Real-time Graphics

4. Global Illumination

Martin Samuelčík
Juraj Starinský

Rendering equation

$$L_o(\mathbf{x},\omega,\lambda,t) = L_e(\mathbf{x},\omega,\lambda,t) + \int_\Omega f_r(\mathbf{x},\omega',\omega,\lambda,t) L_i(\mathbf{x},\omega',\lambda,t)(-\omega' \cdot \mathbf{n}) d\omega'$$

- λ is a particular wavelength of light
- t is time
- $L_o(\mathbf{x},\omega,\lambda,t)$ is the total amount of light of wavelength λ directed outward along direction ω at time t, from a particular position $\mathbf{x}$
- $L_e(\mathbf{x},\omega,\lambda,t)$ is emitted light
- $\int_\Omega \cdots d\omega'$ is an integral over a hemisphere of inward directions
- $f_r(\mathbf{x},\omega',\omega,\lambda,t)$ is the proportion of light reflected from ω' to ω at position $\mathbf{x}$, time t, and at wavelength λ
- $L_i(\mathbf{x},\omega',\lambda,t)$ is light of wavelength λ coming inward toward $\mathbf{x}$ from direction ω' at time t
- $-\omega' \cdot \mathbf{n}$ is the attenuation of inward light due to incident angle

- Global illumination: contribution of neighboring scene points to illumination
- Ambient occlusion, shadows, ray-tracing, radiosity, photon mapping, path tracing, reflections, refractions, caustics, ...

Ambient term

- Simulating light scattered many times by environment
- There are surface points with different number of accumulating rays (plane parts vs. corners)
- Ambient light is NOT constant for all points
- Perceptual clues – depth, curvature, spatial proximity

Ambient occlusion

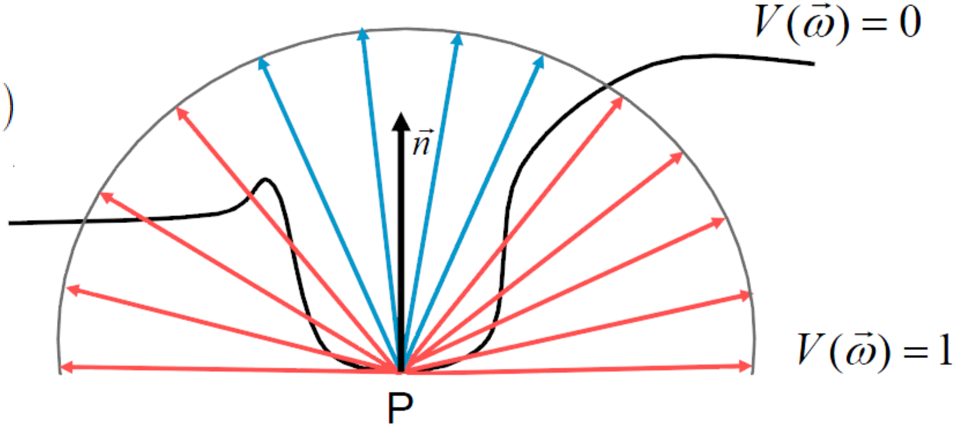
Ambient occlusion

- For illuminating point P, compute visibility from P in all hemisphere directions

$$AO(P,\vec{n}) = \frac{1}{\pi}\int_{\Omega} V(P,\vec{\omega})\cdot max(\vec{n}\cdot\vec{\omega},0)\,d\vec{\omega}$$

$$P - illuminated\ point$$
$$\vec{n} - normal\ at\ point\ P$$
$$V(P,\vec{\omega}) - visibility\ function$$

$$AO(P,\vec{n}) = \frac{1}{\pi}\sum_{\Omega} V(P,\vec{\omega})\cdot max(\vec{n}\cdot\vec{\omega},0)$$



AO computation

- Monte Carlo – computing integral by sampling hemisphere with random rays

$$AO(P,\vec{n}) = \frac{1}{n}\sum_{i=0}^{n-1} V(P, r\vec{nd}_i\omega)\cdot max(\vec{n}\cdot r\vec{nd}_i\omega, 0)$$

$$r\vec{nd}_i\omega = i - th\ random\ vector$$

- Distribution of random rays?
- Still not real-time
- Static geometry & lights – offline precomputation of ambient maps

AO – object space

- Compute intersection with of ray from illuminating point with objects in scene
- Intersection with hierarchical simplified geometry
- Intersection only with close objects
- Usually computation per-vertex
- Performance dependent on geometry complexity

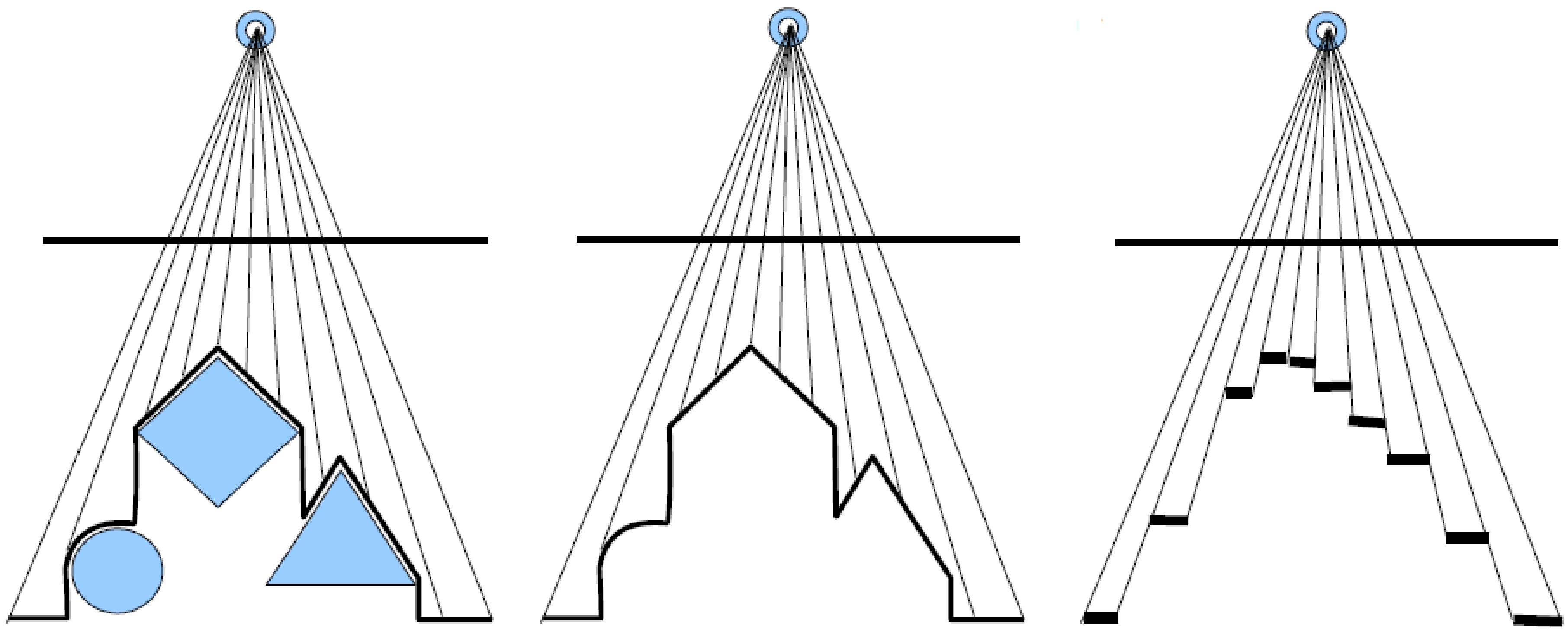
AO – screen space

- Computation of visibility function in screen space, using data about pixels (fragments)
- State of the Art
- Post-processing effect
- Geometry independent
- Requires (1. pass)
  - Pixel depth values
  - Pixel normal values

Blizzard Entertainment

Depth buffer

- Approximation of visible geometry

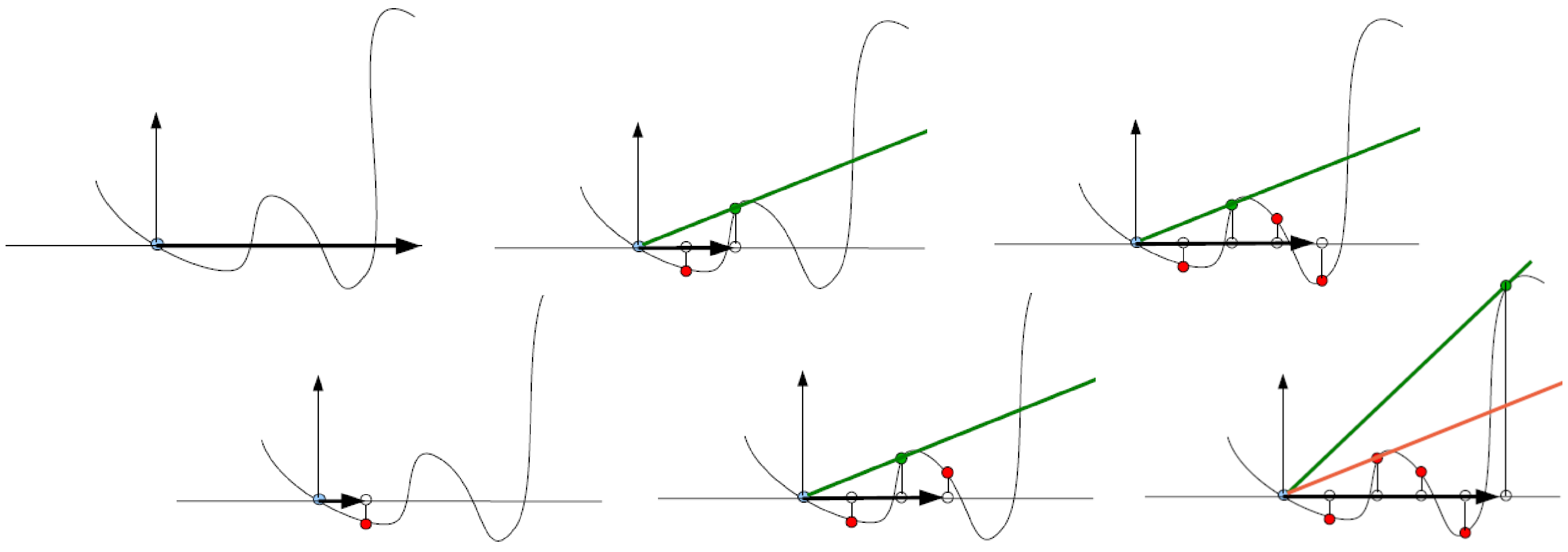


SSAO

- Screen-space ambient occlusion
- Computation of visibility functions from depth values and from normals
- For illuminating point P, sample depth values in P neighborhood and approximate visibility function
- Sampling first in given direction and given region of interest

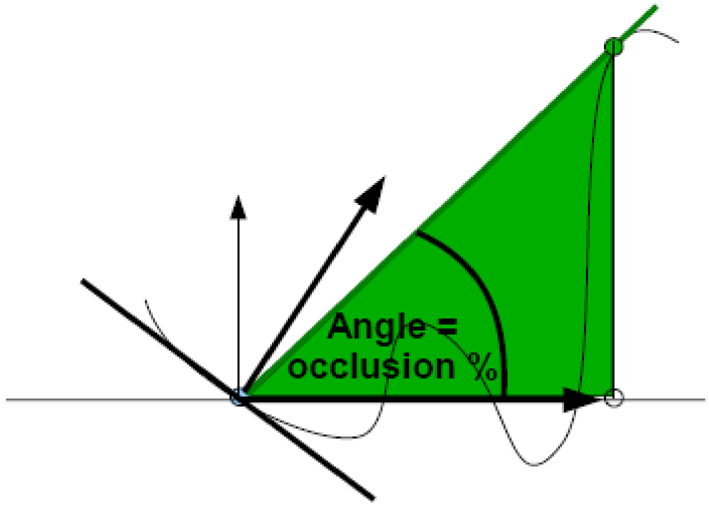
Horizon-based SSAO

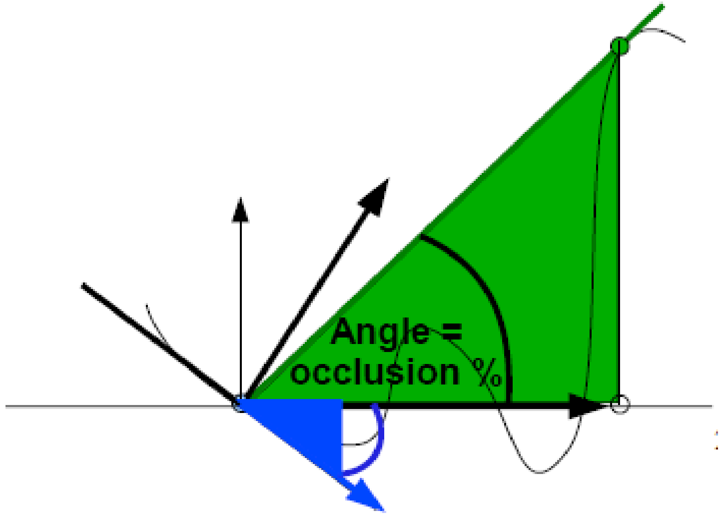
- Occlusion in height field
- Sampling height field along ray



Horizon-based SSAO

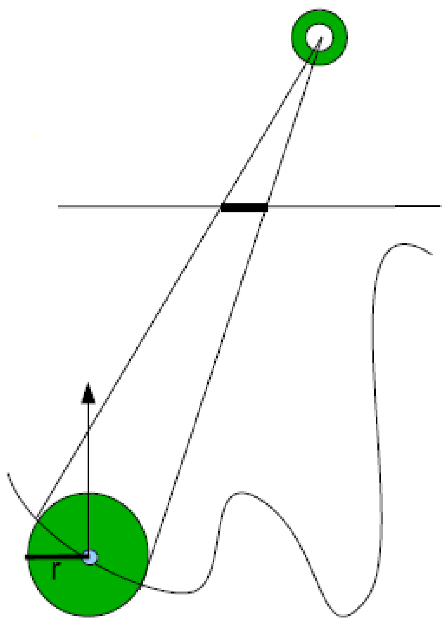
- Getting horizon angle *h(Θ)* for direction *Θ*
- Given normal in P -> tangent vector in P -> signed tangent angle *t(Θ)*
- $AO = \sin(h(\Theta)) - \sin(t(\Theta))$

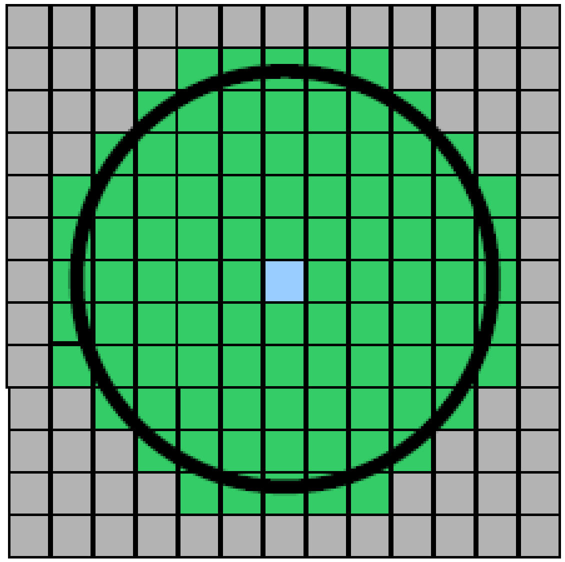




SSAO parameters

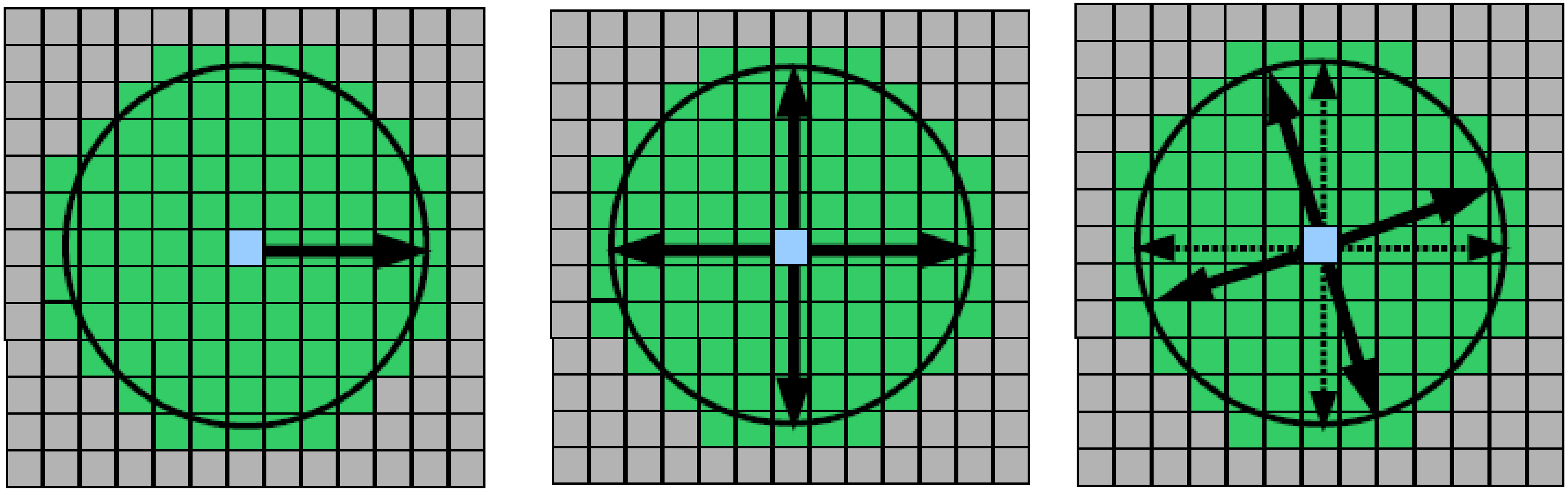
- Number of samples per ray = area of interest
- Radius r is constant and defined in eye space
- Calculate projection of sphere in screen space





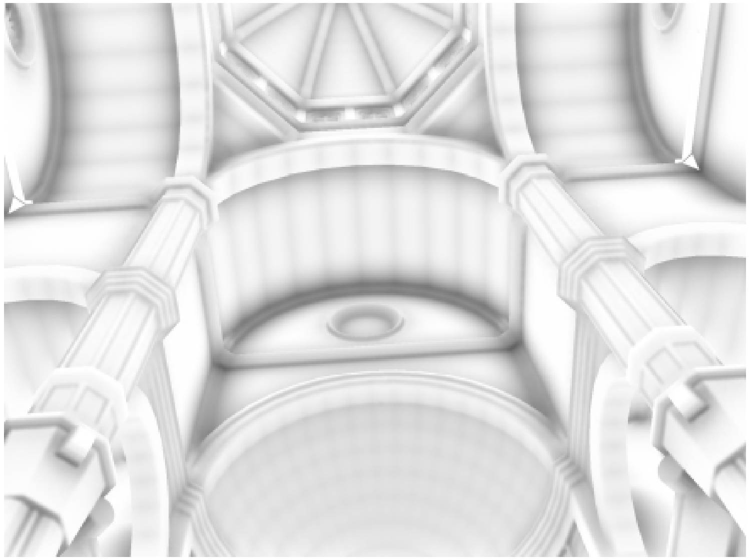
SSAO parameters

- Sampling rays – user defined number
- Random rotation of rays
- Jitter samples along ray

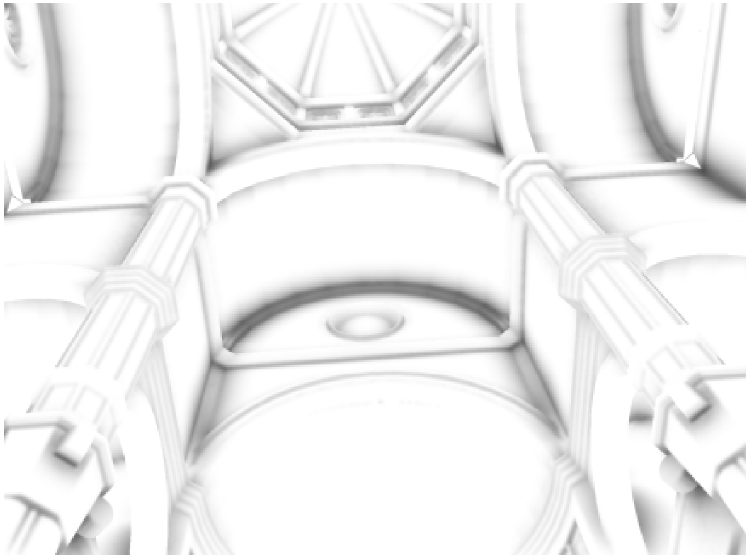


SSAO angle bias

- Ignore occlusion near the tangent plane
- Remove low tesselation artifacts
- Horizon angle is at least some value



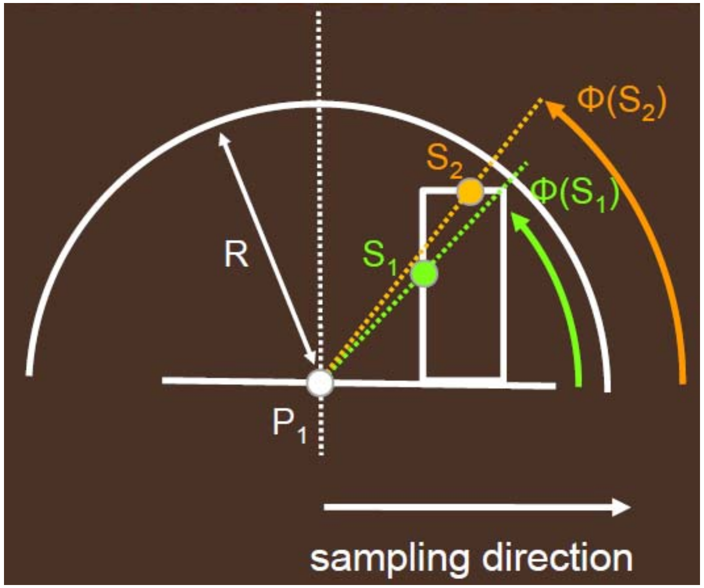
No angle bias



With 30 deg angle bias

Distance attenuation

- Solving large differences of SSAO values for neighboring pixels
- Using weights for each sample, $W(r) = 1\text{-}r^2$, …
- Cumulating AO while sampling along ray
  - Initialize WAO = 0
  - After sample S_1
    - $AO(S_1) = \sin(\Phi(S_1)) - \sin t$
    - $WAO \mathrel{+}= W(S_1)\, AO(S_1)$
  - After sample S_2
    - If $\Phi(S_2) > \Phi(S_1)$
    - $AO(S_2) = \sin(\Phi(S_2)) - \sin t$
    - $WAO \mathrel{+}= W(S_2)(AO(S_2)\text{-}AO(S_1))$



Distance attenuation

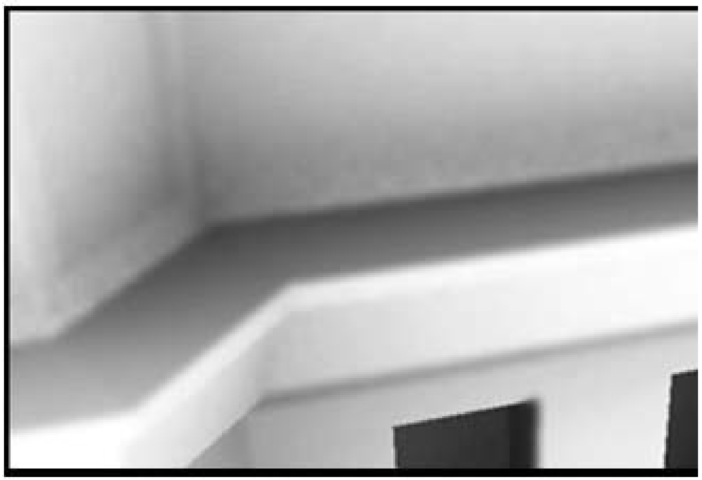
No attenuation

With attenuation, $W(r)=1-r^2$

Noise reduction

- Sampling only few values -> noise, alias
  - Process downscaled depth/normal buffers
  - Blur AO values (remove high frequencies), use depth dependent Gaussian blur

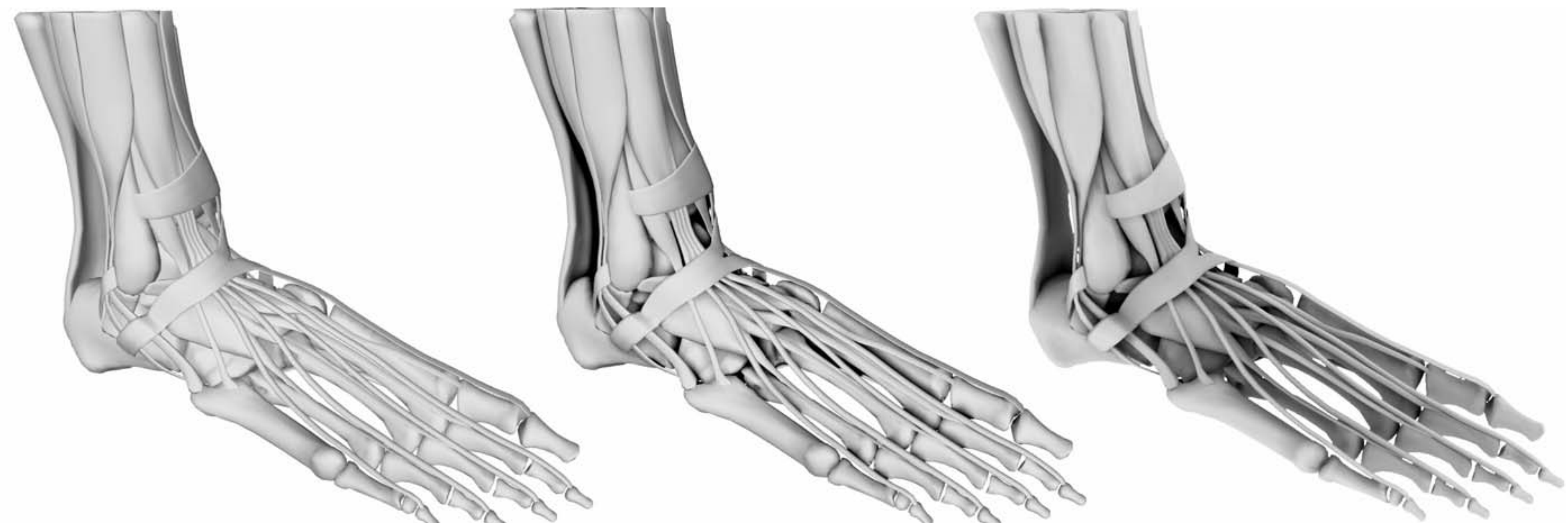
Without Blur



With 15x15 Blur

Depth buffer masking

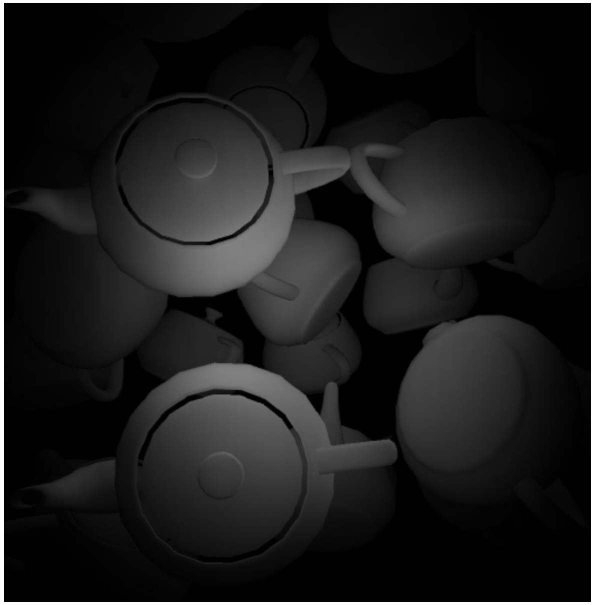
- Depth buffer is blurred, then subtracted from the original depth buffer
- Difference is added to color channels



Original rendering　　Depth darkening　　Ambient occlusion

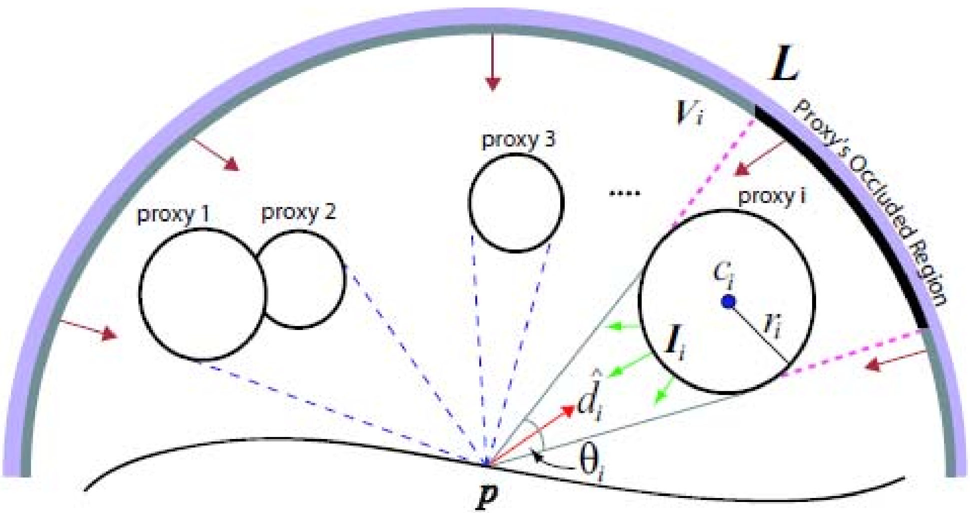
Real-time GI

- Simulating indirect lighting with high number of small direct lights
- Using deferred rendering



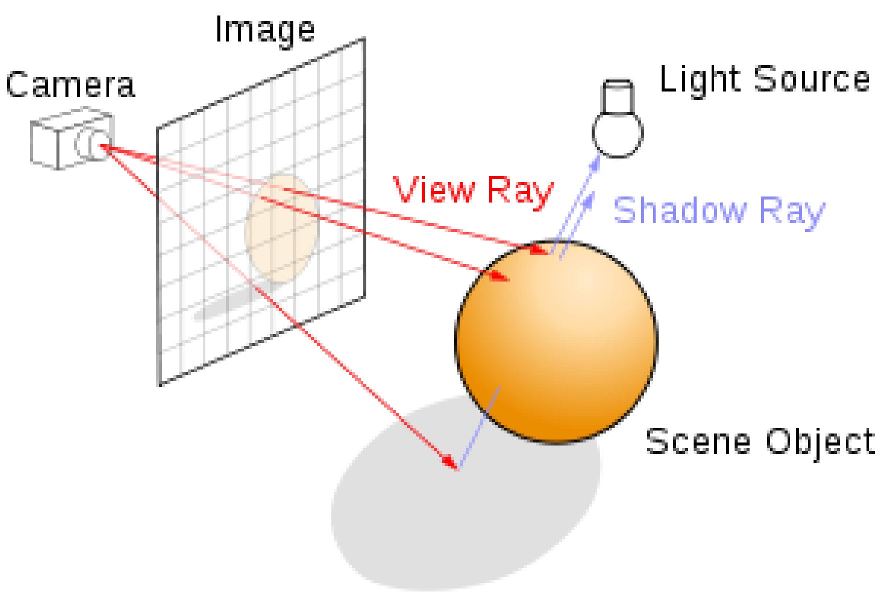
Real-time GI

- Sloan at al.
- Using spherical proxies – simplification of dynamical geometry



Ray tracing

- Backtracking ray that comes to eye
- On surface – multiple bounces – Monte Carlo



Ray tracing

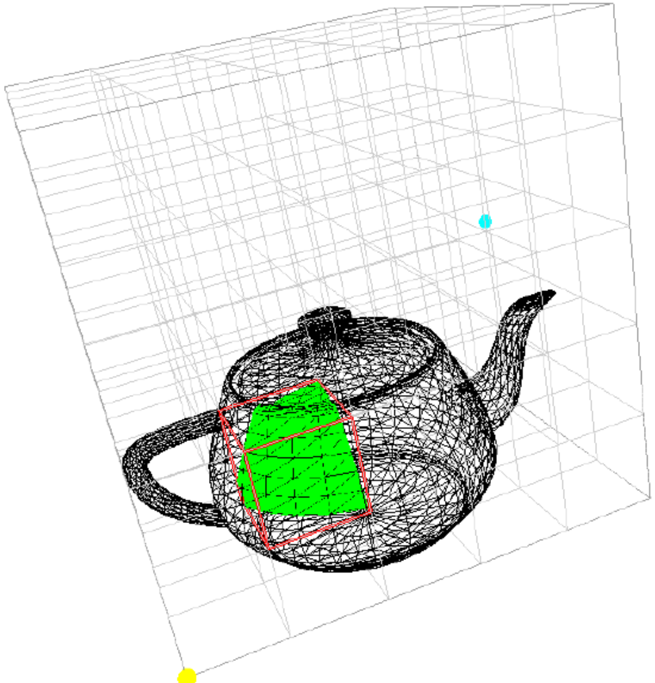
- Crucial – intersection of ray and object in scene
- Intersection speed up
  - Data structures – BVH trees, uniform grids, kD trees, octrees, …
  - Efficient algorithms
- Local illumination in intersection
- Handle absorption, reflection, refraction and fluorescence in intersection

Ray tracing

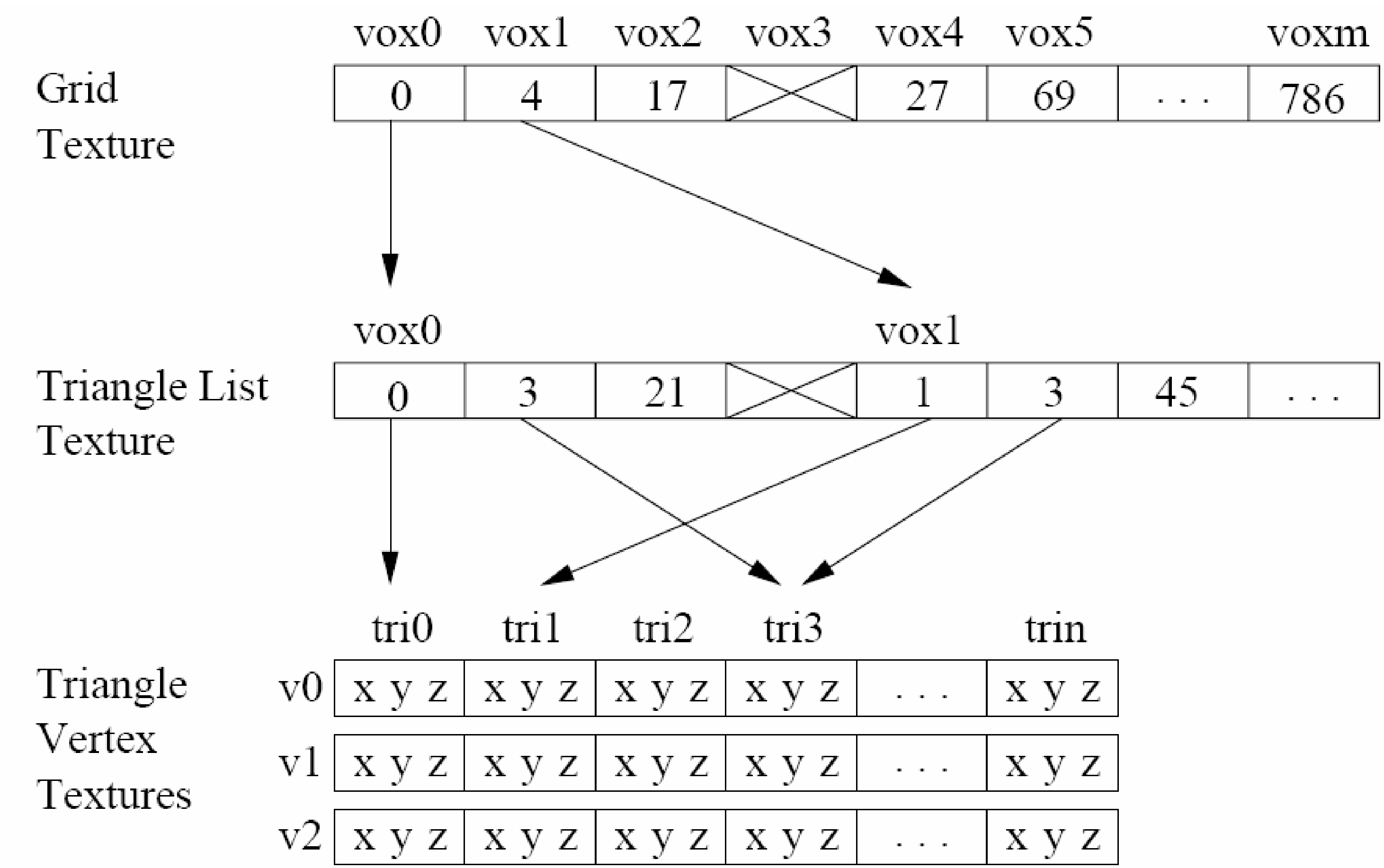
- Traverse acceleration structure
  - Cull away parts that ray cannot hit
  - Leaf nodes contain primitives
- Primitive intersection
  - Intersect ray directly
  - Return hit status to traversal
- Generate secondary rays from hit

Uniform grid

- GPU friendly → 3D texture
- Dependent fetches for lookup
- Each voxel → several primitives (parts)
- Precomputed on CPU
- Static scene
- Resolution?



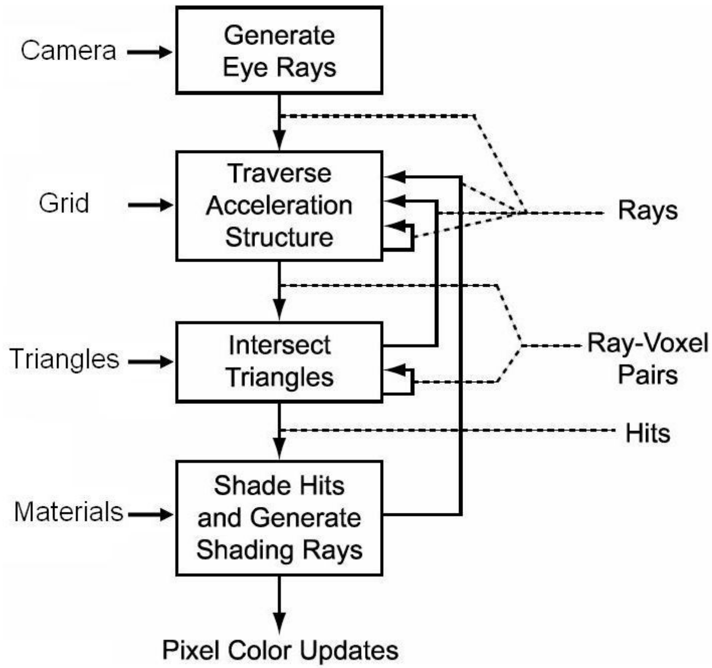
Uniform grid - GPU

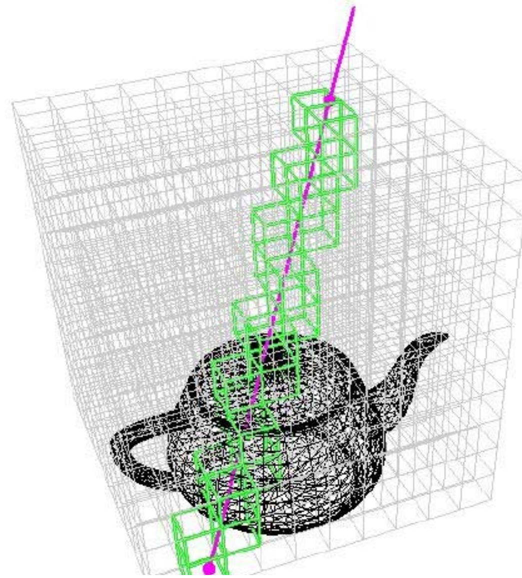


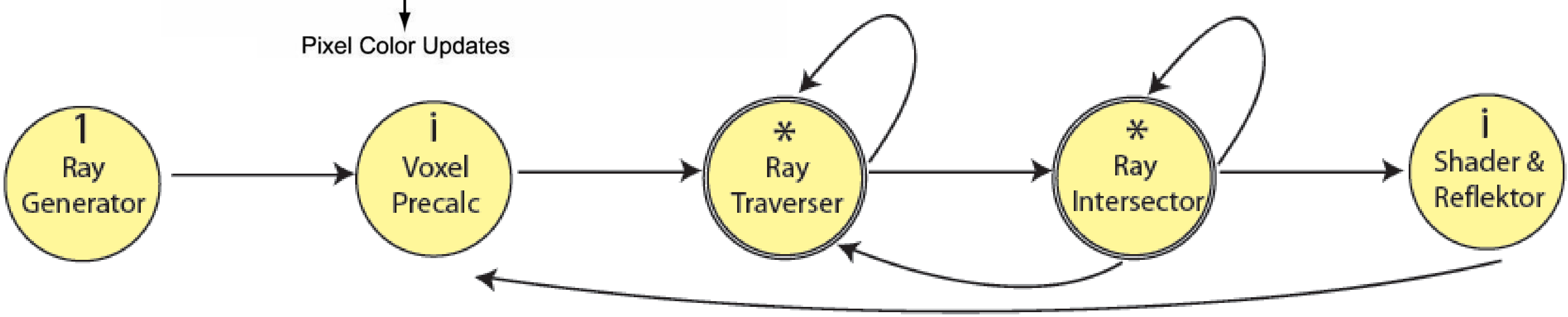
GPU ray tracing

- Rendering quads with screen size
- Using fragment shaders for computation
- Storing data in textures
- Textures for rays, intersections,
- Using 3D DDA for uniform grid traversal
- Computing exact intersection of ray and triangle
- http://www.clockworkcoders.com/oglsl/rt/

GPU ray tacing

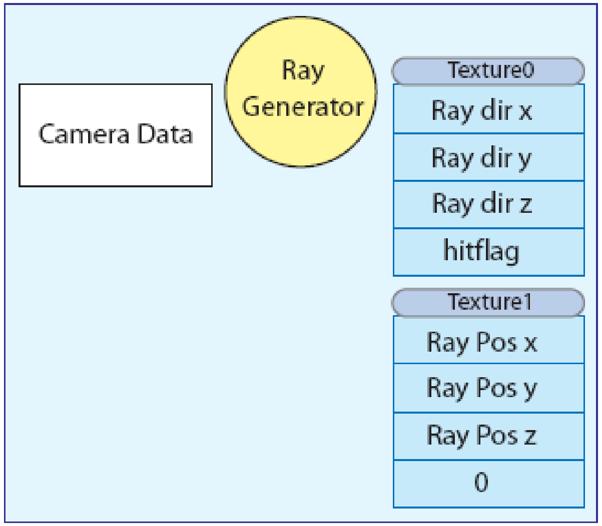






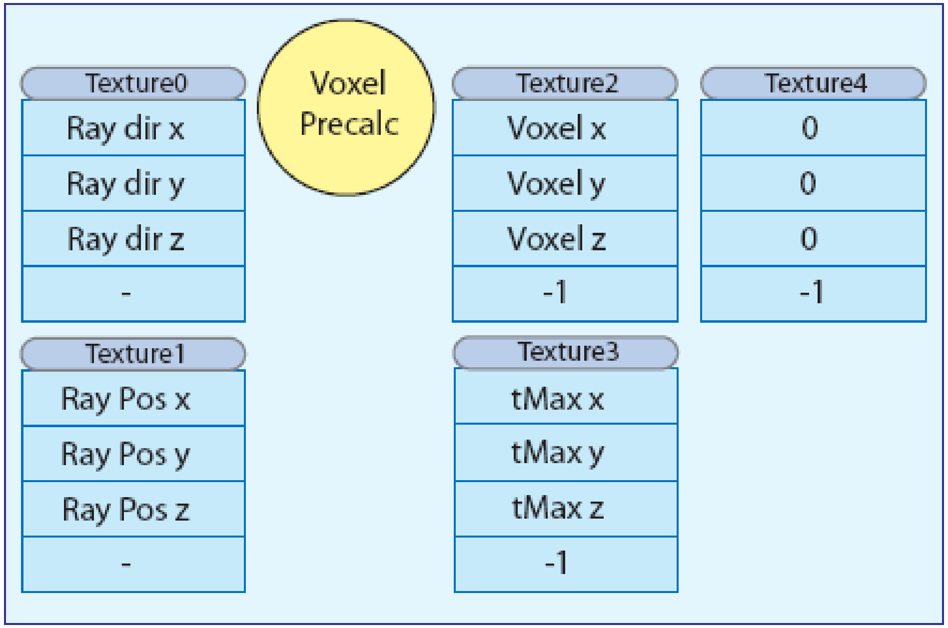
Ray generation

- Generation of ray parameters for each pixel
- In – camera data
- Out – ray starting point and direction, flag if ray hit bounding box of scene



Voxel Precalc

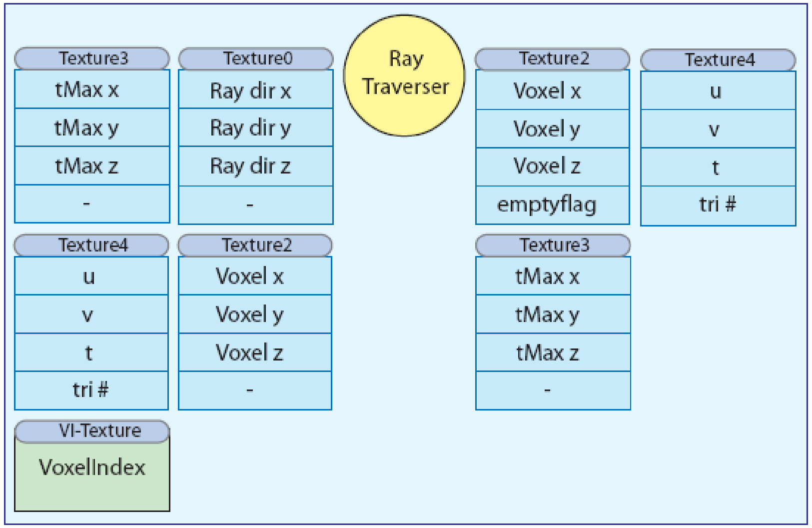
- First voxel of grid along ray
- In – ray vector, position (world coord)
- Out – in/out position (grid coord)



Ray Traverser

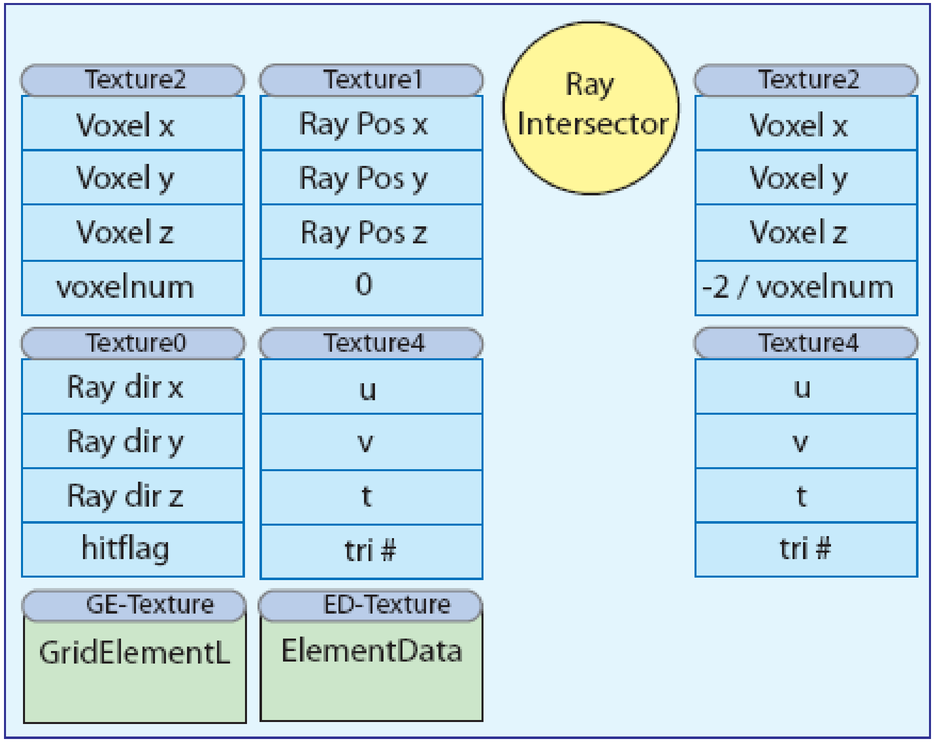
- Traversing grid along ray, setting state of ray based on voxel position and voxel triangles
- In – current voxel
- Out – next voxel

active	traverse Grid
wait	ready to check intersections
dead	ray doesn't hit grid (was already rejected in voxel precalculation)
inactive	a valid hit point was found
overflow	traversal left voxel space (no valid hits)

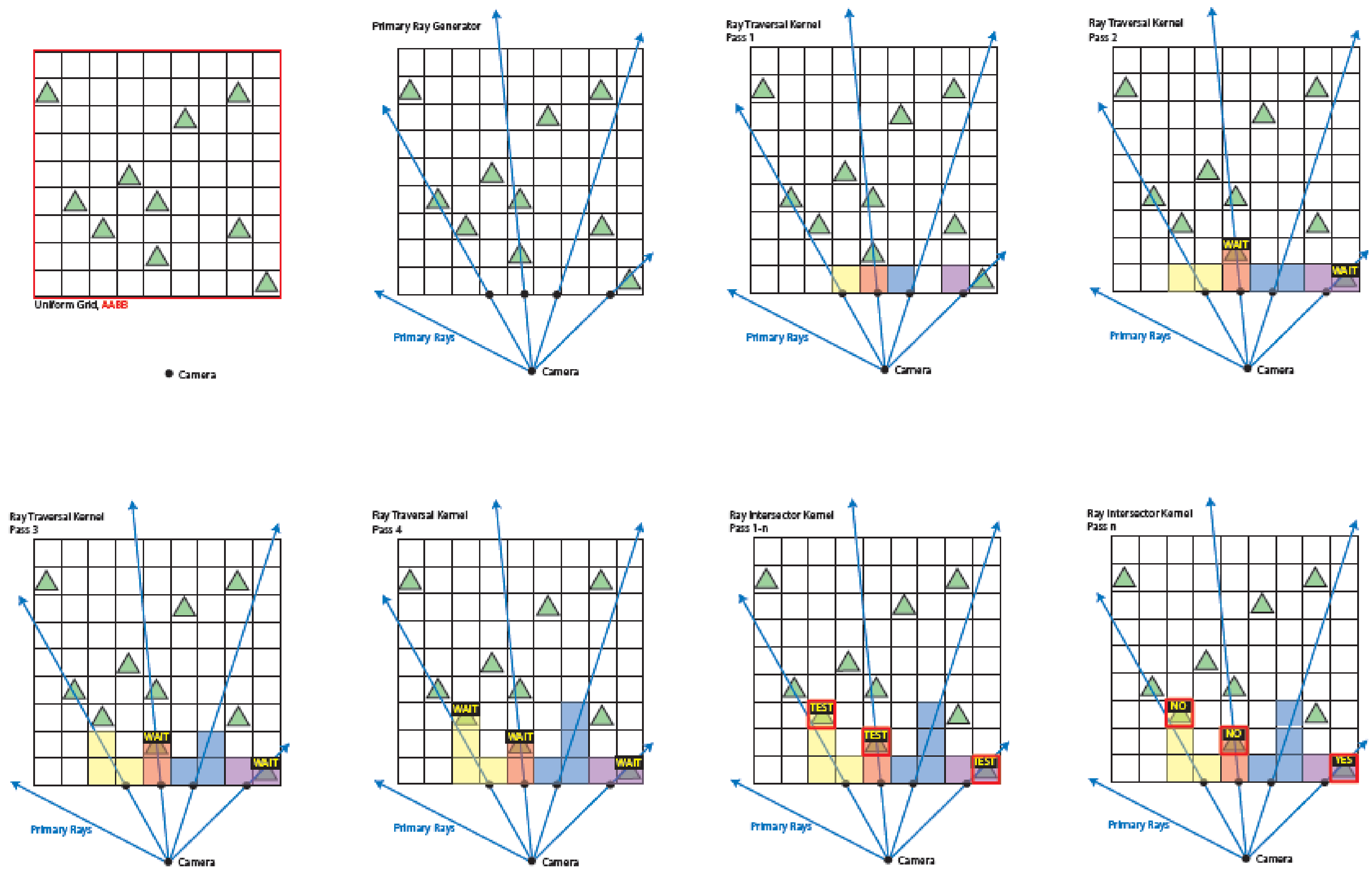


Ray Intersector

- For wait-state rays, computing intersection of ray and triangles in current ray's voxel
- In – current voxel
- Out – intersection



Loop



Uniform Grid, AABB
Camera
Primary Ray Generator
Primary Rays
Camera
Ray Traversal Kernel
Pass 1
Primary Rays
Camera
Ray Traversal Kernel
Pass 2
WAIT
WAIT
Primary Rays
Camera
Ray Traversal Kernel
Pass 3
WAIT
WAIT
Primary Rays
Camera
Ray Traversal Kernel
Pass 4
WAIT
WAIT
WAIT
Primary Rays
Camera
Ray Intersector Kernel
Pass 1-n
TEST
TEST
TEST
Primary Rays
Camera
Ray Intersector Kernel
Pass n
NO
NO
YES
Primary Rays
Camera

GPU ray tracing

Other GI sources

- Using GPGPU capabilities, CUDA, OpenCL
- Path tracing
  - http://igad.nhtv.nl/~bikker/ (CPU)
- Ray tracing:
  - http://www.nvidia.co.uk/object/optix_uk.html
  - http://graphics.stanford.edu/papers/i3dkdtree
  - http://graphics.cs.uiuc.edu/geomrt/
  - http://www.mpi-inf.mpg.de/~guenther/BVHonGPU/index.html

Questions?