spectral characteristics of these fields?

Frequency Analysis of Illumination (1)

- Global illumination effects have distinctively different spectral characteristics
- As a principle:
  - Diffuse inter-reflections produce low frequency directional radiance
  - The same holds for most cases involving occlusion in diffuse light bounces
  - Direct illumination with occlusion (shadows) contains high frequencies in general (discontinuities)
  - Specular transmission usually contains high frequencies

Frequency Analysis of Illumination (2)

Encoding the Radiance/Visibility Field (1)

- Why?
  - Direct illumination is cheap to calculate at every point on the geometry
  - Indirect illumination is not
- Solution:
  - Precalculate on surfaces/cache points OR
  - Calculate at sparse locations at run time
- What:
  - Visibility AND/OR
  - Radiance field of indirect lighting

Encoding the Radiance/Visibility Field (2)

For real-time graphics:

- Calculating and storing the radiance/visibility field once or per frame:
  - Disassociates its utilization from the geometry
  - Enables the easy evaluation of GI in real-time graphics (direct rendering techniques)

Orthonormal Basis Functions

- A basis function b_n is an element of a particular basis for a function space
- Every continuous function in the function space can be represented as a linear combination of basis functions:

$$f(x) = \sum_{n \in N} a_n b_n(x)$$

- Check similarity with vector spaces (the Fourier series is also a periodic function basis)
- An orthonormal basis additionally satisfies the property:

$$\int b_i b_j = \delta(i - j) \qquad \forall i, j \in \mathrm{N}$$

Signal Projection on Orthonormal Bases

- The projection of an arbitrary continuous function on a set of basis functions results in the definition of the blending coefficients a_n
- It can be proven that for orthonormal function bases, the best least squares fitting of a function f over a predefined set of basis functions b_n results in:

$$a_n = \int f(x) b_n(x) \mathrm{d}x$$

- (Again, relate this with the dot product projection in orthonormal bases for vector spaces)

Signal Reconstruction

- The number of basis (blending) functions may be infinite or too large and therefore we must choose a finite subset of them that converges "reasonably" to the desired result

- The reconstructed function (signal) is derived from the linear combination of the (truncated series) of basis functions:

$$\tilde{f}(x) = \sum_{n=1}^{N} a_n b_n(x)$$

Spherical Harmonics (1)

- Spherical Harmonics define an orthonormal basis over the sphere **S**.
- A point s on the sphere is parameterized as:

$$s = (x, y, z) = (\sin\theta\cos\varphi, \sin\theta\cos\varphi, \cos\theta)$$

- They are harmonic functions and more specifically they constitute the angular part of the solution of the Laplace's equation on the unit sphere:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = 0$$

Spherical Harmonics (2)

- The (complex) basis functions are defined as:

$$Y_l^m(\theta,\varphi) = K_l^m e^{im\varphi} P_l^{|m|}(\cos\theta), l \in \mathbf{N}, -l \leq m \leq l$$

where P_l^m are the associated Legendre polynomials and K_l^m are the following normalization factors:

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}}$$

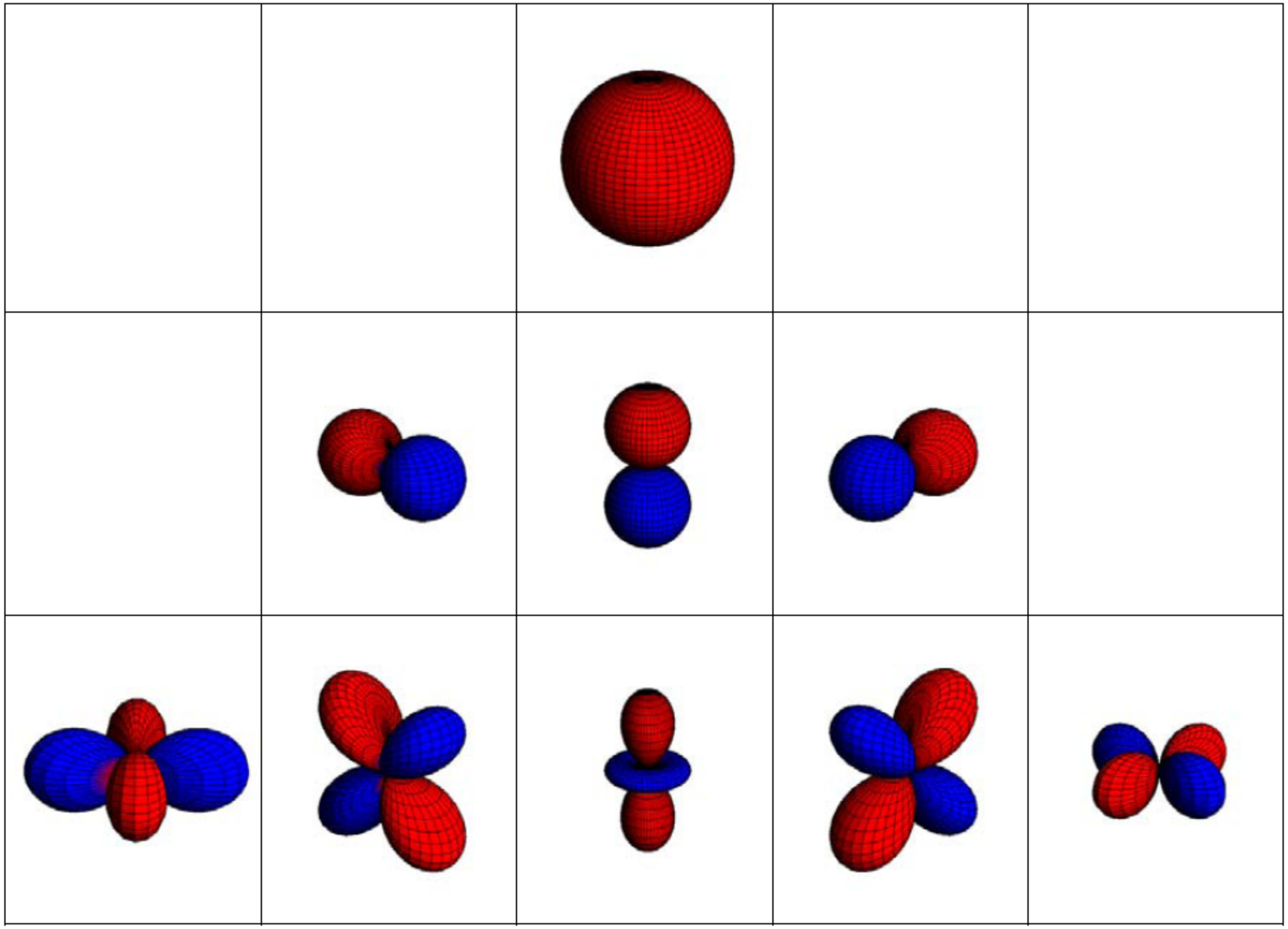
Spherical Harmonics (3)

- Real versions of the SH basis functions can be obtained from the transformation:

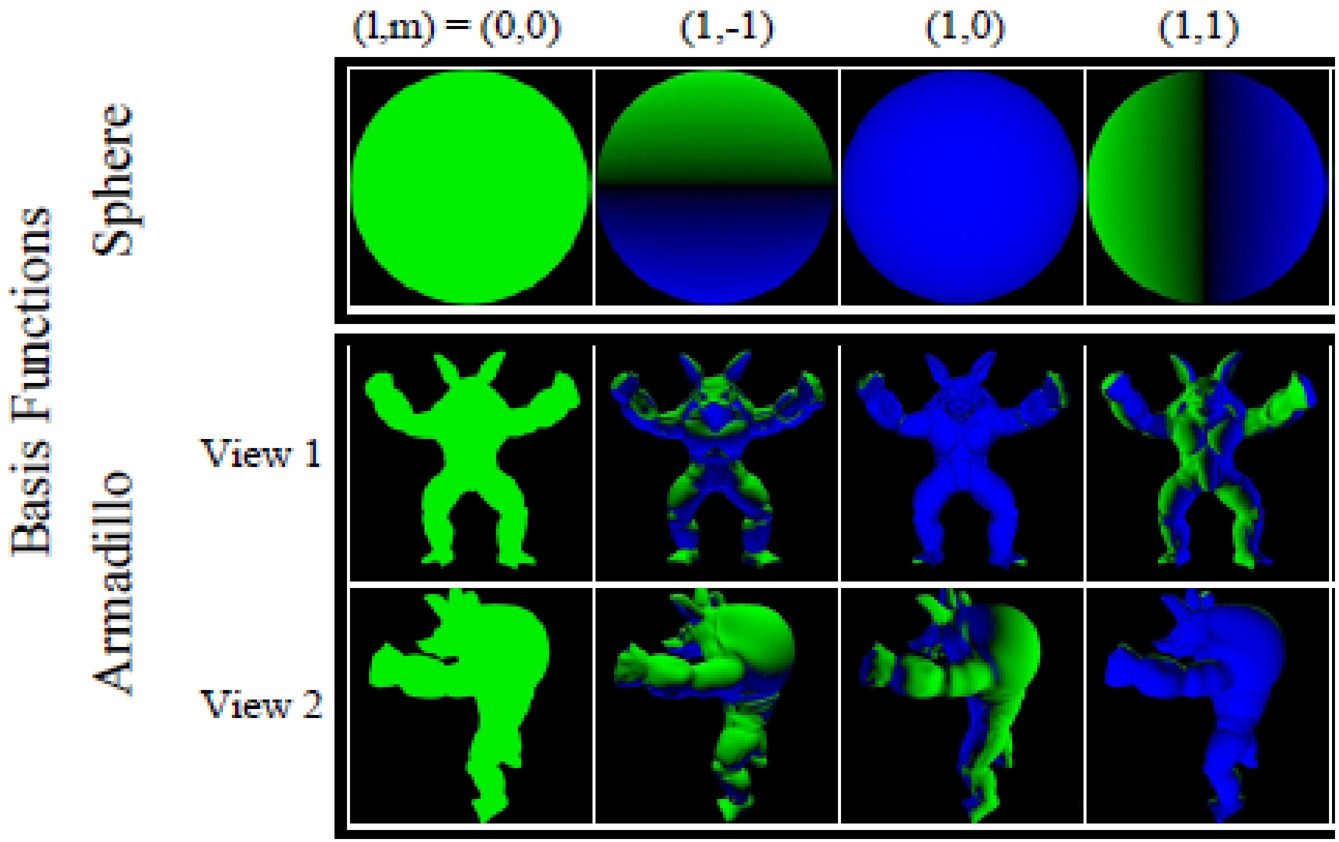
$$y_l^m = \begin{cases} \sqrt{2}\mathrm{Re}(Y_l^m) & m > 0 \\ \sqrt{2}\mathrm{Im}(Y_l^m) & m < 0 \\ Y_l^0 & m = 0 \end{cases} = \begin{cases} \sqrt{2}K_l^m \cos m\varphi \; P_l^m(\cos\theta) & m > 0 \\ \sqrt{2}K_l^m \sin|m|\varphi \; P_l^{|m|}(\cos\theta) & m < 0 \\ K_l^0 P_l^0(\cos\theta) & m = 0 \end{cases}$$

- l represents the band of the SH functions
- Each band has $2l+1$ SH basis functions
- Each band corresponds to an increasing angular frequency

Spherical Harmonics (4)



Spherical Harmonics (5)



Spherical Harmonics (6)

- Being an orthonormal set of basis functions:

$$f_l^m = \int f(s) y_l^m(s)\, ds$$

- The reconstruction of the signal can use up to any order of SH bands, truncating the infinite series of coefficients and respective basis functions
- Similarly, the encoded (projected) signal has to be band limited and encoded in a finite set of SH coefficients
- How many bands should we use?

Encoding Visibility (Distant Illumination) (1)

- From the rendering equation:

$$L_r(\phi_r, \theta_r) = L_e(\phi_r, \theta_r) + \int_{\Omega_i} L_i(\phi_i, \theta_i) f_r(\phi_r, \theta_r, \phi_i, \theta_i) \cos(\theta_i) d\omega_i$$

- If we assume only a “distant” environment emitting the radiance (e.g. sky, sun, distant light sources etc), then:

$$L_r(\phi_r, \theta_r) = \int_{\Omega_i} \underbrace{L(\phi_i, \theta_i)}_{\text{radiance}} \underbrace{V(\phi_i, \theta_i) f_r(\phi_r, \theta_r, \phi_i, \theta_i) \cos\theta_i}_{\text{transfer function}} d\omega_i$$

radiance transfer function

Encoding Visibility (Distant Illumination) (2)

- For diffuse surfaces this is simplified to:

$$L_r(\phi_r, \theta_r) = \frac{\rho}{\pi} \int_{\Omega_i} L(\phi_i, \theta_i) \overbrace{V(\phi_i, \theta_i) \cos\theta_i}^{T(\phi_i, \theta_i)} d\omega_i$$

- The hemisphere is aligned with the surface normal at every point
- The transfer function characterizes the specific point but for diffuse inter-reflection can be considered a slowly varying quantity (thus sparsely evaluated).

Encoding Visibility (Distant Illumination) (3)

- We can encode both the transfer function and the incident radiance using a set of basis functions
- Orthonormal bases (such as SH) are ideal as they provide the useful property:

$$\int \tilde{f}(s)\tilde{g}(s)ds = \sum_{i=1}^{k} f_k g_k$$

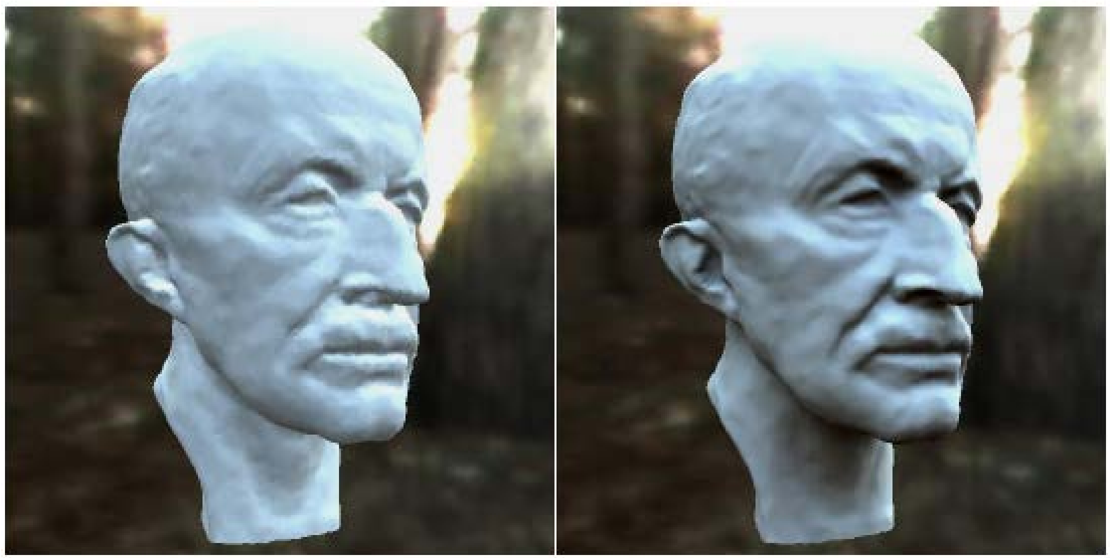
- i.e.: The integral of two band limited functions equals the dot product of their coefficients when projected to the orthonormal basis

Precomputed Radiance Transfer (1)

- The transfer (visibility over the hemisphere) function T can be precomputed and encoded in compact form
- When using Spherical Harmonics, 9 or 16 coefficients can effectively encode both T and L_i for diffuse light transfer
- The coefficients for T can be sparsely (pre-) evaluated, stored to and evaluated from:
  - A sparse lattice
  - A texture atlas

Precomputed Radiance Transfer (2)

$$L_r(\phi_r,\theta_r)=\frac{\rho}{\pi}\int_{\Omega_i} L(\phi_i,\theta_i)\cos\theta_i d\omega_i$$



$$L_r(\phi_r,\theta_r)=\frac{\rho}{\pi}\int_{\Omega_i} L(\phi_i,\theta_i)V(\phi_i,\theta_i)\cos\theta_i d\omega_i$$

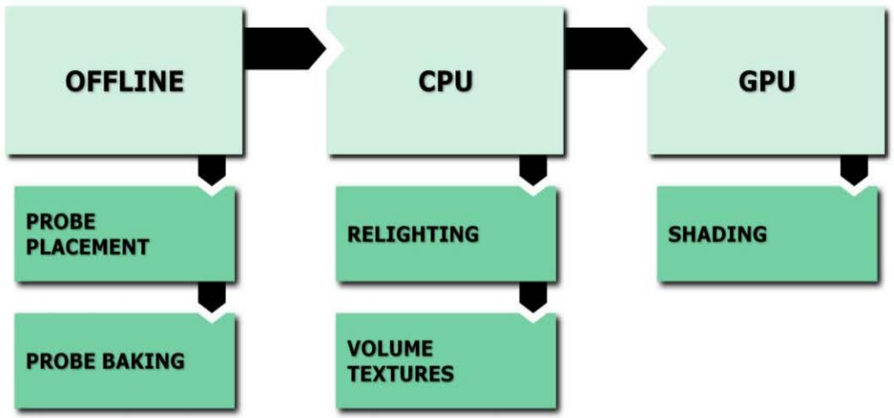
PRT in Games

- PRT can be computed and stored in lightmap format
  - Each texel has all the coefficients for a hemispherical PRT basis OR
- PRT can be volumetric
  - Expresses the visibility or outgoing energy ratio around a point in space
  - This spherical “probe” represents the PRT in the volume near it

PRT Case Study: Far Cry 3

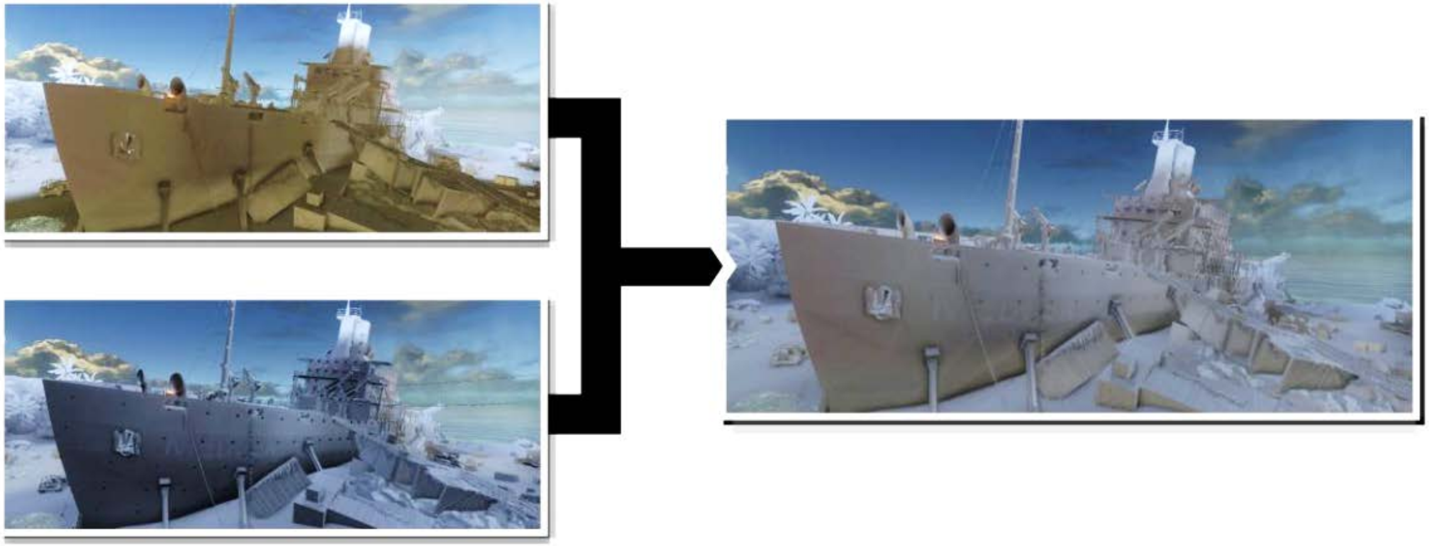
- Uses spherical probes arranged in space
- Precomputed visibility for sky lighting
- PRT (outgoing) for direct light bounce

- Deferred updates



PRT Case Study: Far Cry 3

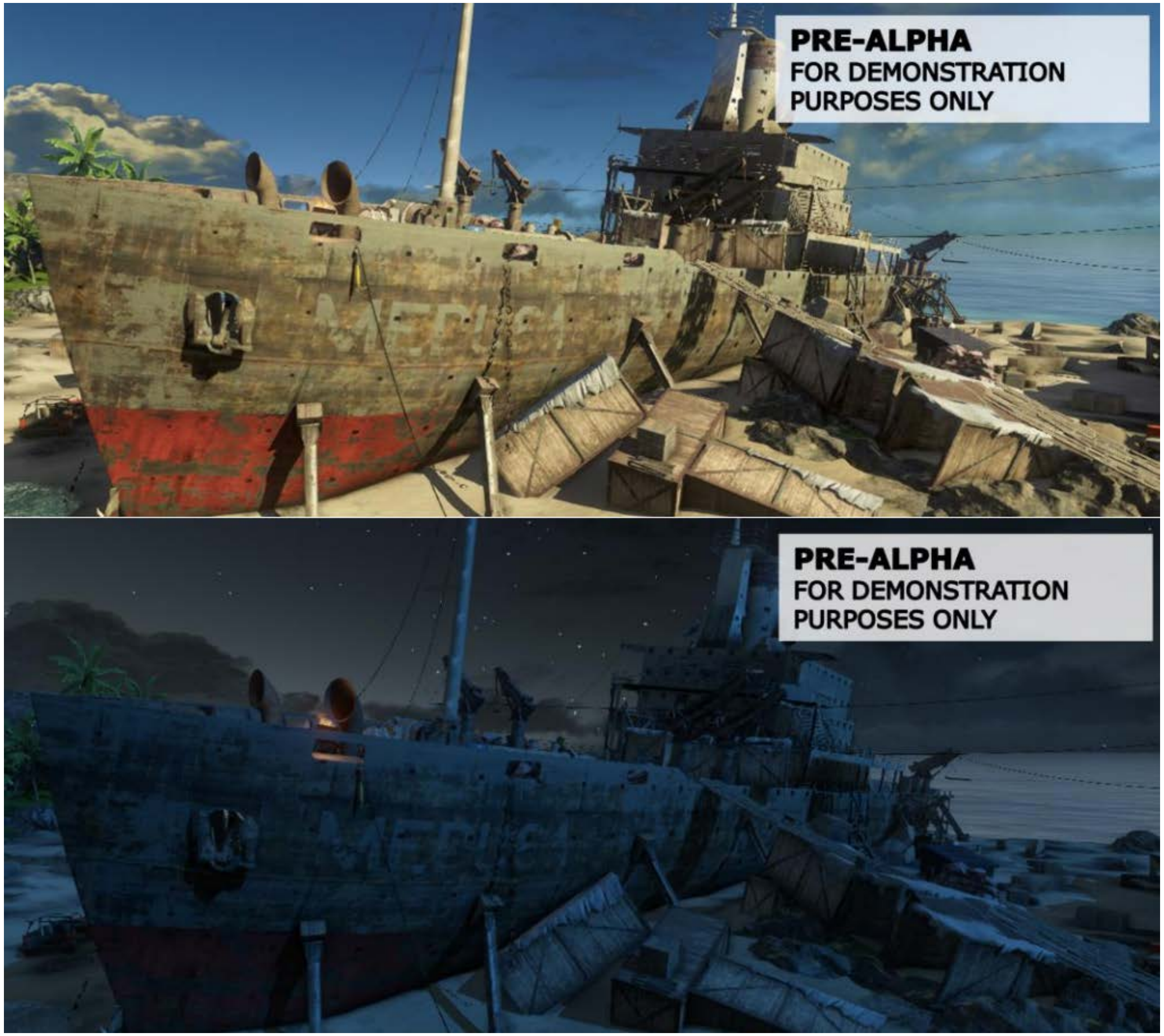
Probes: Reflected radiosity from sun on diffuse surfaces encoded in SH
Reconstructed on hemisphere over each point



Probes: Skylight visibility,
post-multiplied with skylight radiance field (also encoded in SH)

PRT Case Study: Far Cry 3

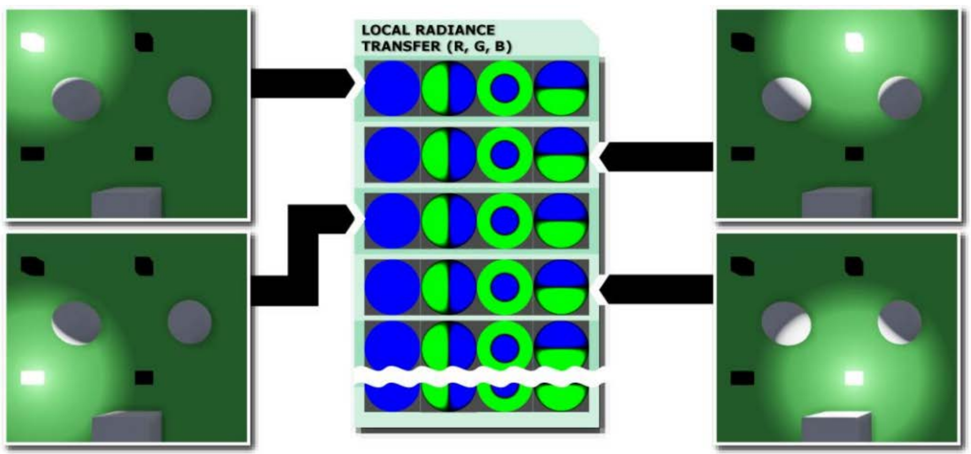
Indirect lighting from sources is dynamically updated to match conditions (see next)



PRT Case Study: Far Cry 3

- Probes are semi-automatically distributed in the environment at sparse locations
- A volumetric grid is overlaid on the environment
  - Each cell indexes the closest probe
  - At run time, shaded points falling within each cell, access the mapped probe for indirect lighting

PRT Case Study: Far Cry 3



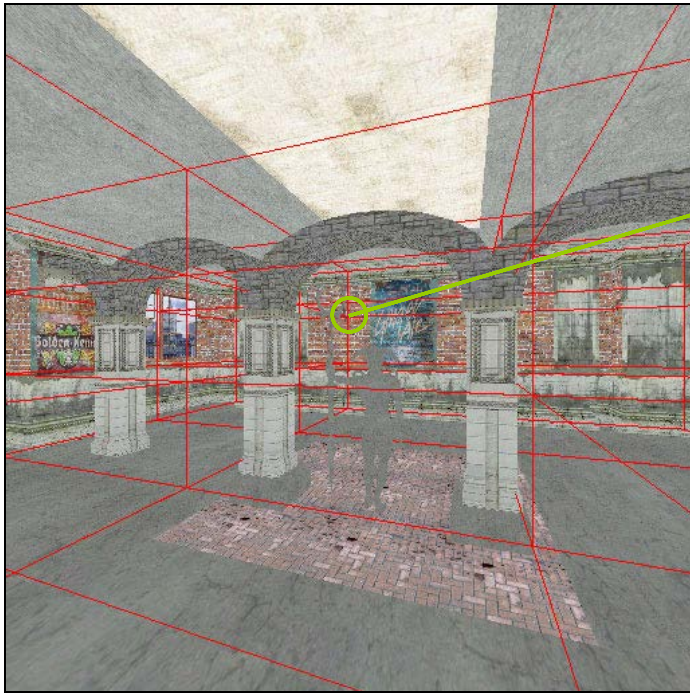
- For light bounce, estimate the average directional output radiance “as if” a unit source was placed directly on the probe
- At run time, for each light source distribute its energy to nearest probes and compute the bounce energy.
- Compute irradiance integral on surfaces using post-multiplied SH coefs (PRT * surface oriented hemisphere)

Radiance Field Caching

Radiance Field Caching

- Estimates the incident radiance field at the vertices of a uniform grid
- Radiance is captured by rendering the scene on a cubical environment map
- Compresses the radiance field using SH
- Evaluates the reflected radiance on surfaces by direct integration of the radiance field with the BRDF at each point in SH space
- SHs for points in between lattice vertices are interpolated

Radiance Field Caching



- For each node, the SH coefs are the superposition of the individual cubemap texel radiance projection:

$$L_l^m \approx \sum_{face=1}^{6} \sum_{i=1}^{size} \sum_{j=l}^{size} L_{face}(i,j) Y_l^m(\omega) A(\omega)$$

$$A(\omega) = \int_{pixel_{ij}} d\omega$$

Radiance Field Caching

- For Lambertian surfaces (diffuse reflection):

$$L_{indirect}(\mathbf{p}) = \frac{\rho(\mathbf{p})}{\pi} \sum_{l} \sum_{m=-l}^{l} L_l^m(\mathbf{p}) H_l^m(\mathbf{n})$$

Radiance field SH coefs interpolated from 8 nearest lattice points

Normal-aligned projected cosine-weighted hemisphere on SH basis

- Diffuse GI is well approximated with 2-3 order SH
- The transfer function can be generalized to Phong-like models (symmetric lobes) but require a significantly larger SH order (6+)→ impractical storage

Radiance Field Caching

- Practical issues:
  - For truly dynamic scenes, cubemaps must be completely re-evaluated often
  - Secondary bounces may be handled by exchanging light among lattice points
  - The sparseness of the grid necessitates additional occlusion criteria when evaluating the radiance field:
    - Depth maps are also acquired per node
    - Instead of simply trilinearly interpolating the node radiance, a visibility check is performed against the node’s range in the direction of the sample

Volume-based Global Illumination

Volume-based GI (1)

- Uses an intermediate regular approximation of the geometry (voxel grid) to store lighting and geometry data →
- Rough discretization of the shaded environment
- Why volume-based GI?
  - Decouples local pixel calculations (GPU pipeline) from full-scene data
  - Provides access to full-scene data in the local-only context of a shaded pixel
  - GI calculations independent of scene complexity

Volume-based GI (2)

- The "lit" voxels represent virtual point lights
- Occupied voxels effectively block light transport
- What do we need to store for one-bounce GI (per voxel):
  - Direct lighting (VPLs) directionally encoded using the normal at the shaded fragments
  - Voxel coverage as occupancy (same storage – black voxels)
- What do we need for extra bounces?
  - Averaged (per voxel) surface normals
  - Average (per voxel) albedo

Volume-based GI (3)

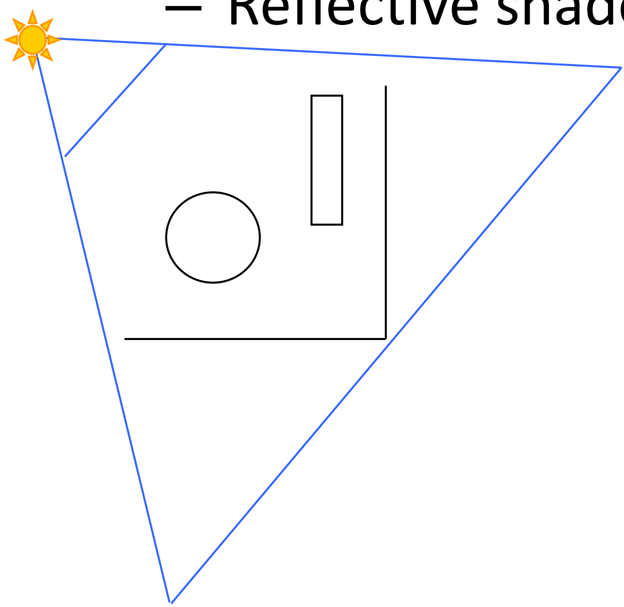
- All methods have two phases:
  - Volume data generation
  - GI estimation
- Volume generation:
  - Point injection
    - Geometry-based
    - Image-based
  - Multi-channel full-scene voxelization
- GI estimation:
  - Iterative radiance diffusion (light propagation volumes)
  - Ray marching

VBGI – Image-based Point Injection (1)

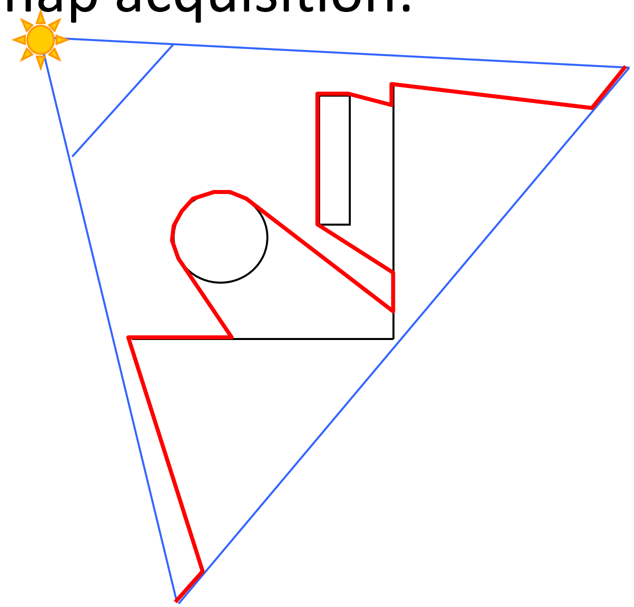
- Samples from the available frame buffers are injected into the volume using the technique discussed in part A
- Shadow maps (RSMs) hold a sampling of the surfaces lit by the particular light source → VPLs
- The camera buffer (MRT G-buffer) contributes additional occupancy-only points

VBGI – Image-based Point Injection (2)

- How are the points injected?
  - Reflective shadow map acquisition:



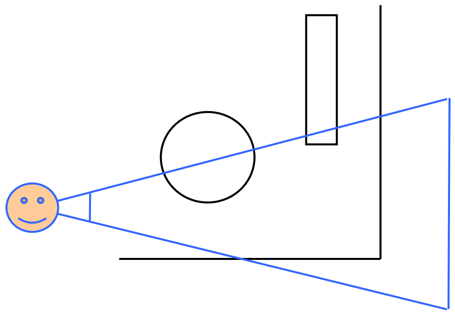
Light setup



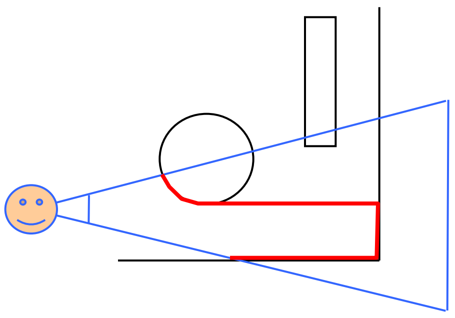
Shadow map points (WCS)

VBGI – Image-based Point Injection (3)

- How are the points injected (cont)?
  - Camera g-buffer acquisition (deferred rendering):



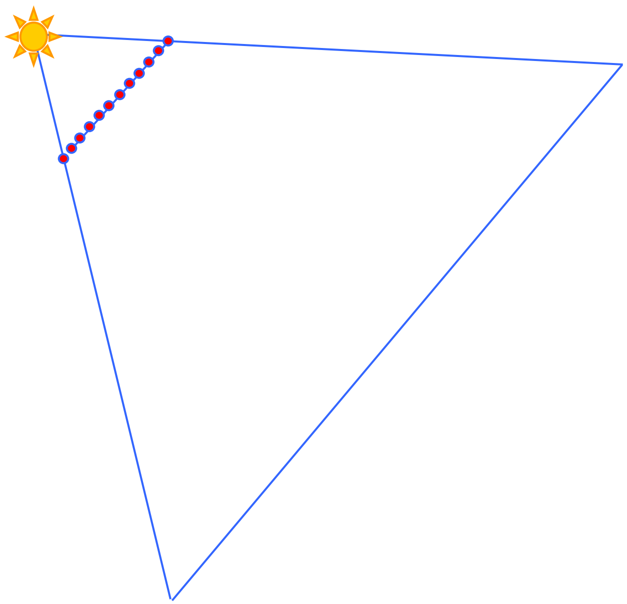
Camera setup

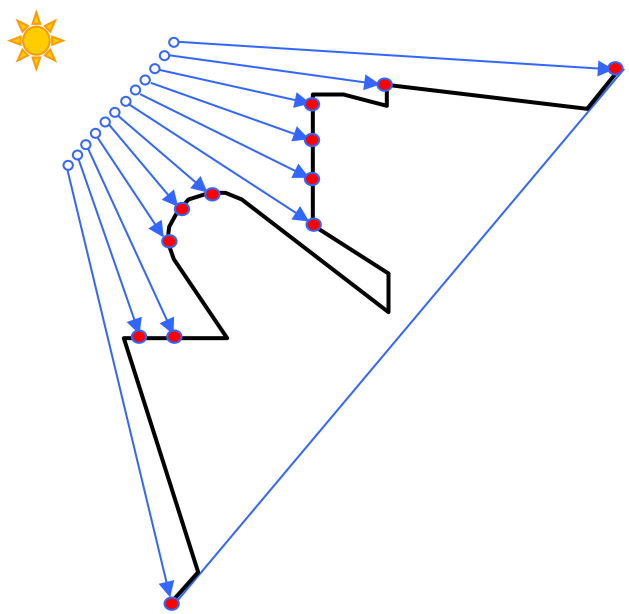


camera depth points (WCS)

VBGI – Image-based Point Injection (4)

- How are the points injected (cont)?
  - Geometry (points) generation:





- Render a planar grid of points.

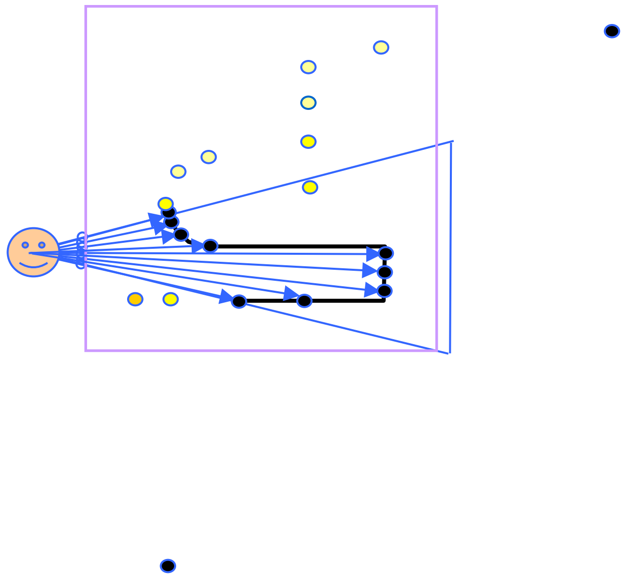
  For simplicity, arrange points in ([0,1],[0,1],0) interval

In a geometry shader:

- Lookup the (x,y) depth from the SM
- Transform (x,y,depth) to vol. coords
- Inject the transformed point in volume

VBGI – Image-based Point Injection (5)

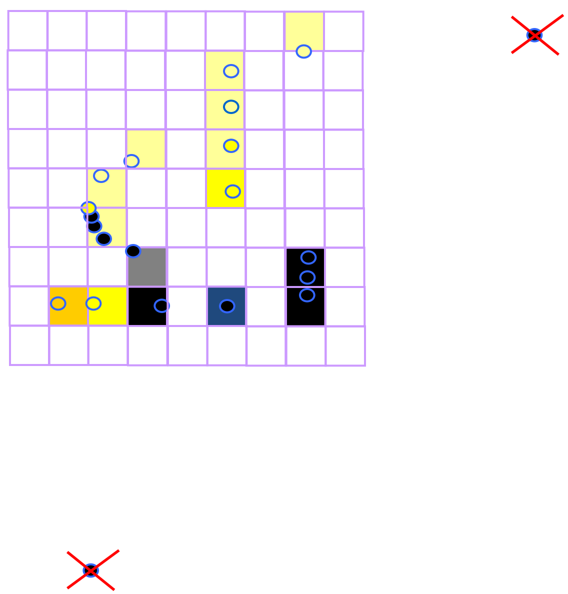
- How are the points injected (cont)?
  - Do the same for the camera buffer points:



- Additional camera points are unlit points
- We repeat the process for all available buffers (lights, reflection buffers, env. maps etc)

VBGI – Image-based Point Injection (6)

- The corresponding voxels now store the encoded lighting, occupancy and other data:

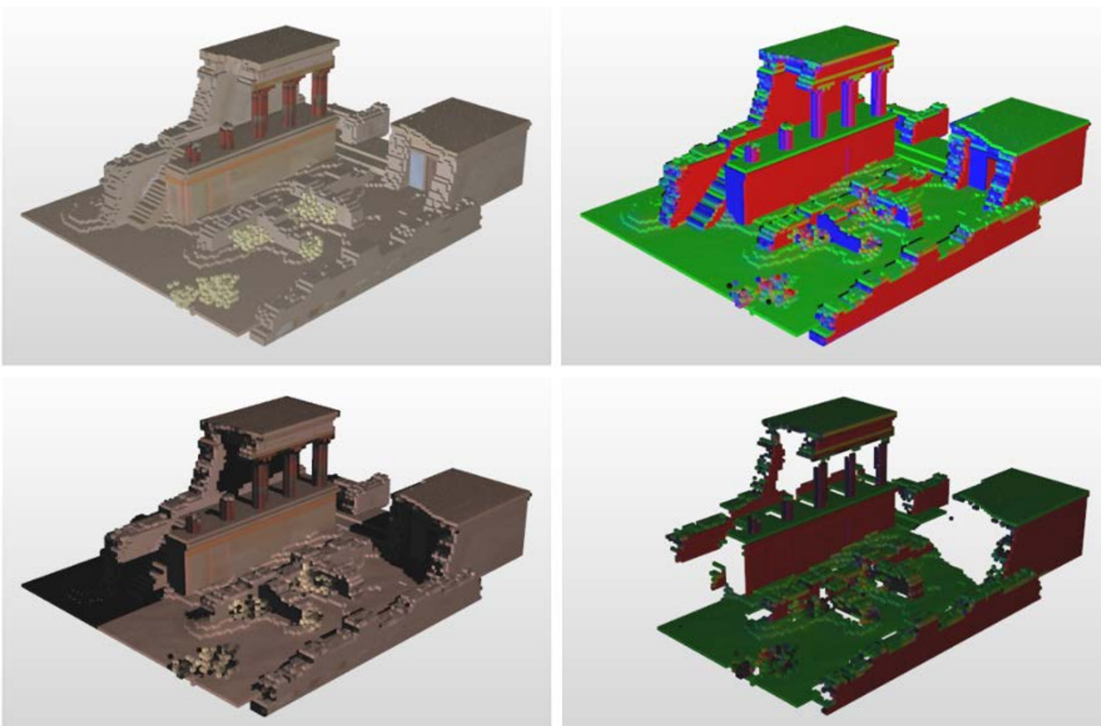


- The injected point contribution is not the same for all points! More on this later

VBGI – Full Scene Voxelization (1)

- Rasterizes the geometry into the volume buffer directly from the geometric data
- Imprints a complete occlusion information, regardless of visibility to buffers
- Voxelization → 3D Rasterization:
  - Voxel shaders compute and encode direct lighting, normals, albedo and occupancy
  - 2-5 volume textures required
- Many ways to perform it
- All methods slice the geometry into volume layers

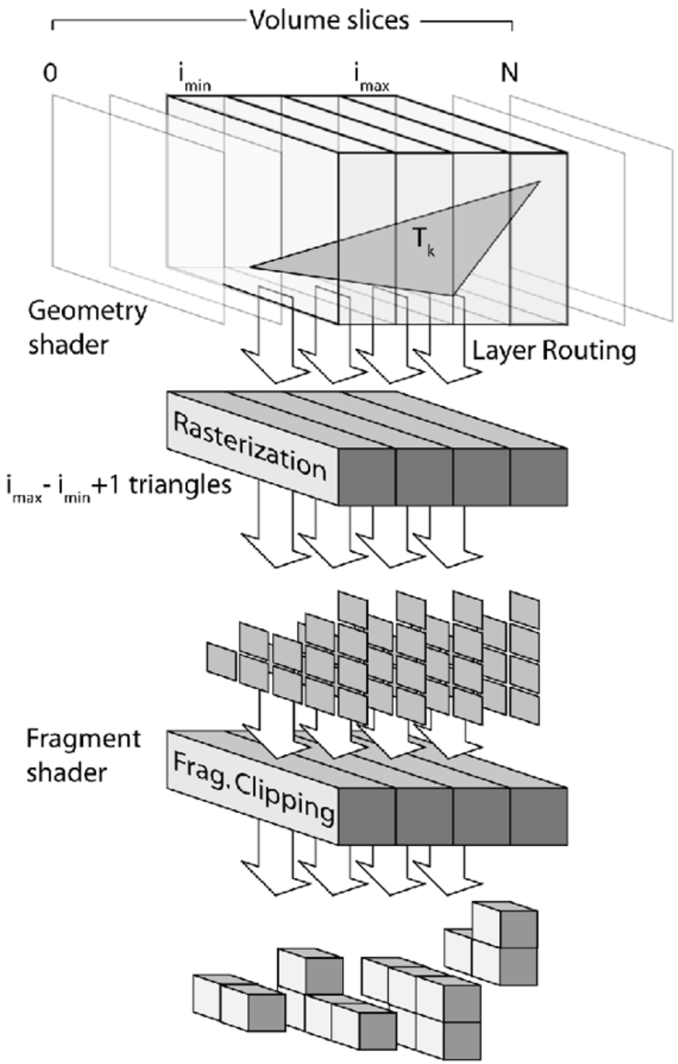
VBGI – Full Scene Voxelization (2)



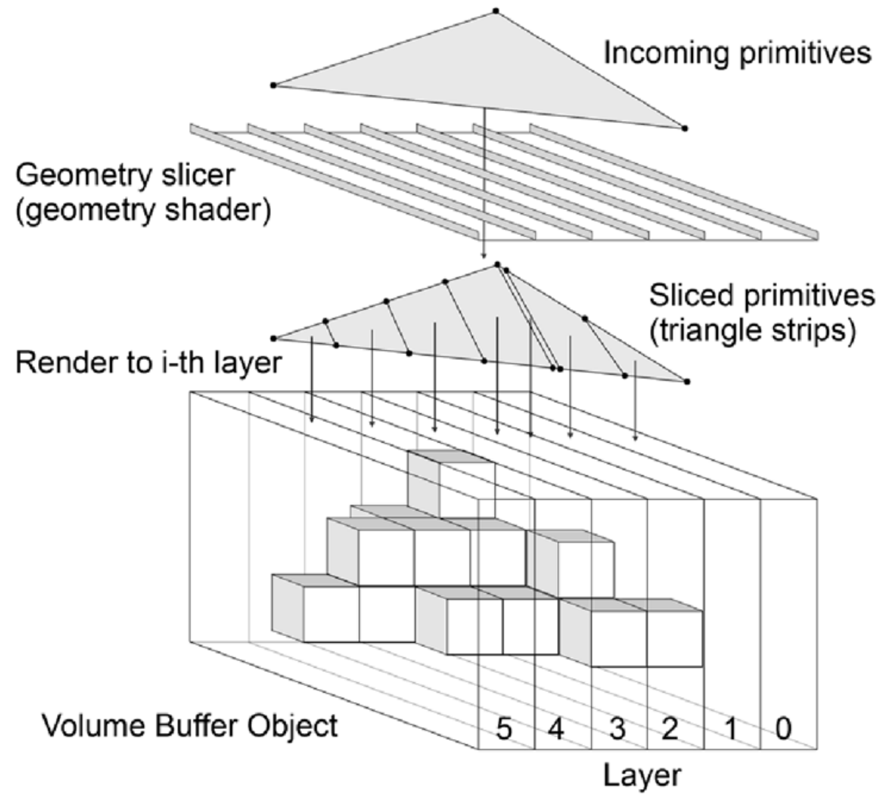
Render targets (volumes)	R	G	B	A	texture channels
0	n_x	n_y	n_z	o	Luminance only
1	s_{00}	s_{1-1}	s_{10}	s_{11}	
0	n_x	n_y	n_z	o	Full color GI
1	sr_{00}	sr_{1-1}	sr_{10}	sr_{11}	
2	sg_{00}	sg_{1-1}	sg_{10}	sg_{11}	
3	sb_{00}	sb_{1-1}	sb_{10}	sb_{11}	
4	c_r	c_g	c_b	c_a	+ color bleeding

VBGI – Full Scene Voxelization (3)

Fragment shader clipping

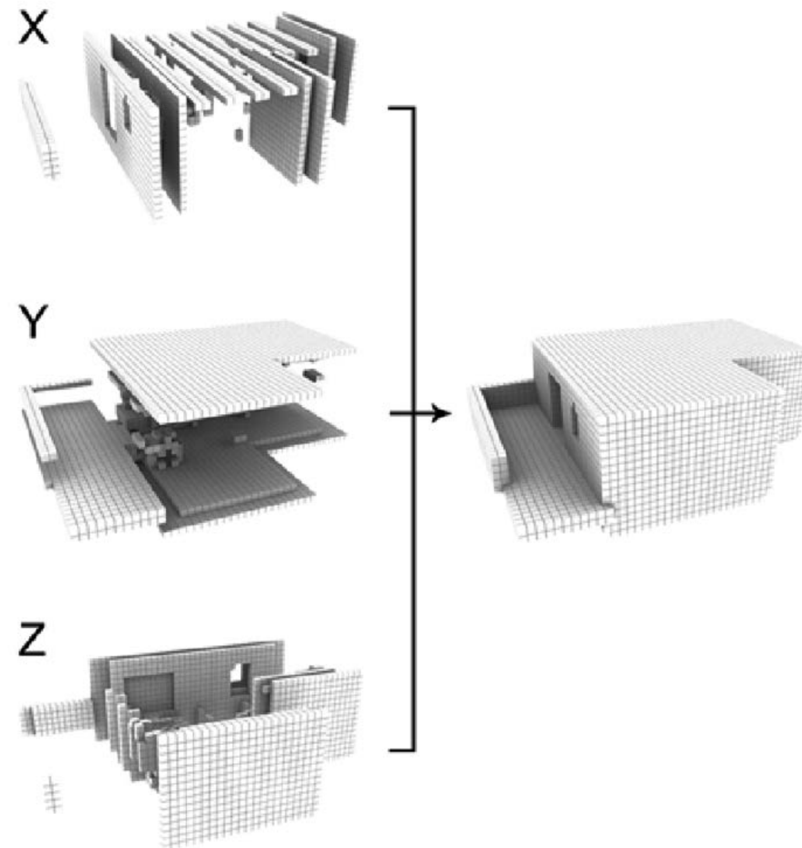


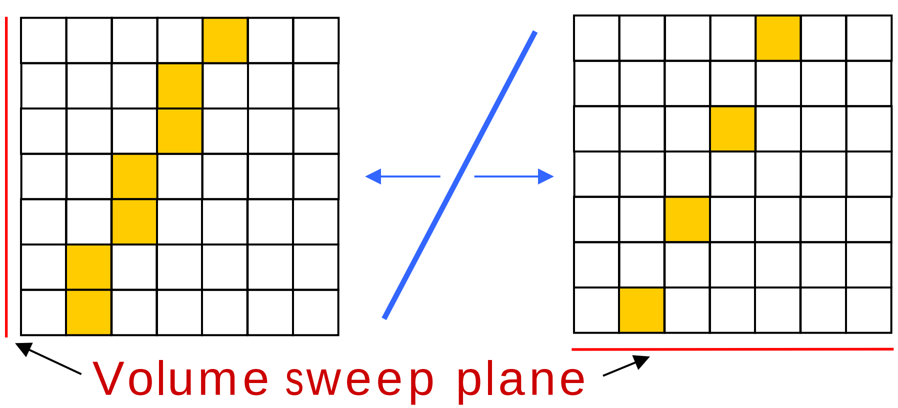
Geometry shader clipping



VBGI – Full Scene Voxelization (4)

- Polygons are rasterized to the volume sweep of maximum projection
- This ensures dense, coherent sampling





Binary data: OR op.   Scalar data: MAX op.

Blocking – Geometry Orientation/Coverage

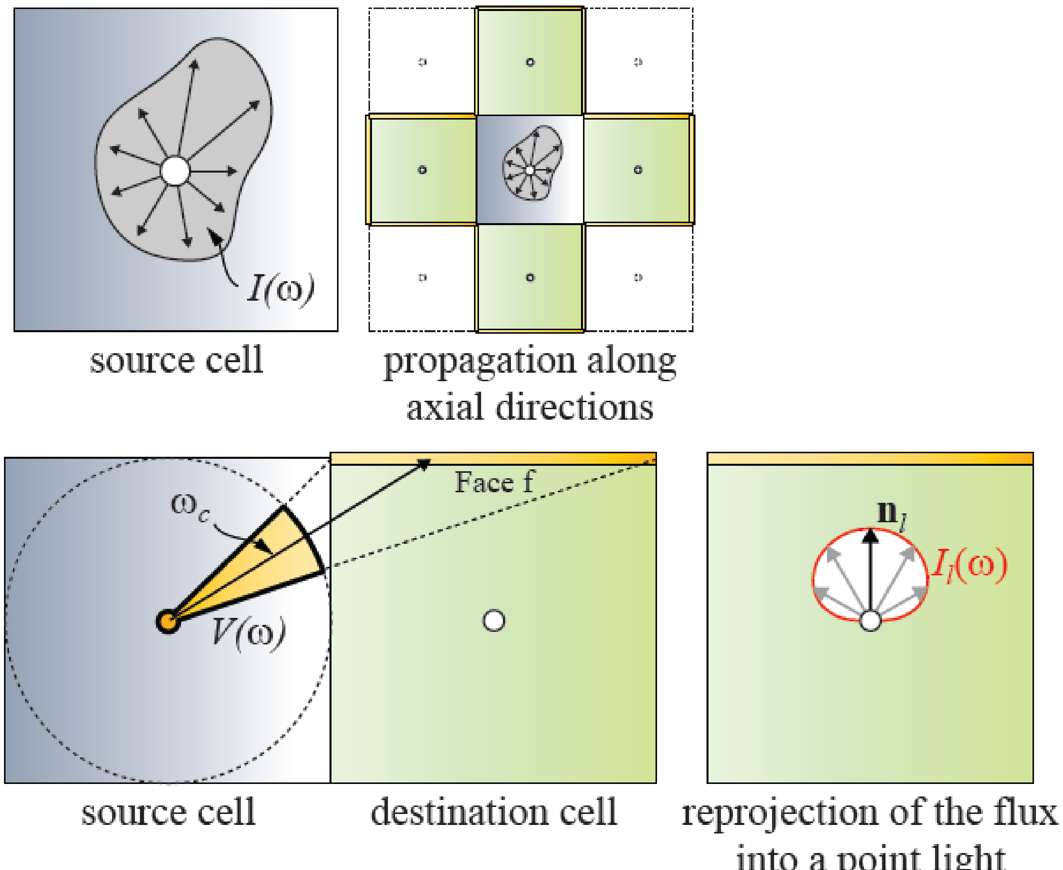
- As volume textures are quite crude (e.g. 32^3), voxels should not be either on or off
- Regardless of volume generation method, volumes should store:
  - Occupancy proportional to voxel coverage and alpha → This is easier in full voxelization
  - Directional data (SHs) for each injected fragment →
    - Multiple surfaces with different orientations cross the voxel

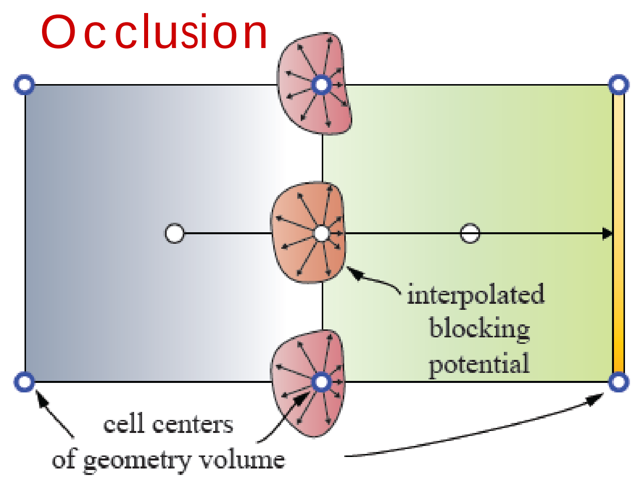
Light Propagation Volumes

Source: http://advances.realtimerendering.com/s2009/Light_Propagation_Volumes.pdf

Light Propagation Volumes (1)

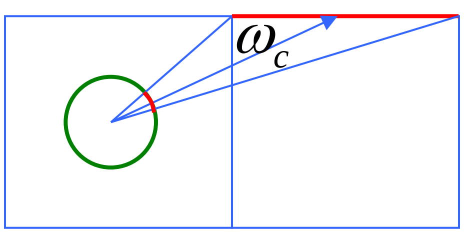
- Iteratively propagates flux from each cell to the next
- Blocks (attenuates) light according to occupancy data





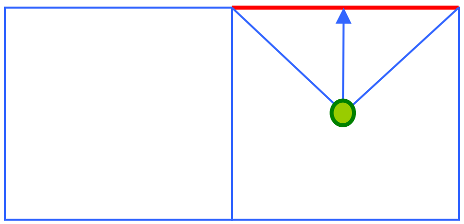
Light Propagation Volumes (2)

- The flux incident to each one of the faces of the neighboring cell is difficult to approximate as an integral using low-order SHs
- A rough empirical approximation is suggested:
  - Estimate the intensity in direction ω_c to the cone V(ω) center
  - Scale by the ratio of the solid angle subtended by the face against 4π (spherical solid angle)



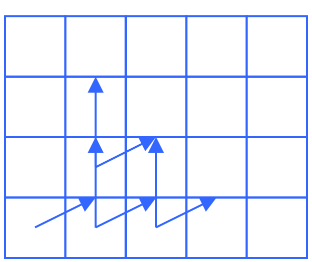
Light Propagation Volumes (3)

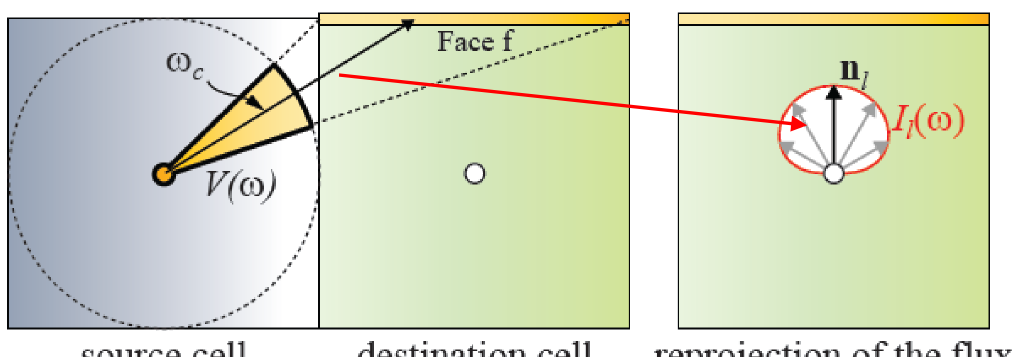
- Then a new VPL is generated at the neighboring cell with intensity matching the total flux of the face
- The VPL is encoded as SH and added to the cells intensity distribution



Light Propagation Volumes (4)

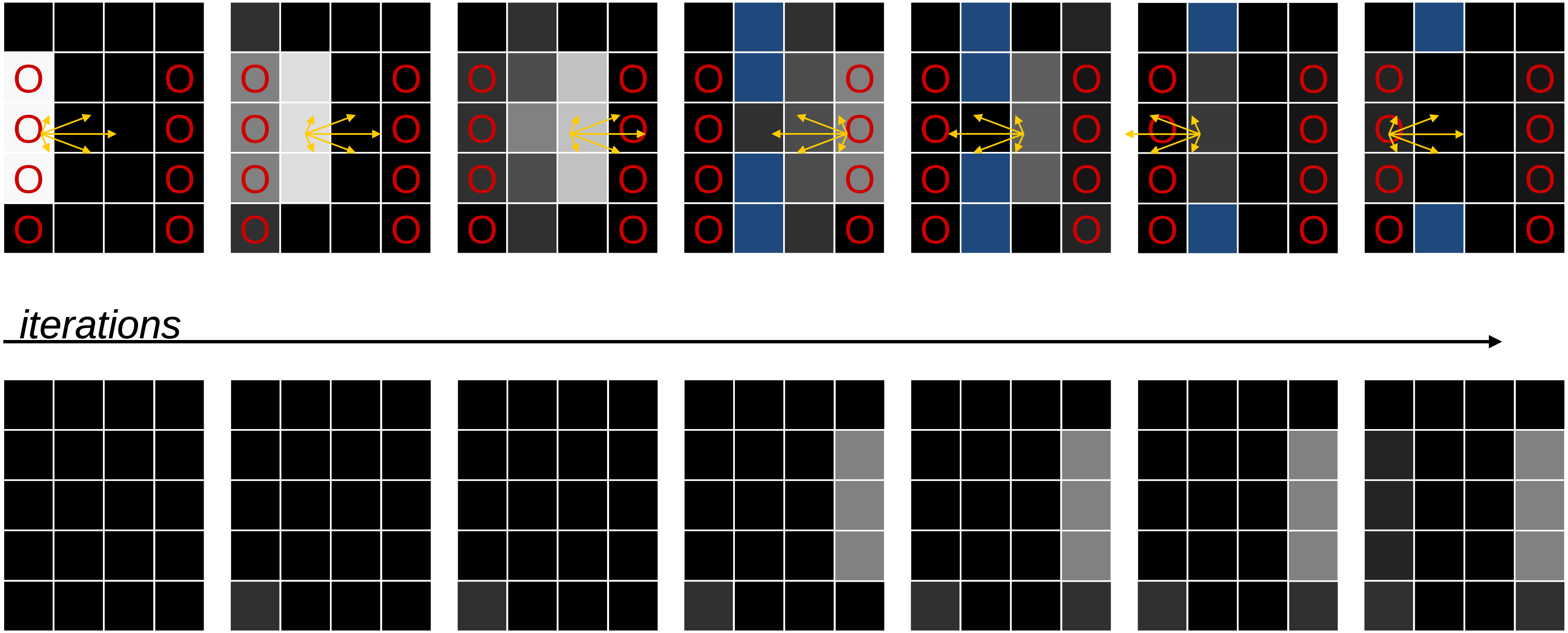
- Not a physically correct solution:
- Although flux balance is maintained,
- Flux is assumed to get diffused on “translucent walls” due to the change in propagation direction





Light Propagation Volumes - Bounces

Spherical harmonic buffer (pair – swapped for reading/writing)

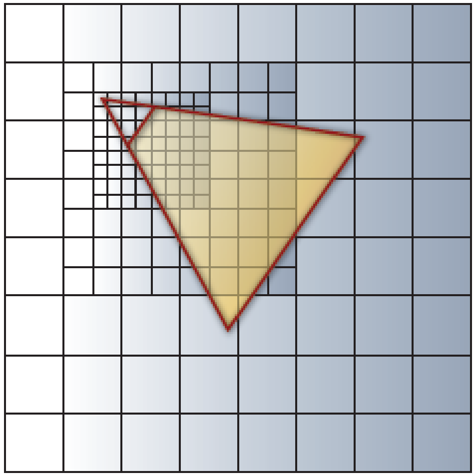


GI accumulation buffer (flux sampled from decoded SH)

- Some leaking still occurs due to low SH order (series truncation) and approximate blocking

Cascaded LPVs

- Why?
  - Scenes are large to be covered by a single low-res volume (large volumes are slow and costly)
  - We need many iterations to transport flux across the scene
- Solution: Cascades
  - Overlapped volumes of same resolution but different size
  - Denser sampling near camera



VBGI - Ray Marching

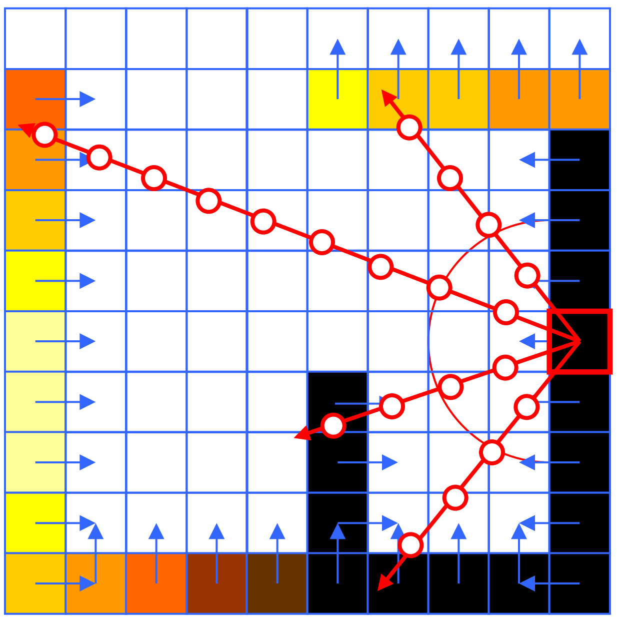
VBGI - Ray Marching (1)

- We can approximate a gathering operation (Monte Carlo integration) by marching rays in the volume instead of intersecting them with the scene
- We can march rays either from the shaded fragments or from the GI volume voxels (faster but cruder)

VBGI - Ray Marching (2)

- Ray marching:
  - Iteratively sample the volume along a line until a fully blocked voxel is reached
  - Gather light along the line from occupied voxels, according to orientation stored in them
  - Perform integration with the BRDF at the shaded point → Simple SH dot product for diffuse reflection

VBGI - Ray Marching (3)

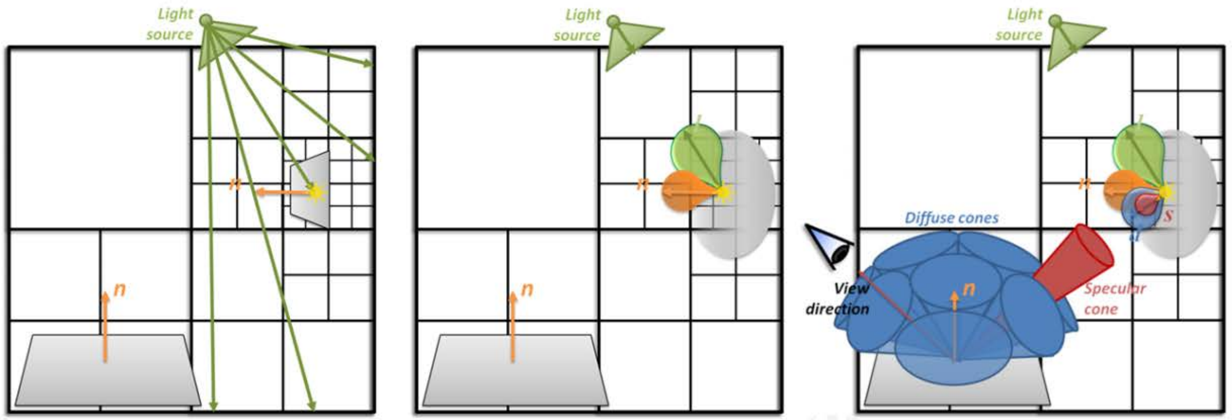


```
Generate N random rays
L_gi = 0;
for each ray dir:
    s = ds;
    while s < r_max
        v = p + s*dir;
        if Occ(v)>0.5
            break;
        s += ds;
    F = clamp(dot(-Normal(v),dir),0,1);
    F *= clamp(dot(Normal(p),dir),0,1);
    L_gi += F*L(v)/((v-p)*(v-p));
L(p) += Color(p)*L_gi/N;
```

Cone Tracing

- Extending the idea of ray marching, instead of tracing a number of rays over the hemisphere to compute irradiance, we can trace bunches of rays grouped in cones → fewer queries
- The cone radius increases with distance to shaded point
- The conical section at a given distance should be used as a filter kernel to gather outgoing radiance from all touched surfaces
- Outgoing radiance can be pre-filtered and hierarchically stored

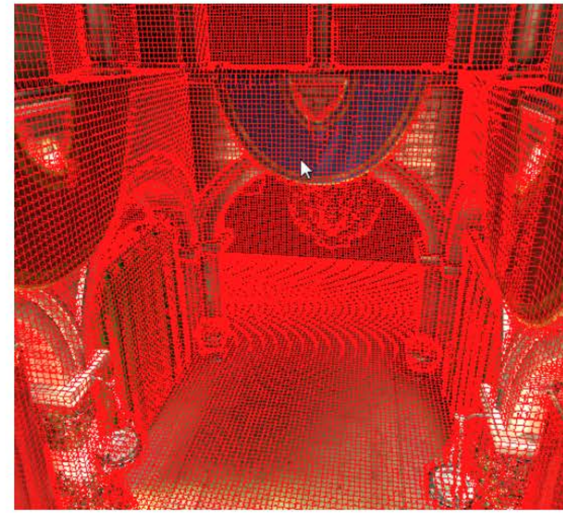
Voxel Cone Tracing



Step 1: Render from light sources. Bake incoming radiance and light direction into the octree

Step 2: Filter irradiance values and light directions inside the octree

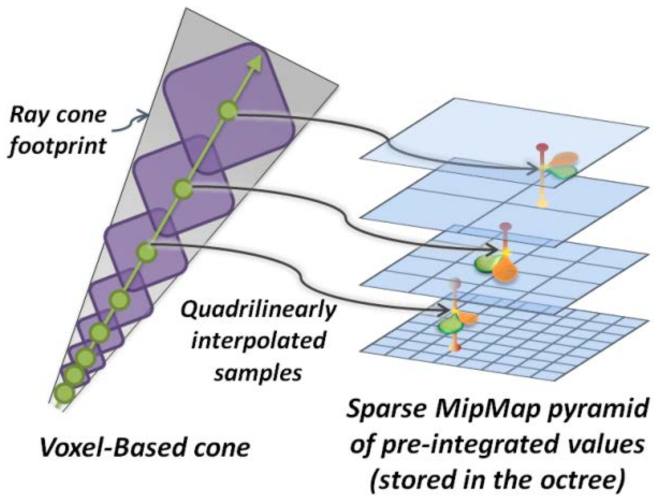
Step 3: Render from camera. Sample diffuse + specular BRDF components using voxel based cone tracing



- Record and pre-filter direct illumination on a hierarchical voxel grid
- Advance a ray at each cone axis in the hierarchical occupancy grid
- Choose appropriate voxel LOD according to current step cone radius
- Gather averaged radiance for each traced cone

Source: https://research.nvidia.com/sites/default/files/pubs/2011-09_Interactive-Indirect-Illumination/GIVoxels-pg2011-authors.pdf

Voxel Cone Tracing



Screen-space Reflections (SSR)

- Idea: Reuse already rendered content as shaded hit locations for reflected rays
- Perform screen-space ray marching using the depth buffer to locate hit points

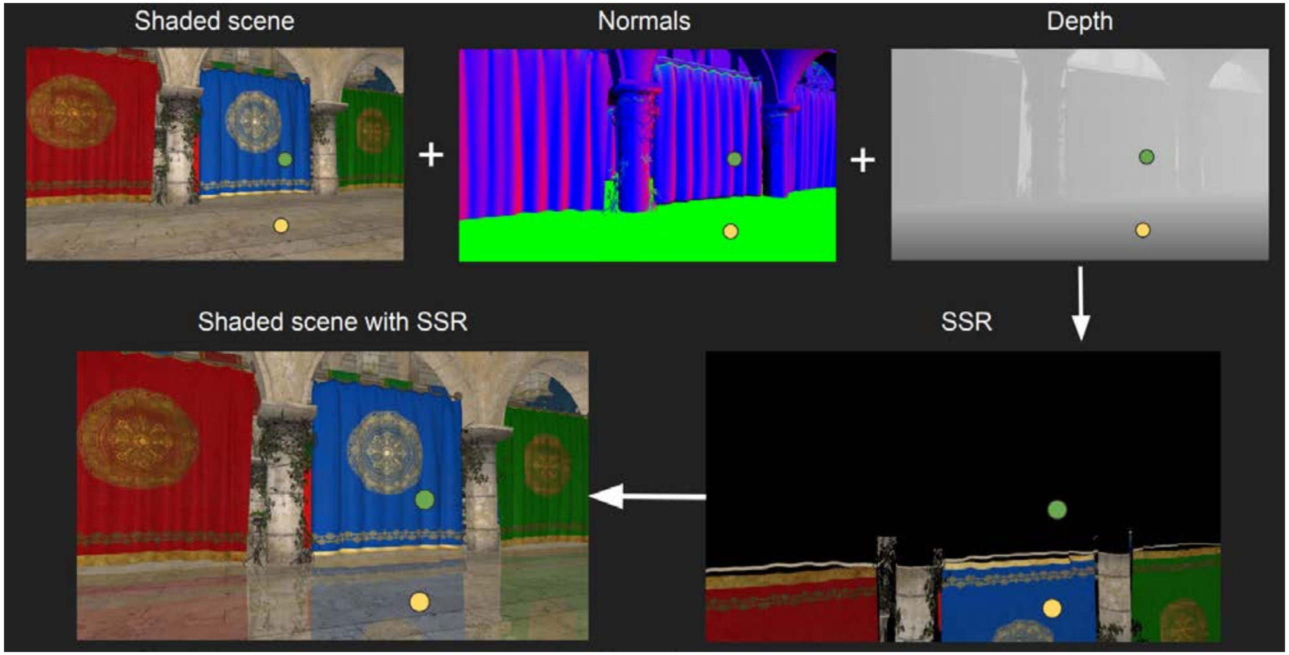
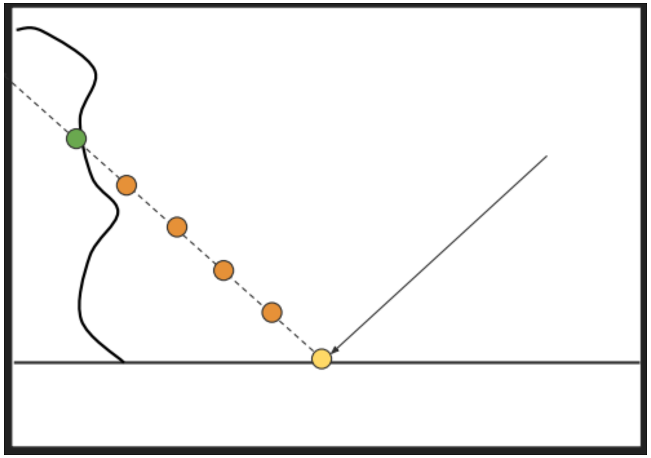


Image source: http://www.cse.chalmers.se/edu/year/2018/course/TDA361/Advanced%20Computer%20Graphics/Screen-space%20reflections.pdf

Screen-space Reflections (SSR)

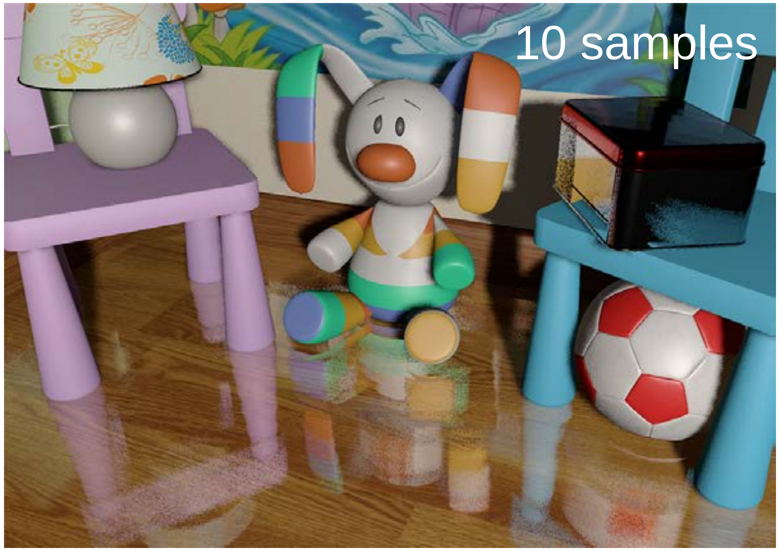
Linear search:

- March along reflected ray in constant strides
- In each step, check depth of ray sample against the depth buffer
- Stop at transition behind visible depth range
- Optionally, refine solution (e.g. bisection)
- Obtain hit point color and normal
- Calculate radiance to shaded point



Screen-space Reflections (SSR)

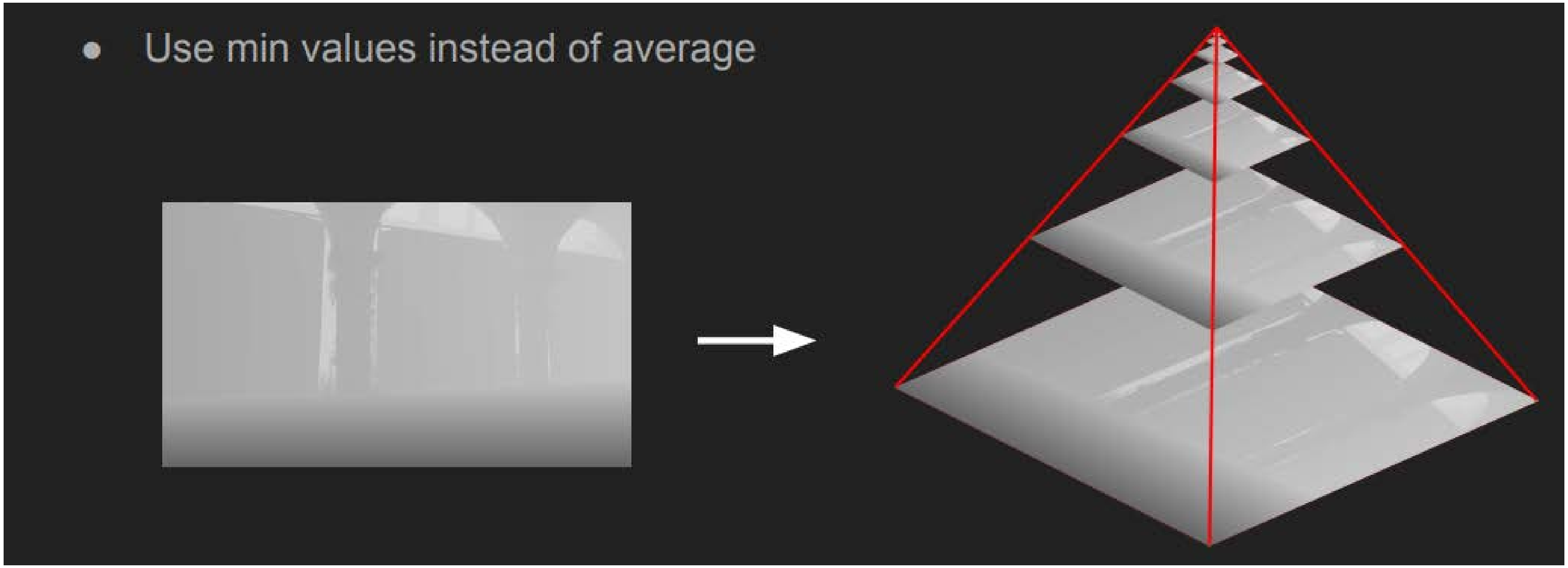
- Linear search requires many samples (expensive)
- With few samples, there is high probability of missing the correct hit point



Screen-space Reflections (SSR)

Hierarchical search:

- Build depth mip-maps (cluster depth values using MIN operation)



Screen-space Reflections (SSR)

- Traverse the depth buffer with adaptive strides, moving up/down the depth LODs

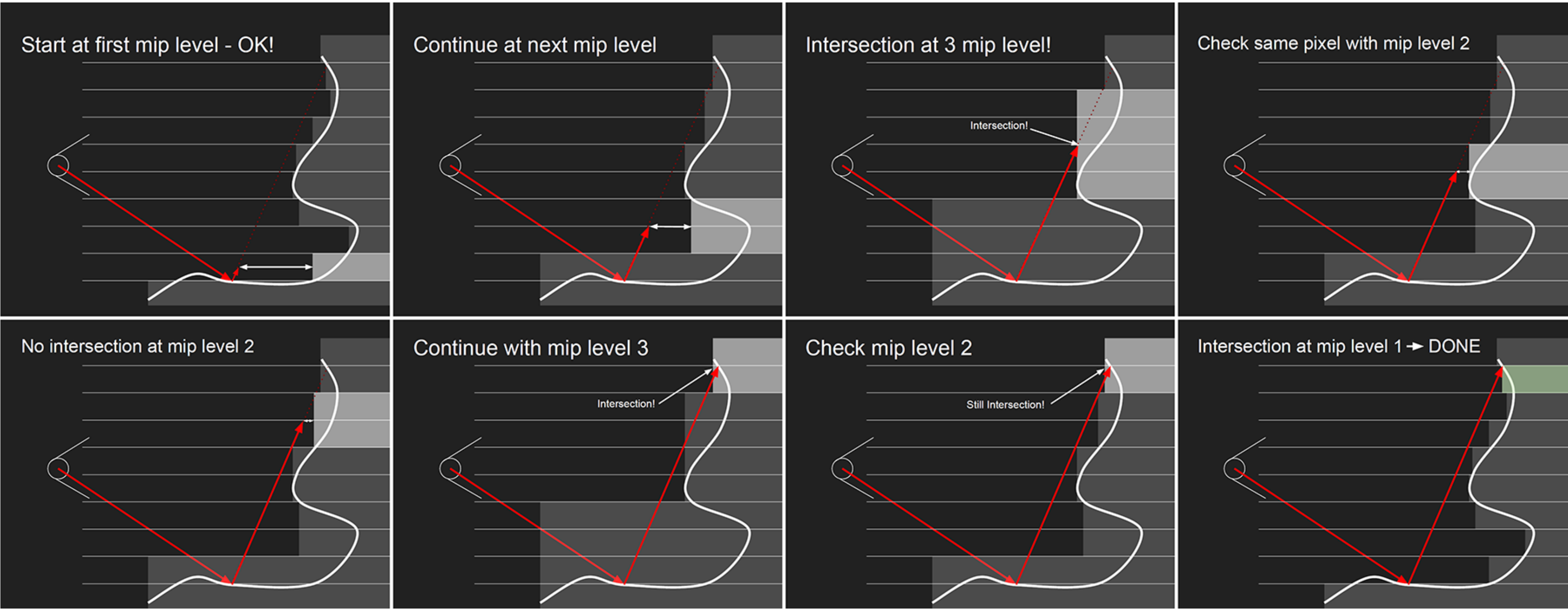


Image source: http://www.cse.chalmers.se/edu/year/2018/course/TDA361/Advanced%20Computer%20Graphics/Screen-space%20reflections.pdf

SSR – Quality vs Performance

- SSR has plenty of room for performance optimization
  - Switch between sparse linear and expensive accurate hierarchical marching according to BRDF
  - Trace reflections at different resolution and upscale
  - Mix with environment maps
  - Mipmap screen-space MRTs to simulate cone tracing for glossy BRDFs
  - …

Problems of SSR

- SSR cannot capture geometry that is not present in the view
  - Hidden depth layers not captured by the Z buffer (left)
  - Off-screen information (right)

Image source: http://www.cse.chalmers.se/edu/year/2018/course/TDA361/Advanced%20Computer%20Graphics/Screen-space%20reflections.pdf

Ray-traced Directional GI

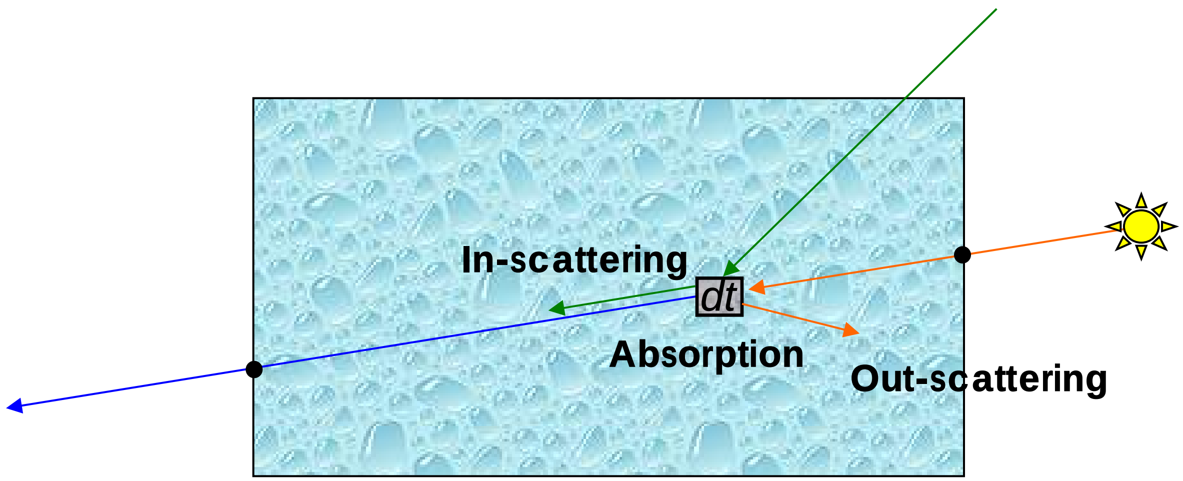
- Ray marching can nowadays be replaced by true ray tracing
- Still expensive, we use it sparsely
- Mainly solves the problem of absence of geometric information in the buffers
- Can be used as evaluation method for all the GI techniques discussed above (baked and real-time)
- Cons:
  - Requires high-end hardware
  - Consumes more memory

Image source: https://www.nvidia.com/en-us/geforce/community/demos/

Volumetric Rendering

- 4 phenomena affect light traveling through a medium:
  - Absorption
  - Out-scattering
  - Emission
  - In-scattering

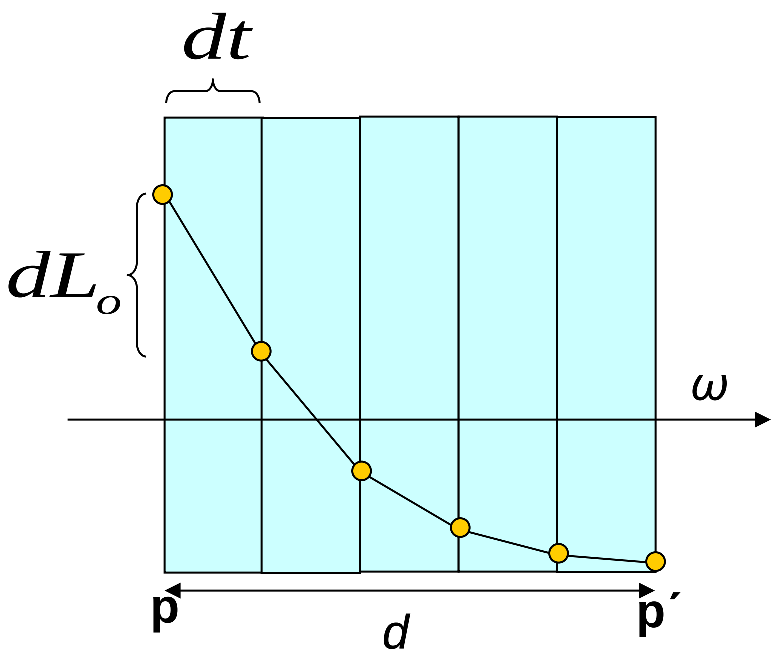
} Attenuation



Attenuation

$$L_o(\mathbf{p},\omega) - L_i(\mathbf{p},\omega) = dL_o(\mathbf{p},\omega) \Rightarrow \frac{dL_o(\mathbf{p},\omega)}{dt} = -\sigma(\mathbf{p},\omega)L_i(\mathbf{p},-\omega)$$

$$\sigma(\mathbf{p},\omega) = \sigma_a(\mathbf{p},\omega) + \sigma_s(\mathbf{p},\omega)$$



Transmittance:
Fraction of light transmitted from **p** to **p´**

$$T_r(\mathbf{p} \to \mathbf{p}') = e^{-\int_0^d \sigma(\mathbf{p}+t\omega,\omega)dt}$$

Beer's Law

- For constant σ (homogeneous medium), transmittance becomes:

$$T_r(\mathbf{p} \to \mathbf{p}') = e^{-\sigma d}$$

- If absorption is constant along small ray segments: ,from Beer's law and the definition of transmittance we get:

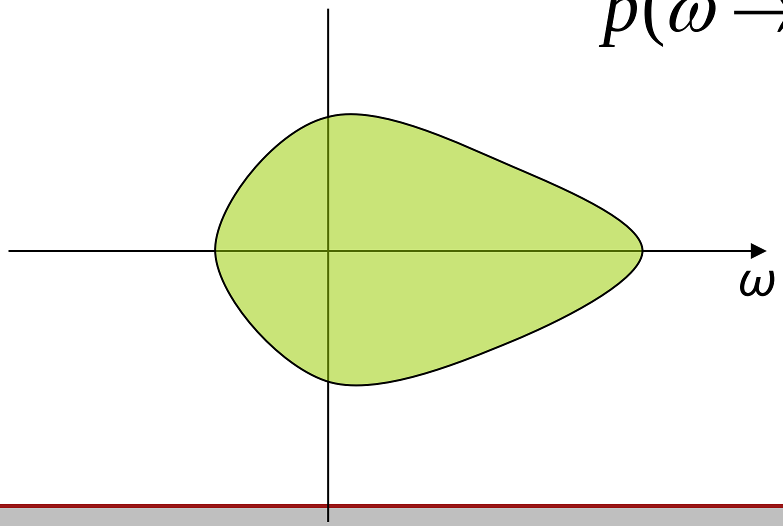
$$T_r(\mathbf{p}_1 \to \mathbf{p}_N) = e^{-(\sigma_1 d_1 + \sigma_2 d_2 + \ldots + \sigma_{N-1} d_{N-1})} \Leftrightarrow$$

$$T_r(\mathbf{p}_1 \to \mathbf{p}_N) = \prod_{i=1}^{N-1} T(\mathbf{p}_i \to \mathbf{p}_{i+1})$$

In-scattering – Phase Functions

- The directional distribution of scattered light at a point is called a **phase function**.
- It is similar to the BSDF but expresses the probability that light from ω is deflected towards ω' :

$$p(\omega \to \omega'): \quad \int_S p(\omega \to \omega')\, d\omega' = 1$$



In-scattering – Phase Functions (2)

- Popular phase functions:
  - Isotropic

$$p_{isotropic}(\omega \to \omega') = \frac{1}{4\pi}$$

  - Henyey-Greenstein

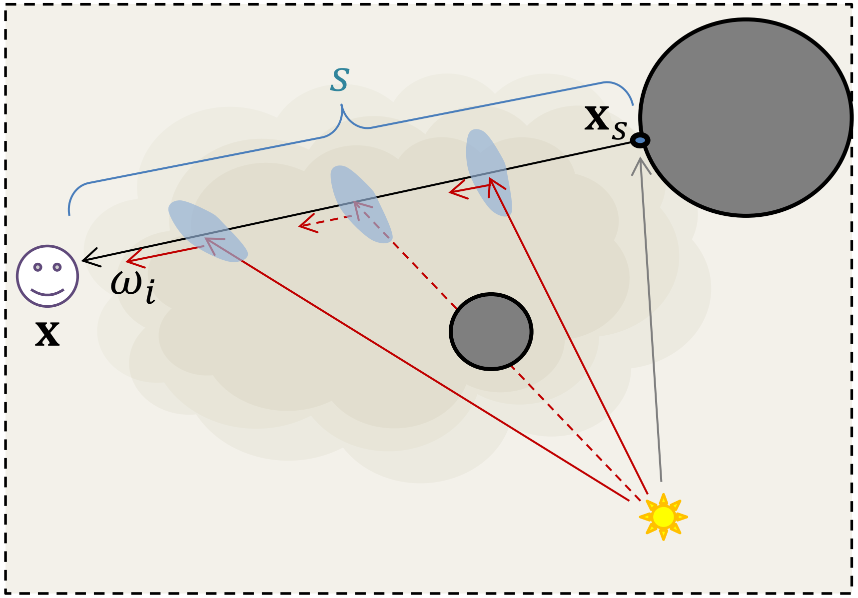
$$p_{Henyey-Greenstein}(\omega \to \omega') = \frac{1}{4\pi} \frac{1-g^2}{\left(1+g^2-2g\cos\theta\right)^{3/2}}$$

  - Mie (atmosphere)
  - Rayleigh (droplets, steam etc)

Combining Out/In-scattering

Extinction/absorption In-scattering

$$L_i(\mathbf{x},\omega_i) = T(\mathbf{x}_s \to \mathbf{x})L_e(\mathbf{x}_s,\omega_i) + \int_0^s T(\mathbf{x}_t \to \mathbf{x})L_{scatter}(\mathbf{x}_t,\omega_i)dt$$



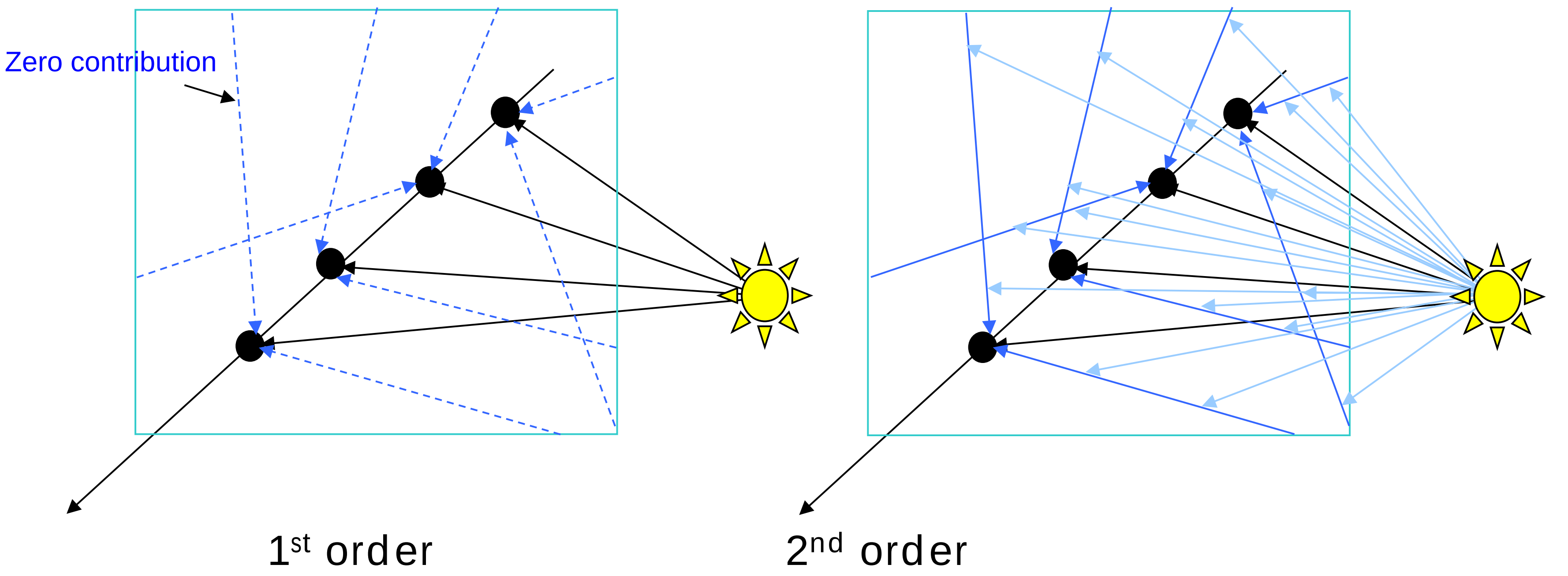
$$T(\mathbf{x}_s \to \mathbf{x}) = e^{-\int_0^s \sigma_t(x)dt}$$

Recursive form

$$L_{scatter}(\mathbf{x},\omega_i) = \sum_{\mathbf{l}\ \text{lights}} p(\omega_i,\mathbf{x}\to\mathbf{l})\ V(\mathbf{x},\mathbf{l})\ L_i(\mathbf{x},\mathbf{l}\to\mathbf{x})$$

In-scattering Equation

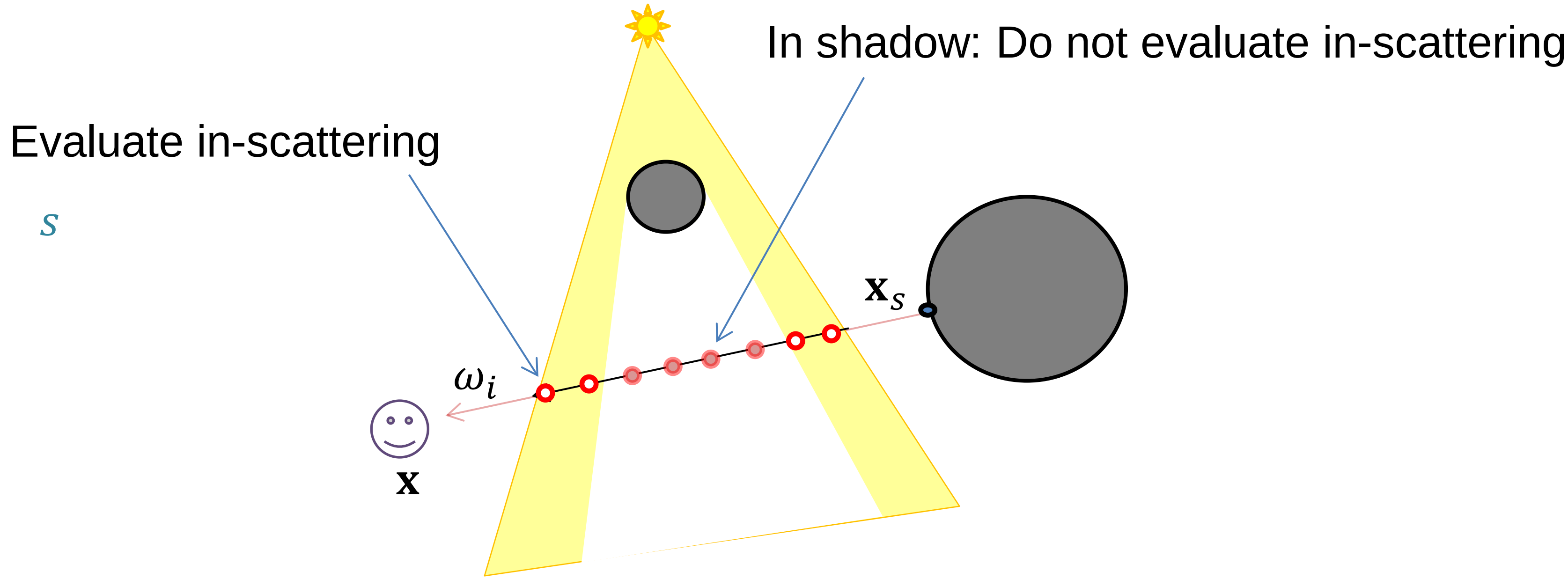
- In-scattering equation is actually computed recursively, although usually 1-2 levels are used:



Volumetric Shadows

- In-scattering can create very interesting “godray” effects and realistic fog
- The most common approach to achieve volumetric shadows is via ray marching on the shadow map

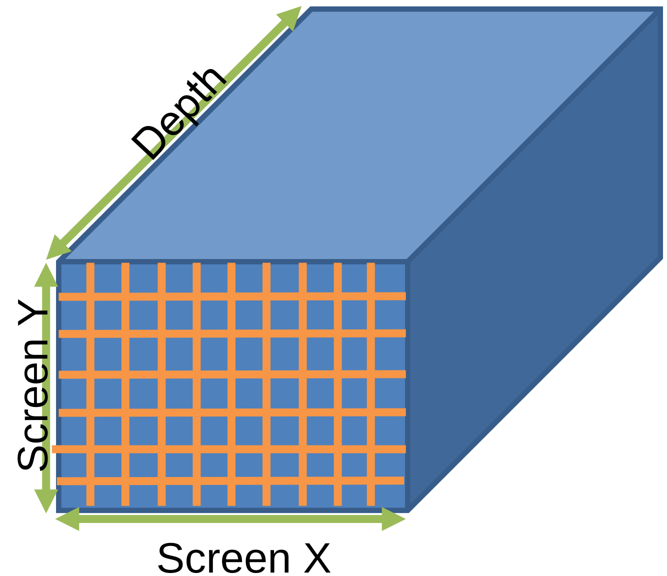
Volumetric Shadows: Ray Marching

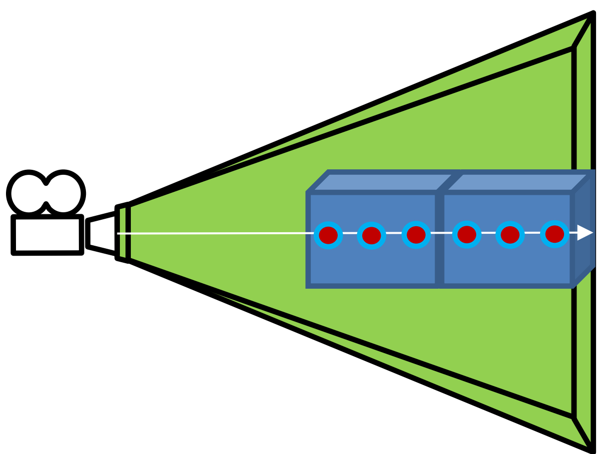


- Keep samples within the shadow volume extents
  - Otherwise, they will be thinly spread along large distances → very poor sampling → aliasing
- Jitter samples per ray and over time to avoid banding artefacts

Volumetric Shadows: Volume Caching

- Used by the Frostbite engine
- Idea: Generate a view-aligned (clip-space) low-res volumetric grid
  - Sample materials (+emissive particles and surfaces), particles and participating media and store scattering coefficients in volume cells
  - Sample sources and store directional distribution of out-scattered light in each cell (due to phase function) in SH
  - Reconstruct volume rendering integral per pixel, exploiting interpolation





Volumetric Shadows: Volume Caching

STYLIZED RENDERING

Stylized Rendering

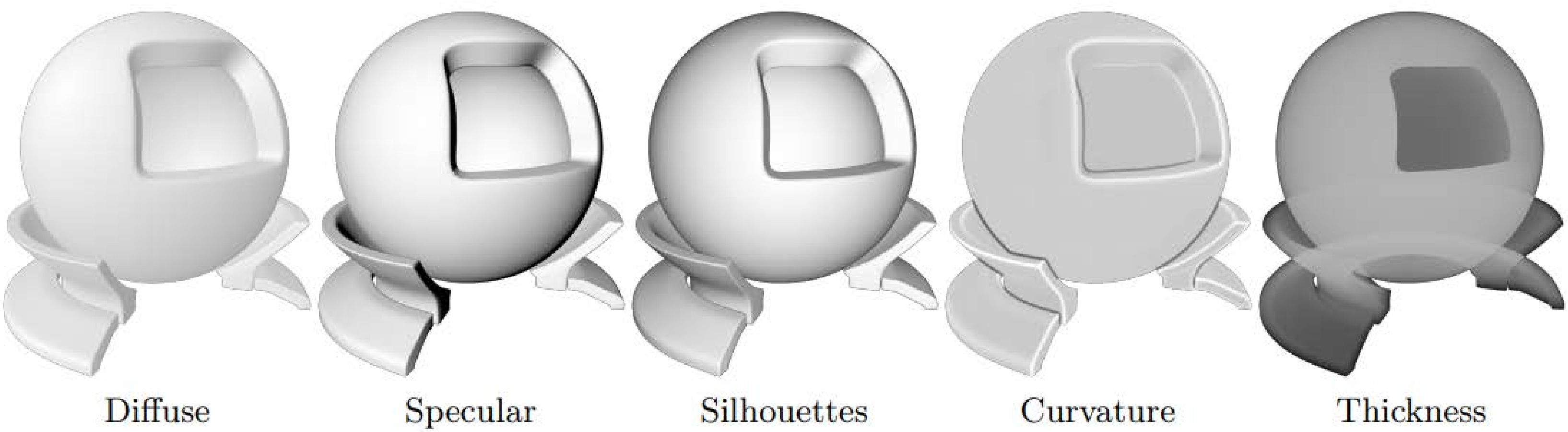
- Games often dispense with realistic models to simulate a comic book look and feel
- Many effects discussed so far still apply, but surface shading is altered to combine irradiance in a different, non-physically-based manner

Cell Shading

- Cell shading has two main characteristics:
  - Simplified BRDF response, with pen-and-ink separated highlights, base color and shadowed regions
  - Strong sketch-like silhouettes
- Additionally:
  - Artificial color bleeding from extra lights and effects
  - Intentional rim lighting
  - Post-processing effects for masking, stippling, color grading and saturation, etc.

Cell Shading Dependencies

- To compute borders and extra highlights, we need to generate and access extra information, such as:
  - Depth discontinuities (depth buffer derivatives)
  - Screen-space (or object-/texture-space static) curvature
  - Normal gradients
- Deferred pipelines can easily provide the above

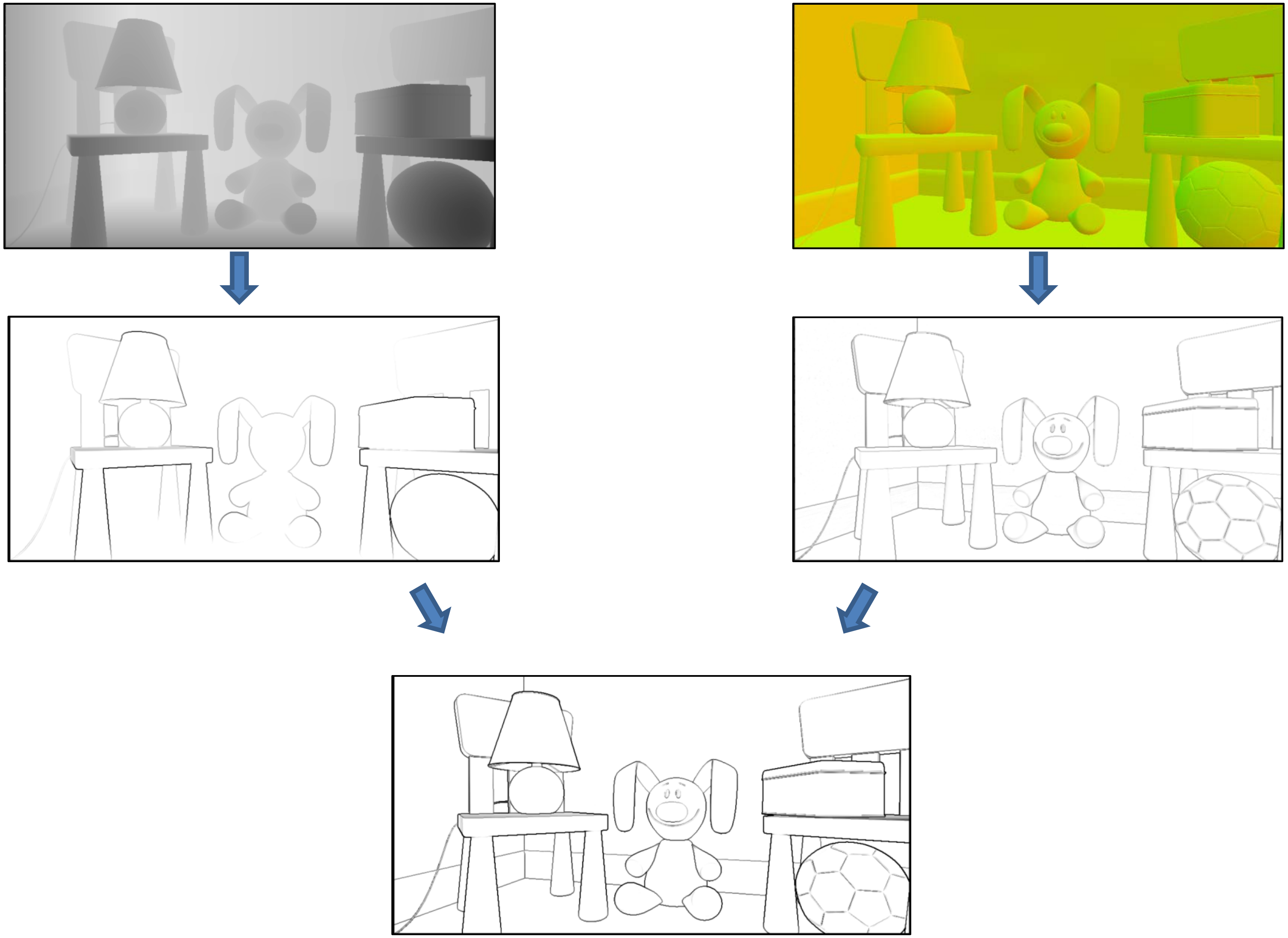


Source: https://hal.inria.fr/hal-01919501/document

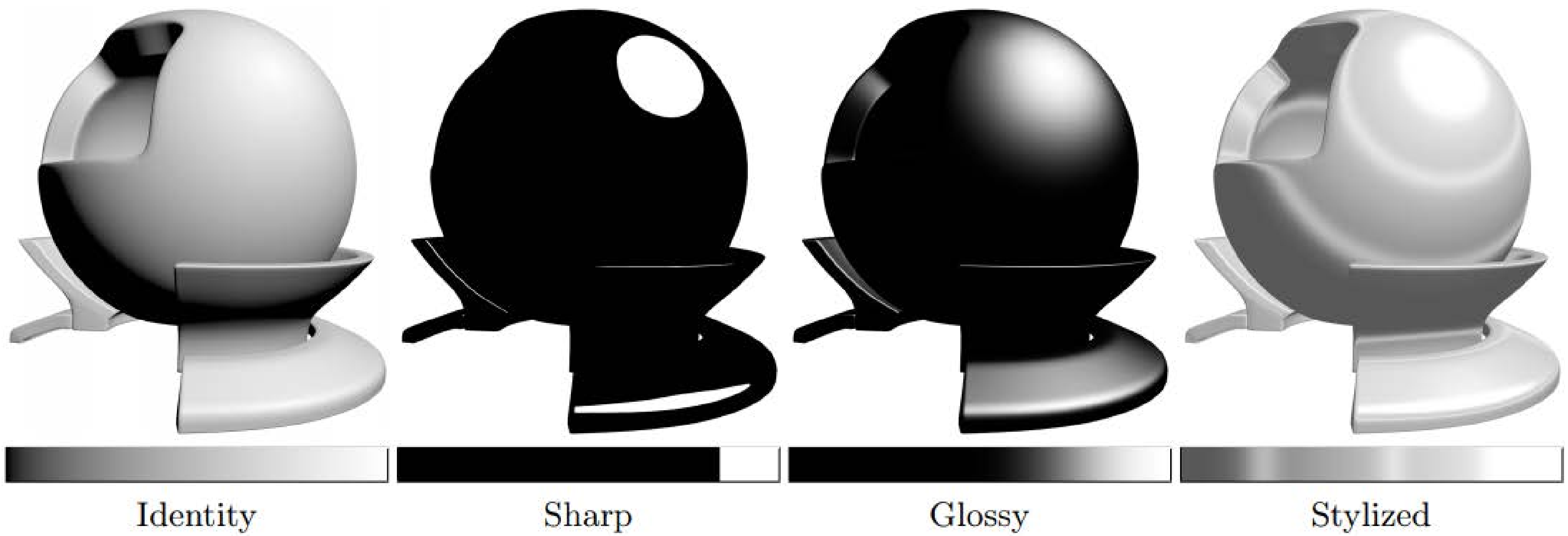
NPR - Silhouettes

- Derive surface saliency (edge) from:
  - Pixel depth gradients: $e_1 = max\left\{\left|\frac{\partial z}{\partial x}\right|, \left|\frac{\partial z}{\partial y}\right|\right\}$
  - Normal buffer gradients: $e_2 = L_\infty(\nabla \mathbf{n})$
  - Object ID bundaries
  - Other (e.g. curvature peaks from screen-space AO)

NPR - Silhouettes



NPR – Highlight Response Curves



- Given a default cosine-weighted (diffuse) surface shading, we can use custom response curves to create artificial highlights

Additional Reading

- Moving Frostbite to Physically Based Rendering 3.0 https://seblagarde.files.wordpress.com/2015/07/course_notes_moving_frostbite_to_pbr_v32.pdf
- Real Shading in Unreal Engine 4 https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf

Contributors

- Georgios Papaioannou