Computer Graphics I

Lecture 9: Ray tracing basics

Xiaopei LIU

School of Information Science and Technology
ShanghaiTech University

Projection-based graphics

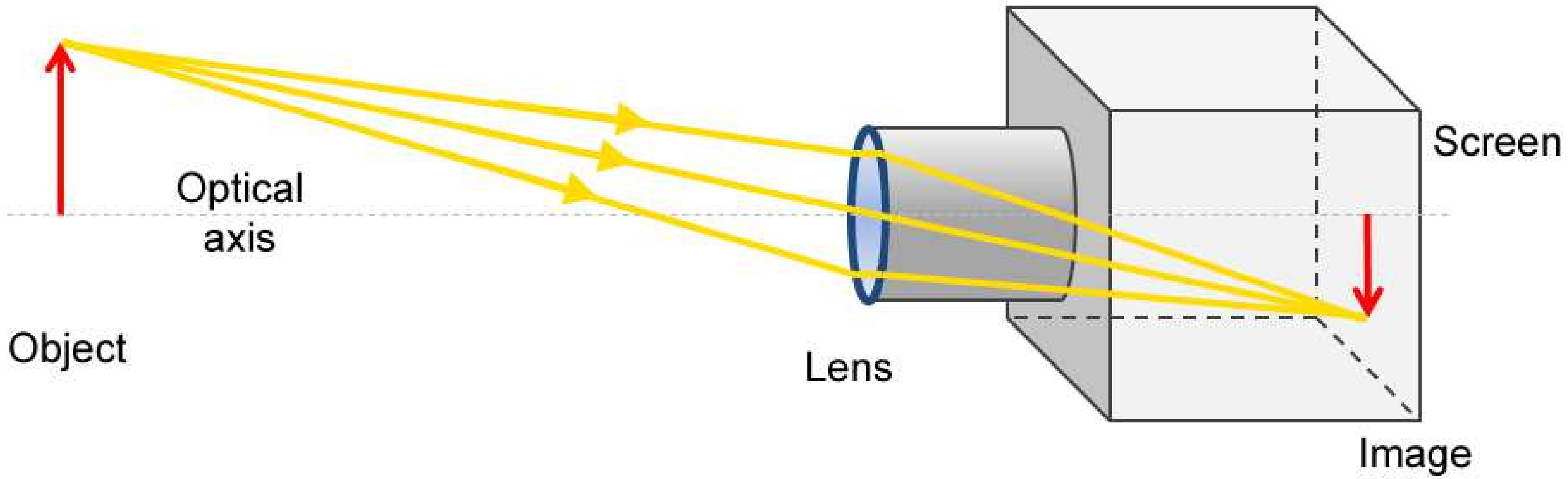
- **Start from geometric points**
  - Project to 2D imaging plane
  - Shade the 2D primitives based on lighting, texture…
  - Efficient on the GPU
- **What's the problem for projection-based graphics?**
  - Difficult to have realistic lighting (distribution of light)
  - Lighting is one of the most important factors human can differentiate a real and a fake image

Purpose of ray tracing

- **To generate photo-realistic images**
  - Geometrical optics involved
  - More sophisticated method for lighting calculation

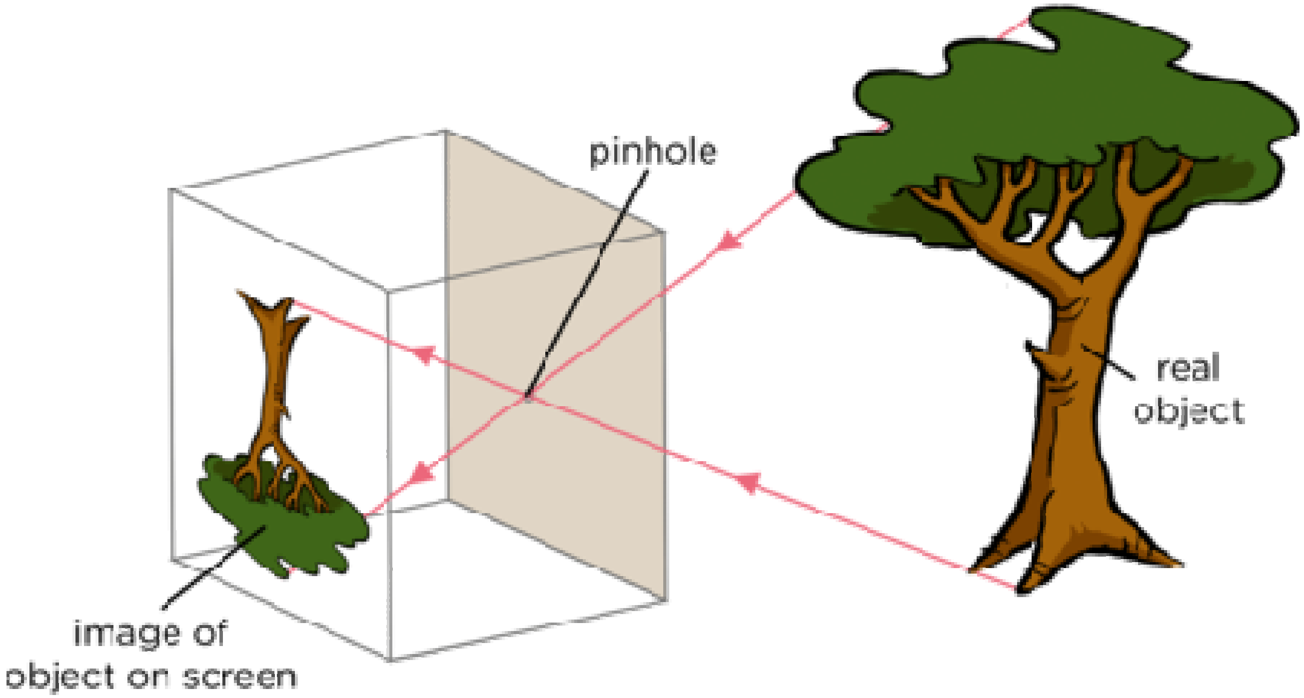
Producing photorealistic images

- **Recall our real camera system**



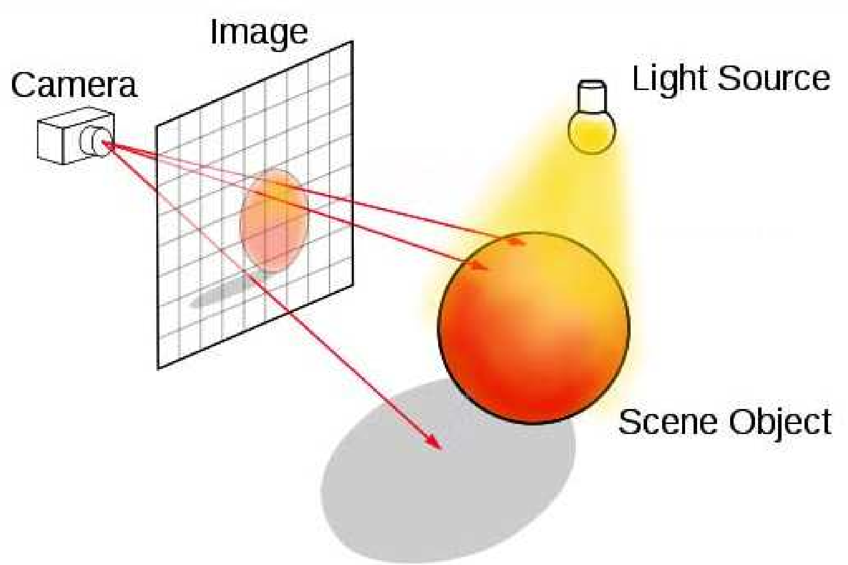
Producing photorealistic images

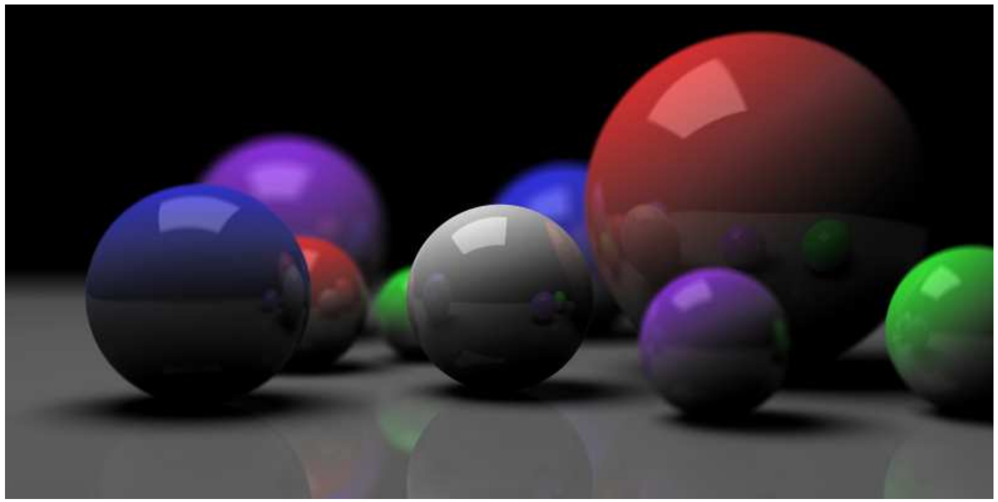
- **Camera model**
  - Pin-hole camera model



Producing photorealistic images

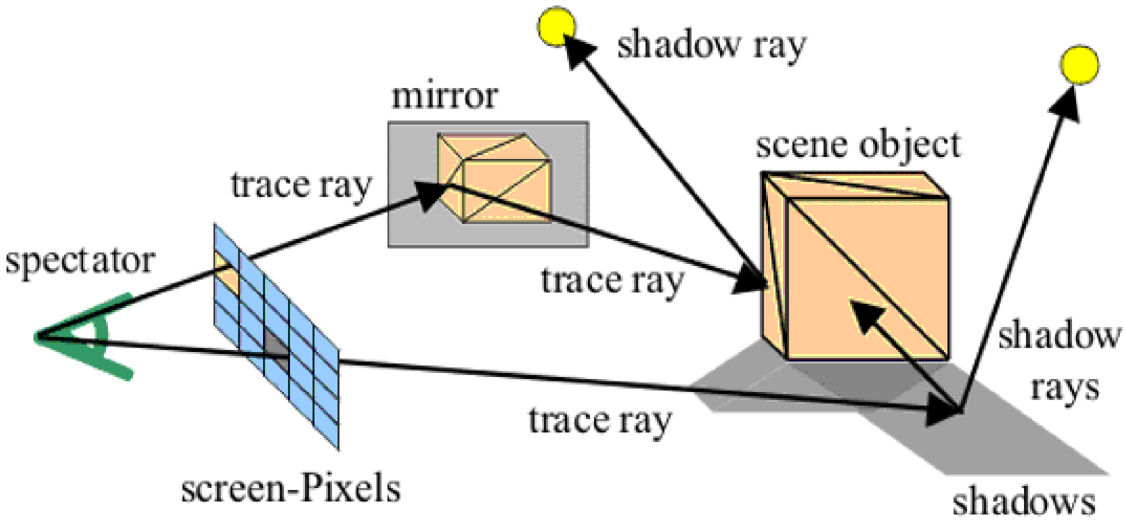
- **Ray tracing using pin-hole camera**
  - Light rays are reversible
  - Camera rays shooting from the center of the pixel





Ray tracing

- **What shall we need for ray tracing?**
  - A set of rays shooting from imaging plane
  - Light source distribution
  - Ray-object intersection
    - Normal, texture coordinates
    - Reflected and refracted rays
  - How the object reflects light



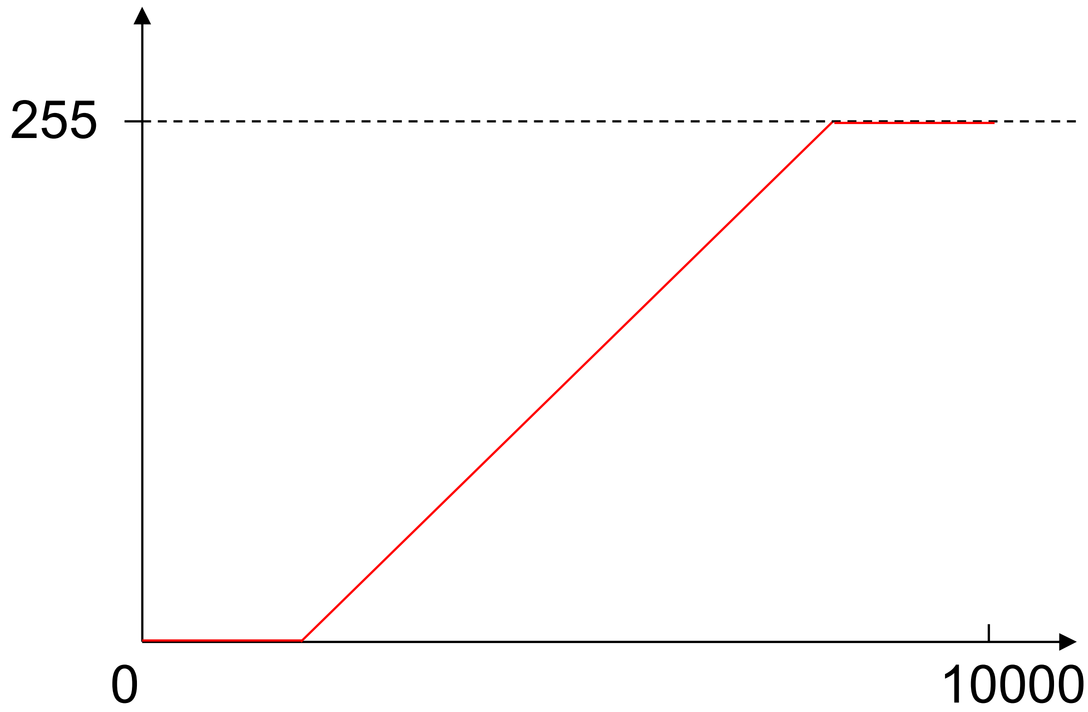
HDR in ray tracing

- **We need HDR intensity/color representation in ray tracing, why?**
  - The natural image radiance range is large
  - A natural way for image representation

- **Similarity with camera imaging process**
  - In camera system, incoming light radiance range is large
  - Image is formed under a non-linear process
  - Exposure (intensity scaling), clamping

HDR in ray tracing

- **How to obtain the final image from HDR representation?**
  - Select a suitable range in HDR radiance and map to a LDR
  - Or tone-mapping

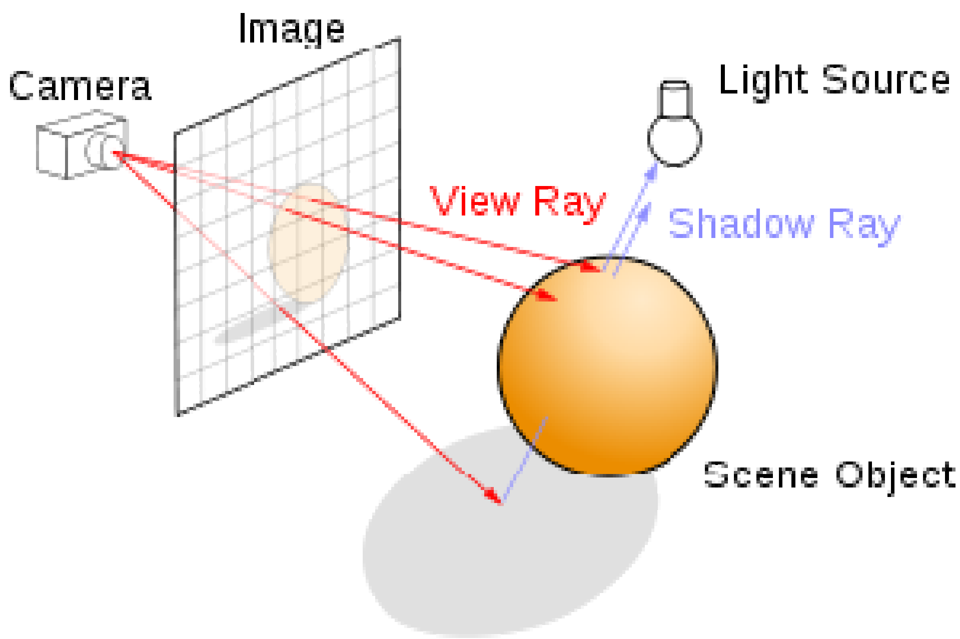


Optical ray

- **Shooting optical rays from focal point and through each pixel in imaging plane**
  - Rays are generated by connecting focal point and image pixel center

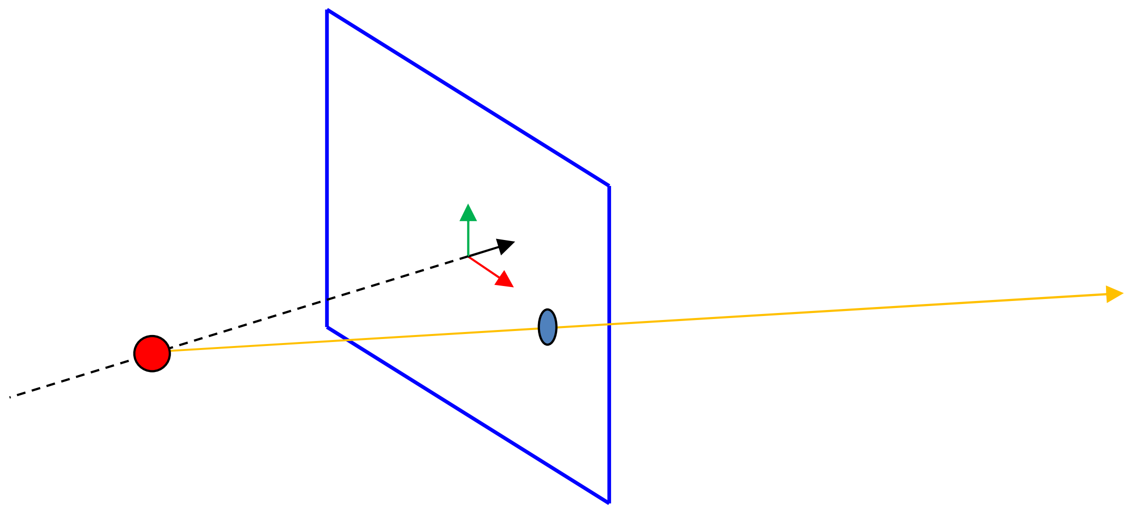
- **Ray expression**

$$r(t) = o + t\mathbf{d} \qquad 0 \leq t \leq \infty$$



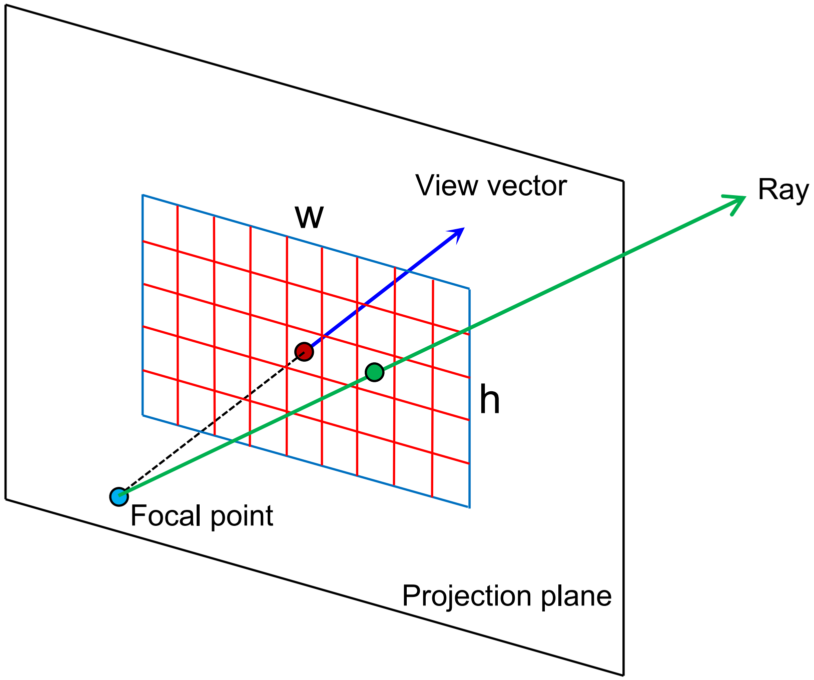
How camera is constructed

- **Virtual camera**
  - A focal point (world coordinate) + focal length
  - An imaging plane (world coordinate)
  - Image resolution
    - Thus we have the pixel center coordinate in world space



Building virtual camera system

- **Building camera in a global coordinate system**
  - Camera center
  - Viewing direction
  - Projection plane
  - Focal length -> focal point
  - Determine viewing range
    - World coordinate unit
  - Sample view range (resX x resY)
  - Shoot rays



1. Ray-geometry intersection

Ray-geometry intersection

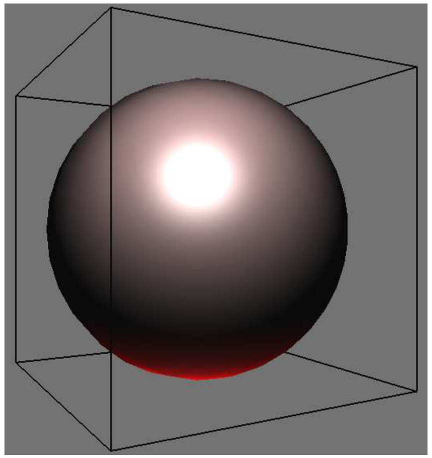
- **Ray-geometry intersection is an important step in ray-tracing**
  - Determine the intersection point
  - Normal, texture coordinate etc. at that point

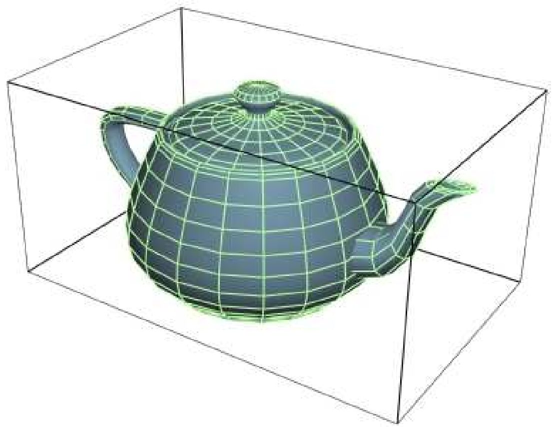
- **Such a process can be recursively called**
  - Reflection/refraction
  - Ray distribution

Bounding box

- **Minimum bounding box**
  - The box with the smallest measure (area, volume etc.) within which the object lies

- **Axis-aligned minimum bounding box (AABB)**
  - minimum bounding box with the constraint that the edges of the box are parallel to the coordinate axes





Ray equation

- **Can express ray as**

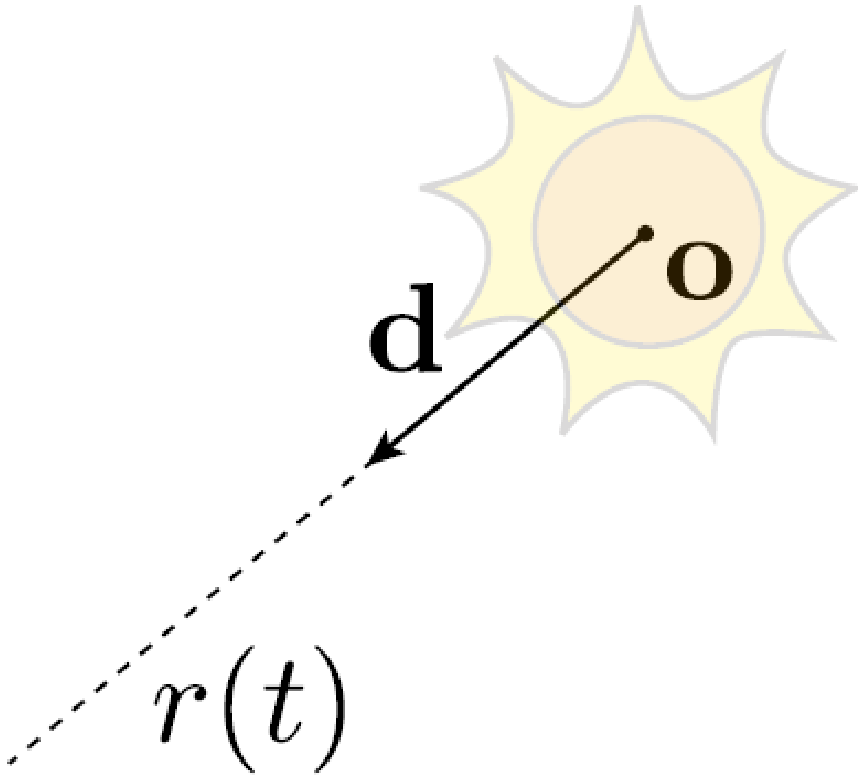
origin

unit direction

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

point along ray

"time"



Intersecting a ray with an implicit surface

- Recall implicit surfaces: all points x such that f(x) = 0
- Q: How do we find points where a ray pierces this surface?
- Well, we know all points along the ray: r(t) = o + td
- Idea: replace "x" with "r" in 1st equation, and solve for t
- Example: unit sphere

$$f(\mathbf{x}) = |\mathbf{x}|^2 - 1$$

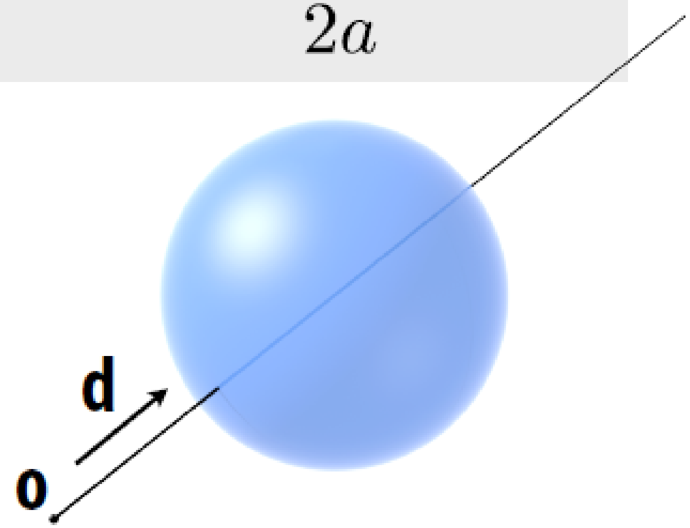
$$\Rightarrow f(\mathbf{r}(t)) = |\mathbf{o} + t\mathbf{d}|^2 - 1$$

$$\underbrace{|\mathbf{d}|^2}_{a} t^2 + \underbrace{2(\mathbf{o} \cdot \mathbf{d})}_{b} t + \underbrace{|\mathbf{o}|^2 - 1}_{c} = 0$$

$$t = \boxed{-\mathbf{o} \cdot \mathbf{d} \pm \sqrt{(\mathbf{o} \cdot \mathbf{d})^2 - |\mathbf{o}|^2 + 1}}$$

quadratic formula:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



Why two solutions?

Ray-sphere intersection

- **Sphere equation**
  - With center located at (0,0,0)

$$x^2 + y^2 + z^2 - r^2 = 0$$

- **Substitute ray equation**
  - Ray in parametric form

$$(\mathrm{o}_x + t\mathbf{d}_x)^2 + (\mathrm{o}_y + t\mathbf{d}_y)^2 + (\mathrm{o}_z + t\mathbf{d}_z)^2 = r^2$$

Ray-sphere intersection

- **A general quadratic equation in *t***

$$At^2 + Bt + C = 0$$

where

$$A = \mathbf{d}_x^2 + \mathbf{d}_y^2 + \mathbf{d}_z^2$$

$$B = 2(\mathbf{d}_x \mathrm{o}_x + \mathbf{d}_y \mathrm{o}_y + \mathbf{d}_z \mathrm{o}_z)$$

$$C = \mathrm{o}_x^2 + \mathrm{o}_y^2 + \mathrm{o}_z^2 - r^2.$$

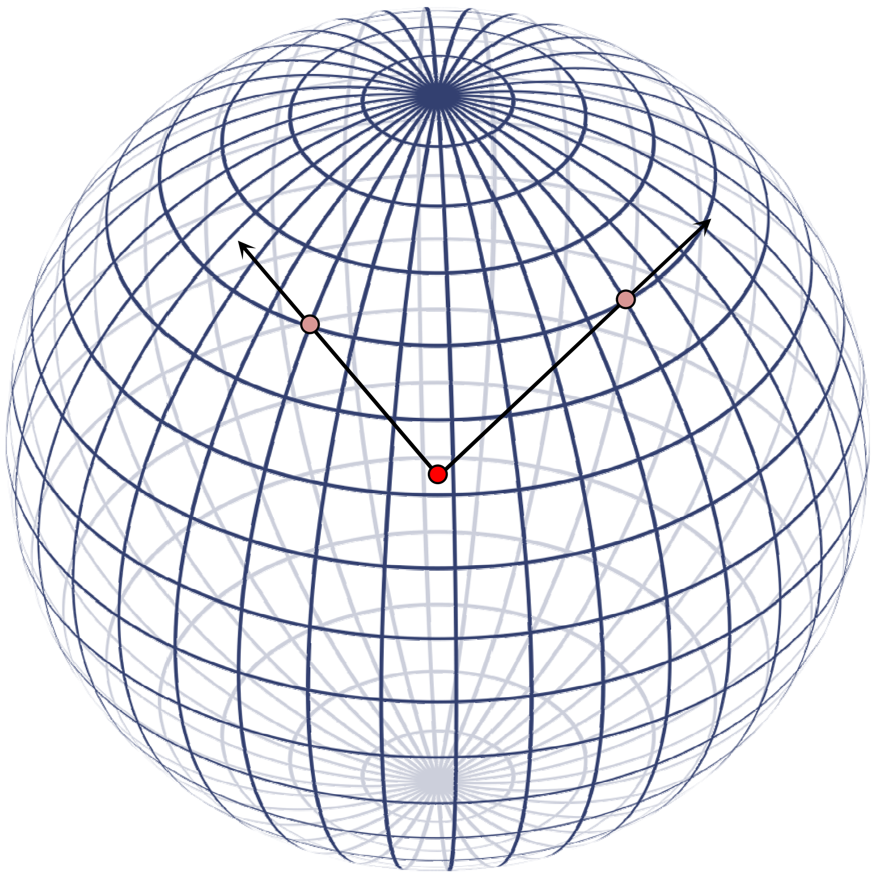
- **Solving for *t***

$$t_0 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

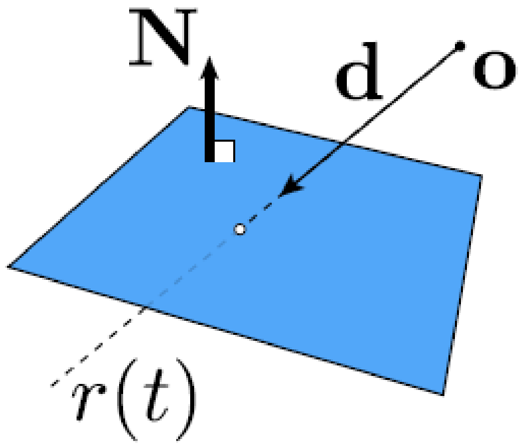
$$t_1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

Ray-sphere intersection

- **Normal at intersection**
  - A vector starting from the center to the intersection point



Ray-plane intersection



- Suppose we have a plane $N^T x = c$
  - N - unit normal
  - c - offset
- How do we find intersection with ray $r(t) = o + td$?
- *Key idea:* again, replace the point x with the ray equation t:

$$\mathbf{N}^\mathsf{T}\mathbf{r}(t) = c$$

- Now solve for t:

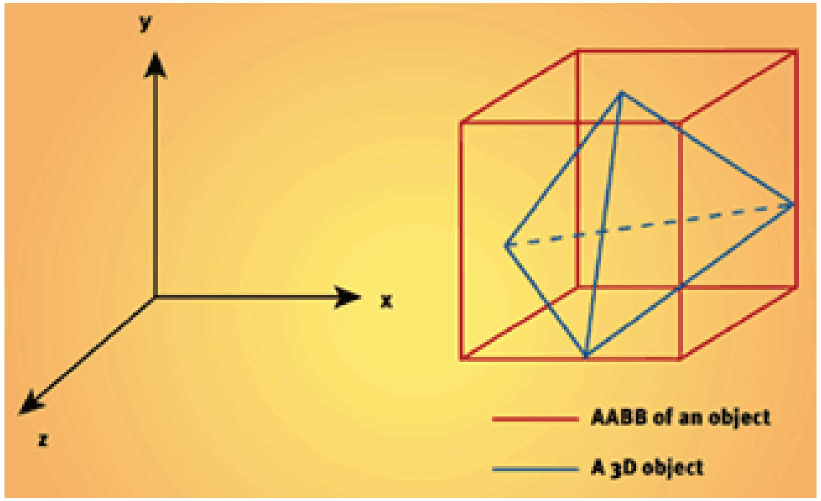
$$\mathbf{N}^\mathsf{T}(\mathbf{o} + t\mathbf{d}) = c \qquad \Rightarrow t = \frac{c - \mathbf{N}^\mathsf{T}\mathbf{o}}{\mathbf{N}^\mathsf{T}\mathbf{d}}$$

- And plug t back into ray equation:

$$r(t) = \mathbf{o} + \frac{c - \mathbf{N}^\mathsf{T}\mathbf{o}}{\mathbf{N}^\mathsf{T}\mathbf{d}}\mathbf{d}$$

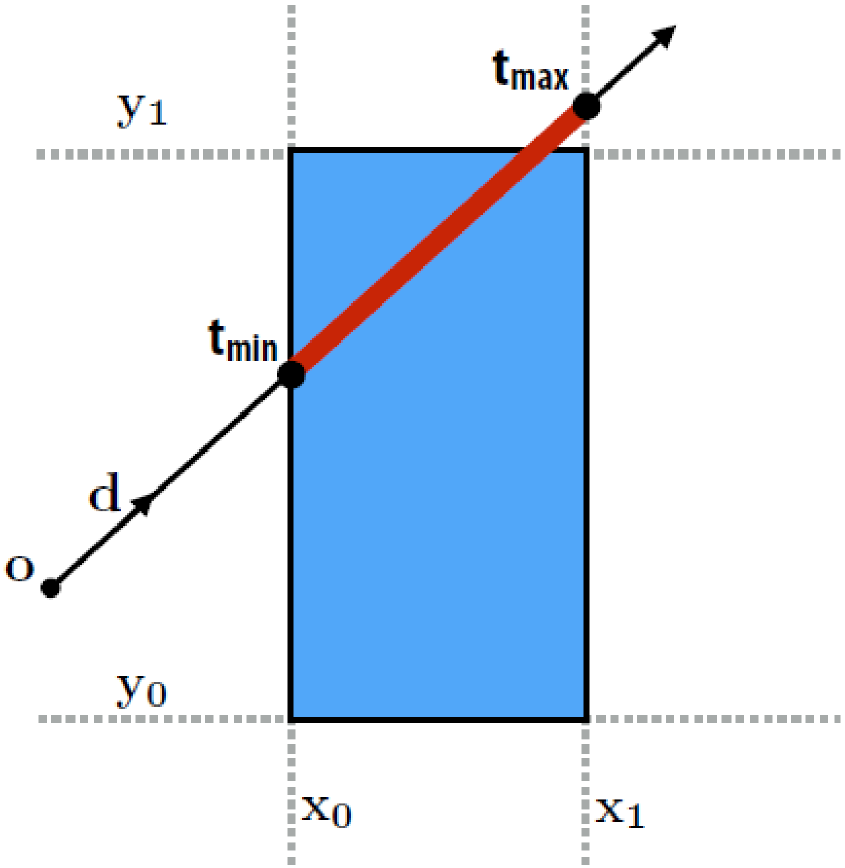
Ray-bounding-box intersection

- **How to intersect a ray with AABB?**
  - Intersection with planes and check intersection point range
- **Ray-plane intersection**
  - The AABB plane equations
    - Parallel to a coordinate plane
      - E.g., z=10;
  - Insert ray equation into the plane
    - Solve for parameter t
    - E.g., insert $o_z+td_z=10$ -> get x,y values
    - Check the ranges, e.g., check the x, y value for whether they are within the AABB range



Ray-bounding-box intersection

What is ray's closest/farthest intersection with axis-aligned box?



Find intersection of ray with all planes of box:

$$\mathbf{N}^{\mathbf{T}}(\mathbf{o} + t\mathbf{d}) = c$$

Math simplifies greatly since plane is axis aligned (consider $x = x_0$ plane in 2D):

$$\mathbf{N}^{\mathbf{T}} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$$

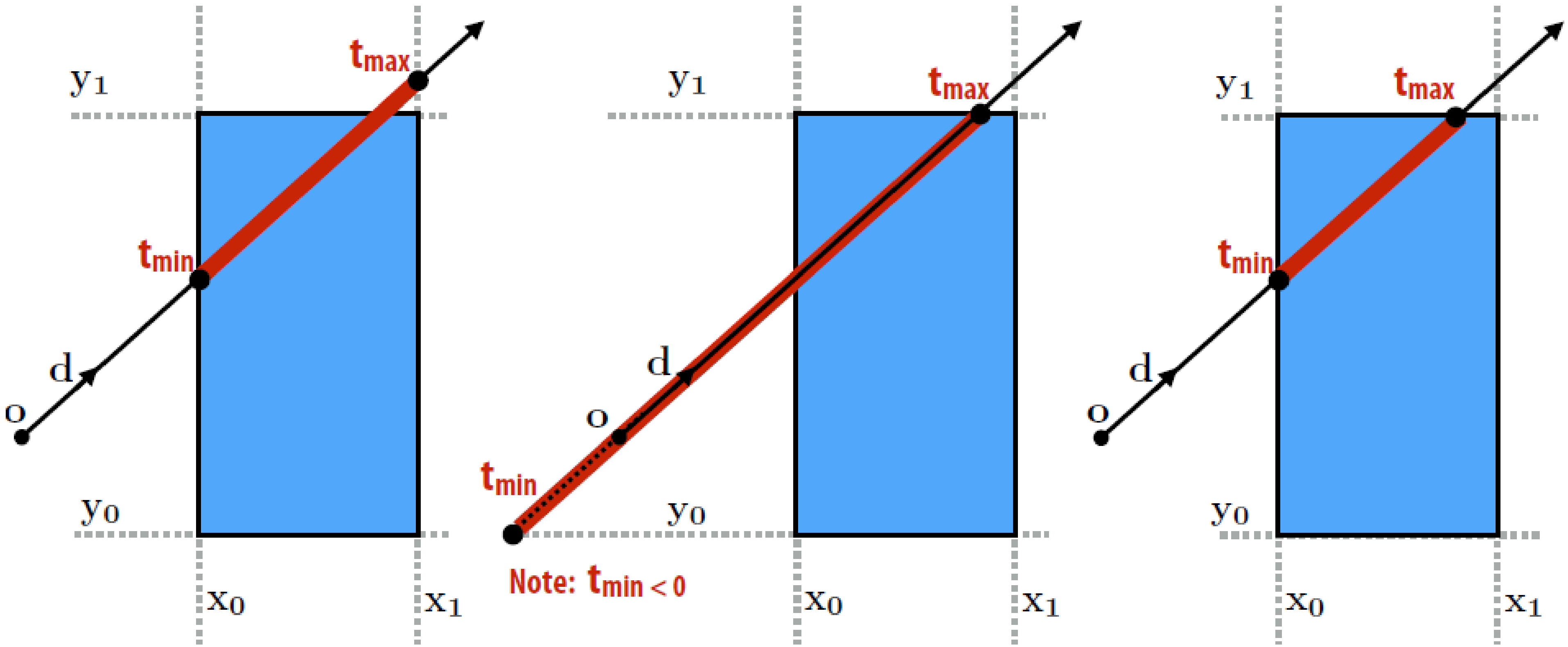
$$c = x_0$$

$$t = \frac{x_0 - \mathbf{o_x}}{\mathbf{d_x}}$$

Figure shows intersections with $x = x_0$ and $x = x_1$ planes.

Ray-bounding-box intersection

Compute intersections with all planes, take intersection of t_{min}/t_{max} intervals



Intersections with *x* planes | Intersections with *y* planes | Final intersection result

How do we know when the ray misses the box?

Ray-triangle intersection

- **Find ray-plane intersection**

  Parametric equation of a ray:

  $$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

  ray origin

  normalized ray direction

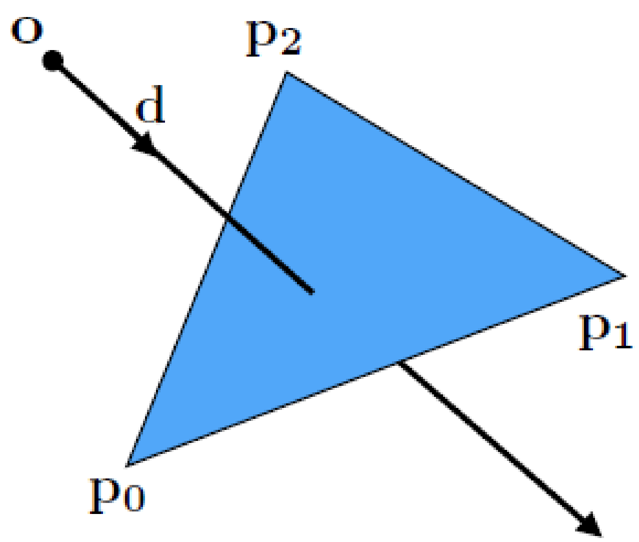
  Plug equation for ray into implicit plane equation:

  $$\mathbf{N}^{\mathbf{T}}\mathbf{x} = c$$

  $$\mathbf{N}^{\mathbf{T}}(\mathbf{o} + t\mathbf{d}) = c$$

  Solve for *t* corresponding to intersection point:

  $$t = \frac{c - \mathbf{N}^{\mathbf{T}}\mathbf{o}}{\mathbf{N}^{\mathbf{T}}\mathbf{d}}$$



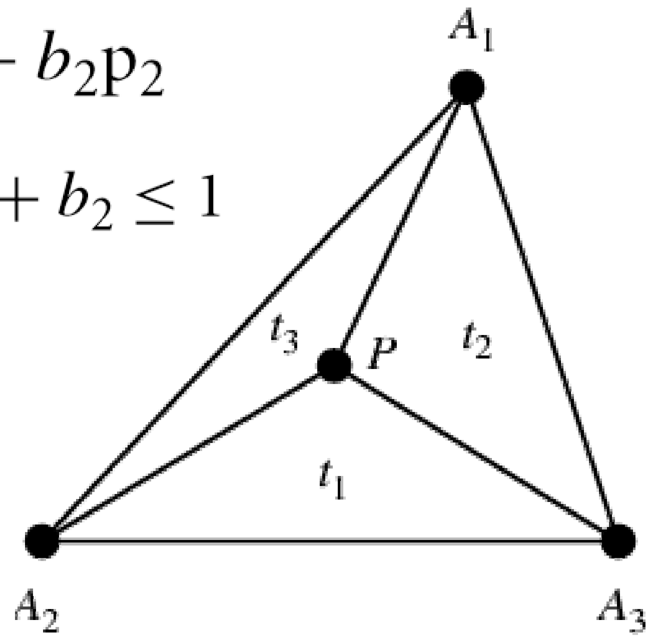
- **Determine if point of intersection is within triangle**

Ray-triangle intersection

- **An efficient ray-triangle intersection algorithm?**
  - Can also be derived using barycentric coordinates
- **For triangle**
  - Any point inside the triangle can be written as:

$$\mathrm{p}(b_1, b_2) = (1 - b_1 - b_2)\mathrm{p}_0 + b_1\mathrm{p}_1 + b_2\mathrm{p}_2$$

With conditions: $b_1 \geq 0, b_2 \geq 0$, $b_1 + b_2 \leq 1$



  - Insert parametric ray equation

$$\mathrm{o} + t\mathbf{d} = (1 - b_1 - b_2)\mathrm{p}_0 + b_1\mathrm{p}_1 + b_2\mathrm{p}_2$$

Ray-triangle intersection

- **Equation to solve**

$$\begin{pmatrix} -\mathbf{d} & \mathbf{e}_1 & \mathbf{e}_2 \end{pmatrix} \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \mathbf{s}$$

- **How to solve such an equation?**
  – Cramer's rule

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{|\,-\mathbf{d} \quad \mathbf{e}_1 \quad \mathbf{e}_2\,|} \begin{bmatrix} |\quad \mathbf{s} \quad \mathbf{e}_1 \quad \mathbf{e}_2| \\ |-\mathbf{d} \quad \mathbf{s} \quad \mathbf{e}_2| \\ |-\mathbf{d} \quad \mathbf{e}_1 \quad \mathbf{s}| \end{bmatrix}$$

we write $|\,\mathbf{a} \quad \mathbf{b} \quad \mathbf{c}\,|$ to mean the determinant

Ray-triangle intersection

- **Is this solver efficient?**
  - No!
- **More observation**
  - Determinant identification in 3D

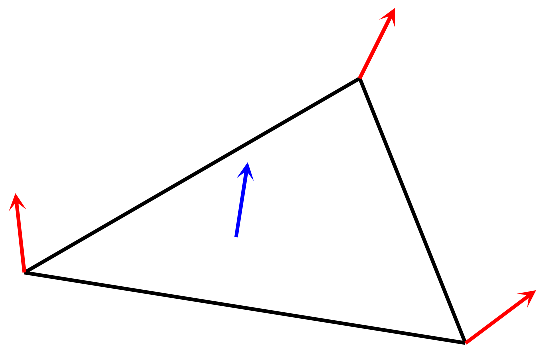
$$|\,\mathbf{a} \quad \mathbf{b} \quad \mathbf{c}\,| = -(\mathbf{a} \times \mathbf{c}) \cdot \mathbf{b} = -(\mathbf{c} \times \mathbf{b}) \cdot \mathbf{a}$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{(\mathbf{d} \times \mathbf{e}_2) \cdot \mathbf{e}_1} \begin{bmatrix} (\mathbf{s} \times \mathbf{e}_1) \cdot \mathbf{e}_2 \\ (\mathbf{d} \times \mathbf{e}_2) \cdot \mathbf{s} \\ (\mathbf{s} \times \mathbf{e}_1) \cdot \mathbf{d} \end{bmatrix} \Rightarrow \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\mathbf{s}_1 \cdot \mathbf{e}_1} \begin{bmatrix} \mathbf{s}_2 \cdot \mathbf{e}_2 \\ \mathbf{s}_1 \cdot \mathbf{s} \\ \mathbf{s}_2 \cdot \mathbf{d} \end{bmatrix}$$

Ray-triangle intersection

- **Internal normal interpolation from vertex normals**
  - Interpolation by barycentric coordinate computed previously when a ray intersects the triangle

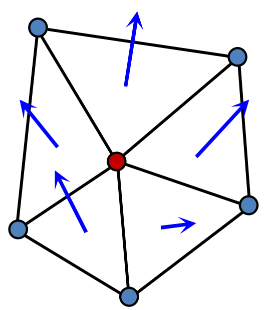
$$\mathbf{n}(b_1, b_2) = (1 - b_1 - b_2)\mathbf{n}_0 + b_1\mathbf{n}_1 + b_2\mathbf{n}_2$$



Ray-triangle intersection

- **If a mesh is formed with only triangles**
  - How can we determine normal at the intersection point?

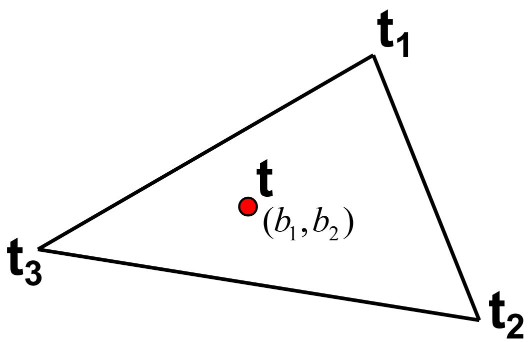
- **Vertex normal estimation**
  - Compute face normals
  - Compute vertex normal by averaging normals of the connected faces



Ray-triangle intersection

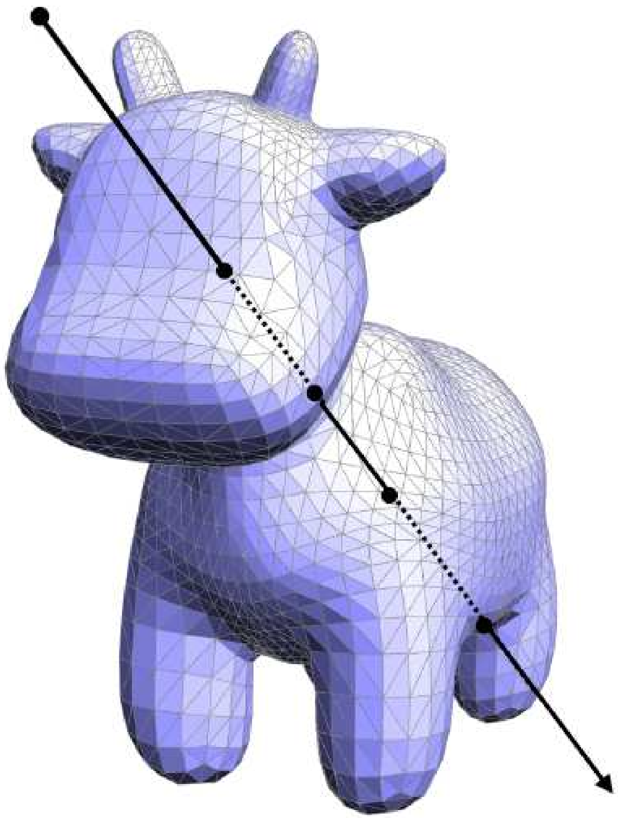
- **Internal texture coordinate interpolation from vertex texture coordinate**
  – Texture coordinate interpolation from triangle vertex texture coordinates by barycentric coordinate

$$\mathbf{t}(b_1, b_2) = (1 - b_1 - b_2)\mathbf{t}_0 + b_1\mathbf{t}_1 + b_2\mathbf{t}_2$$



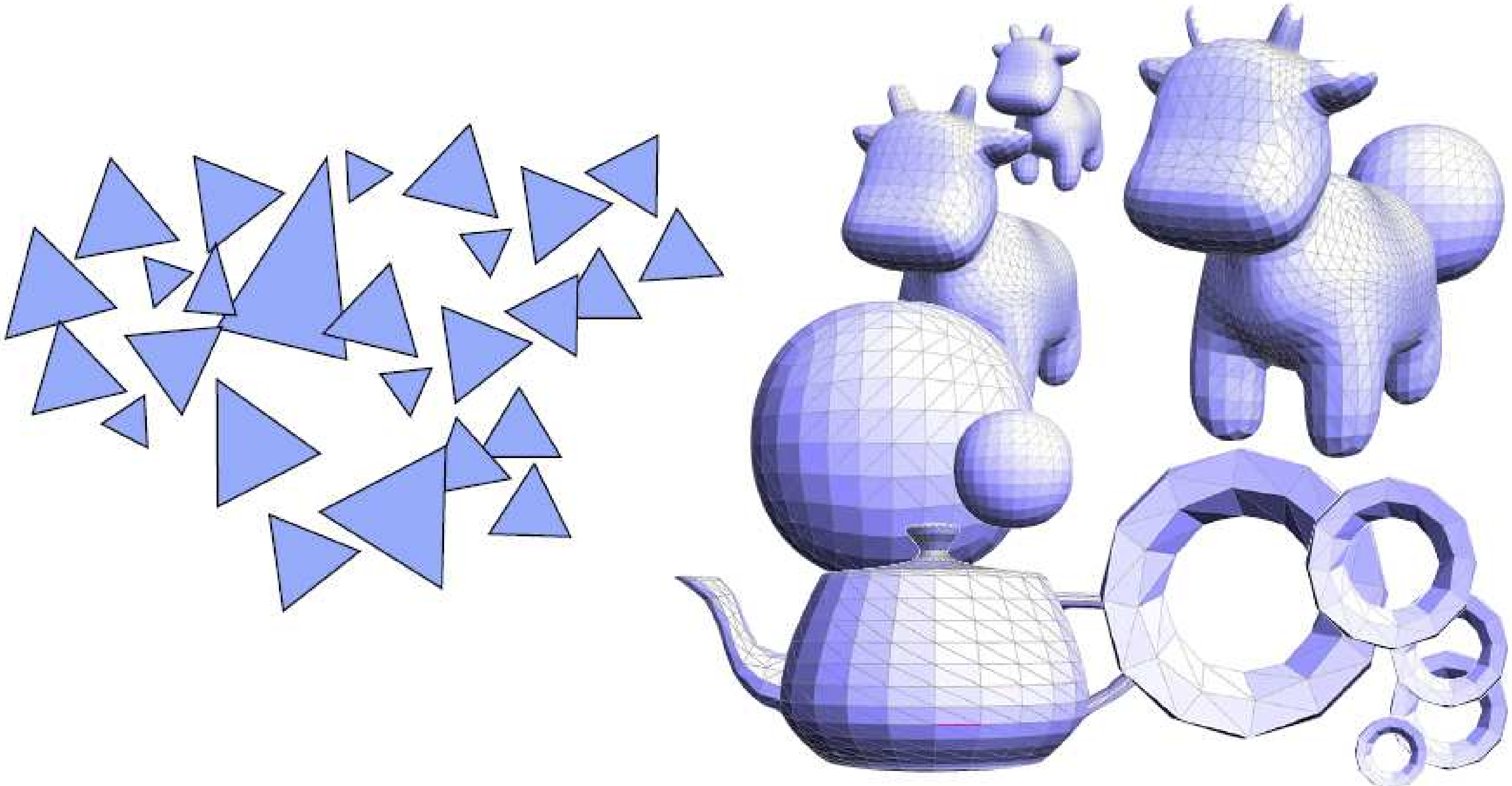
Ray-mesh intersection

- **How to intersect a mesh?**
  - Intersect its triangles
  - Search the triangles the ray hits
  - Obtain intersection point and normal



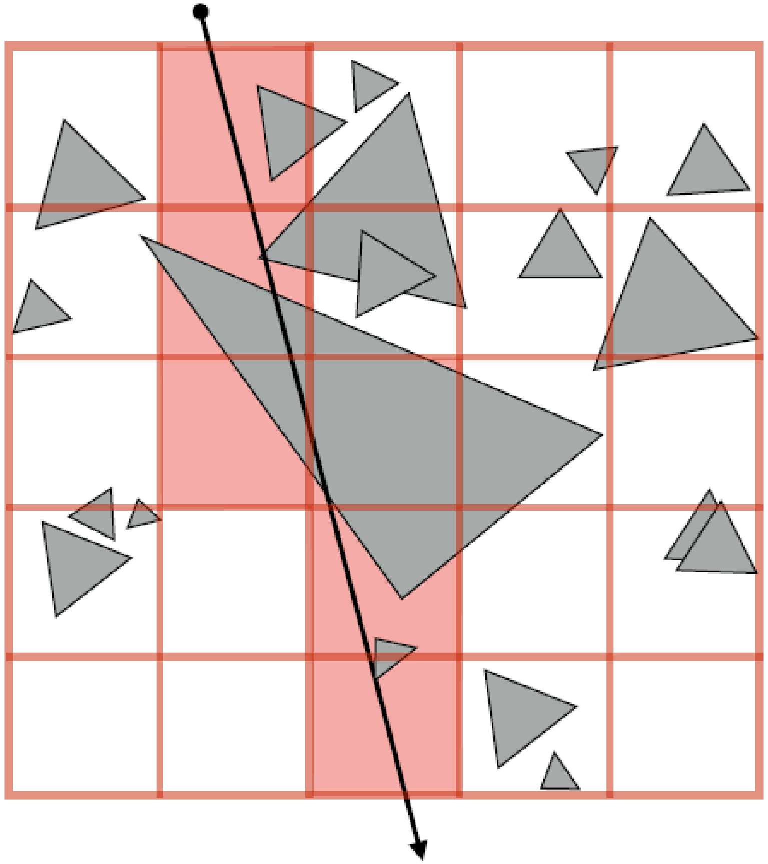
Ray-mesh intersection

How do we organize scene primitives to enable fast ray-scene intersection queries?



Ray-mesh intersection

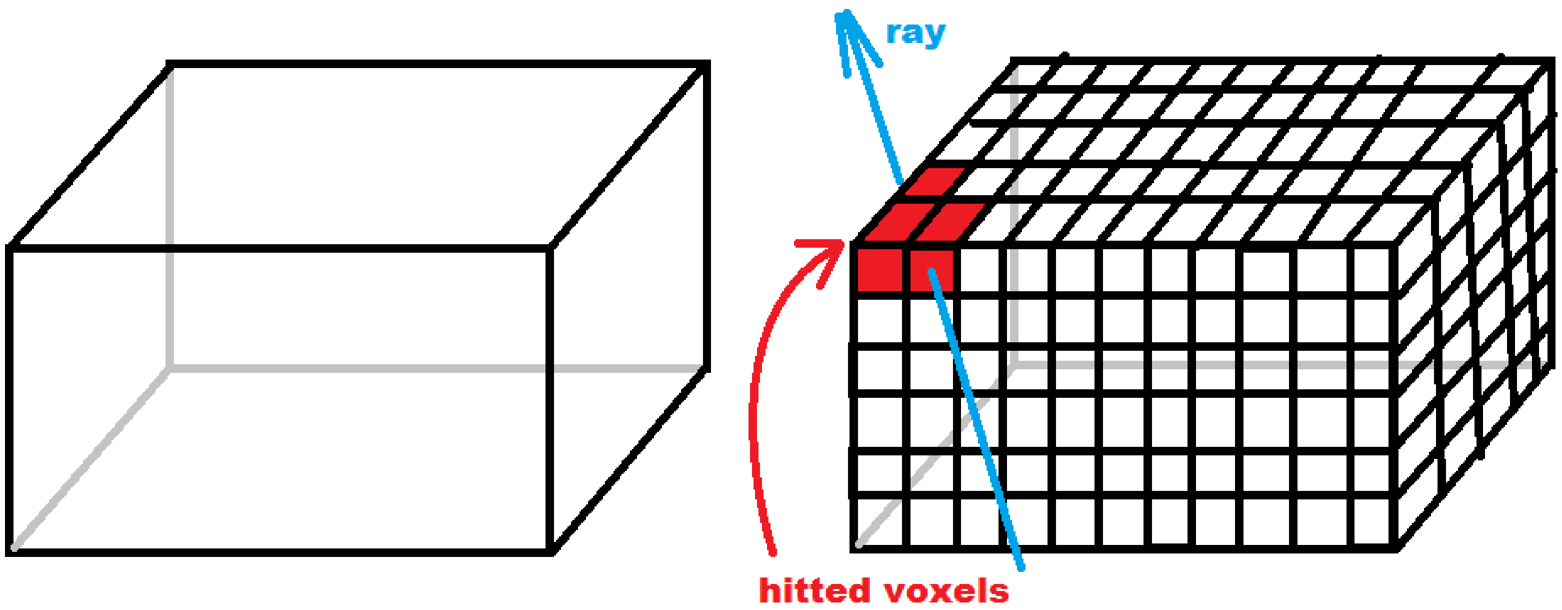
- **How to search the intersected triangles?**
  - Linear search -> too slow
  - Grid acceleration structure
    - For each cell, we record 2D triangles included



- **Partition space into equal sized volumes ("voxels")**
- **Each grid cell contains primitives that overlap voxel. (very cheap to construct acceleration structure)**
- **Walk ray through volume in order**
  - **Very efficient implementation possible (think: 3D line rasterization)**
  - **Only consider intersection with primitives in voxels the ray intersects**

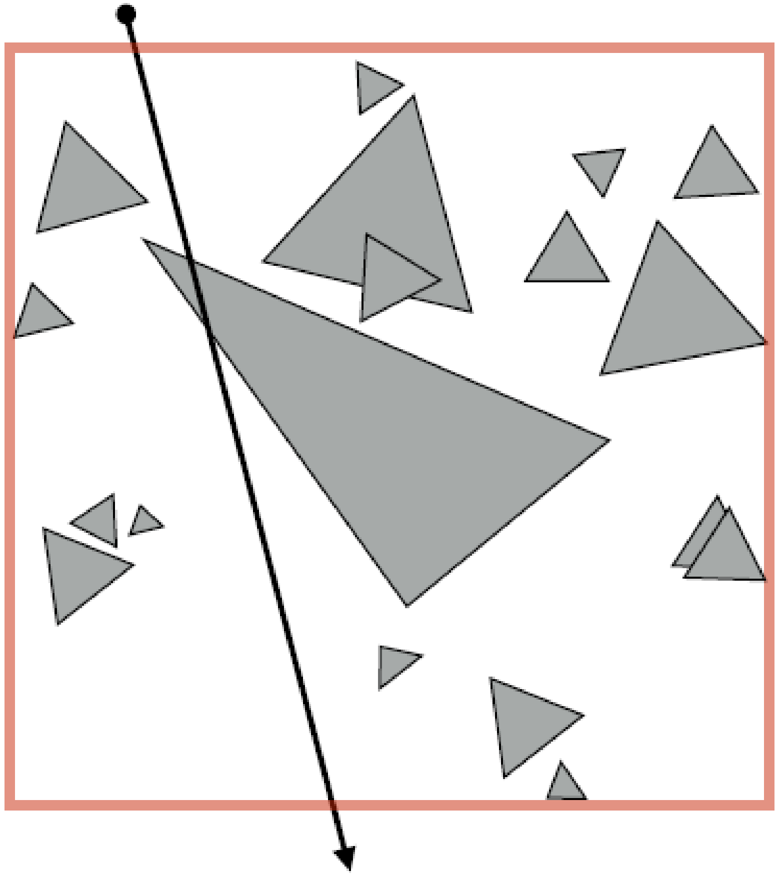
Ray-mesh intersection

- **How to search the intersected triangles?**
  - Grid acceleration structure
    - 3D case
    - For each voxel, we record 3D triangles that are included

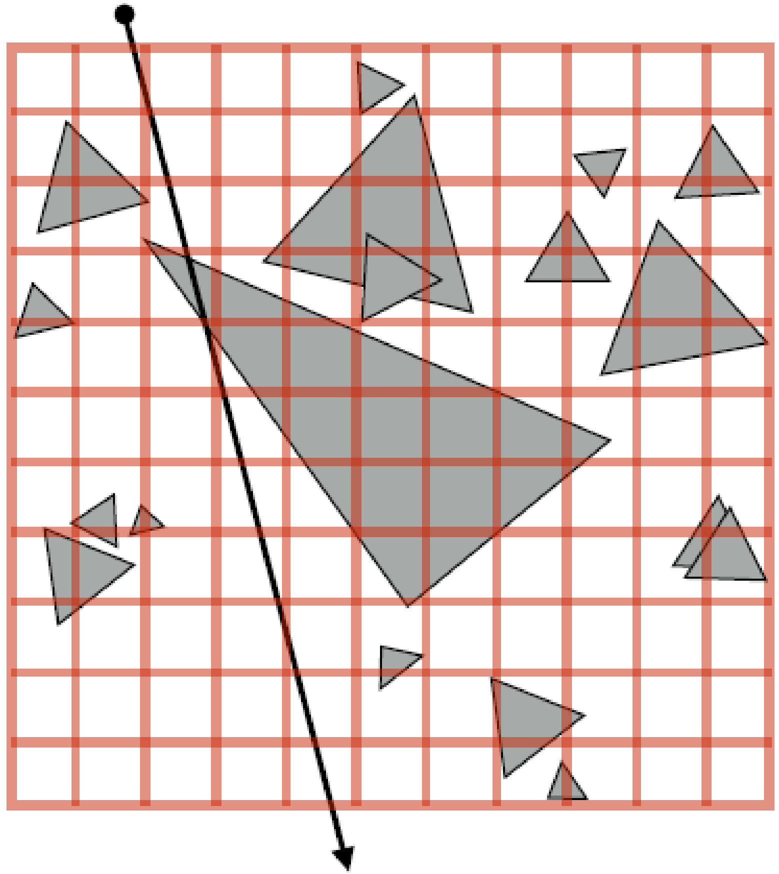


Ray-mesh intersection

- **What should the grid resolution be?**



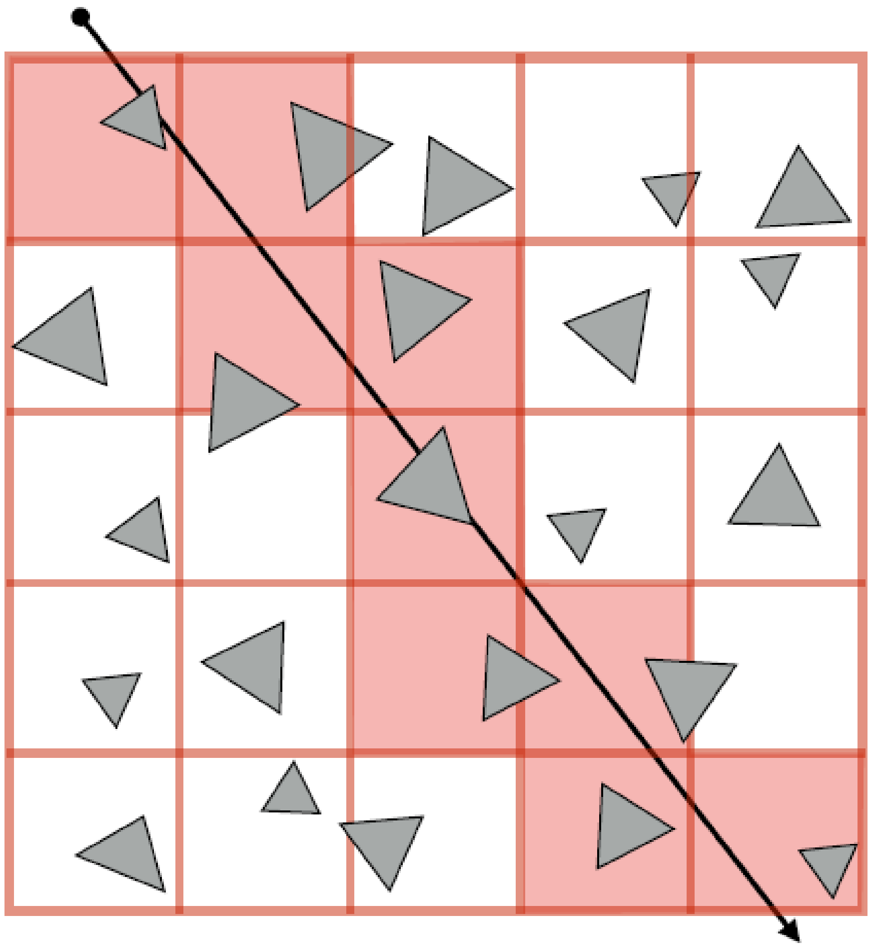
Too few grids cell: degenerates to brute-force approach



Too many grid cells: incur significant cost traversing through cells with empty space

Ray-mesh intersection

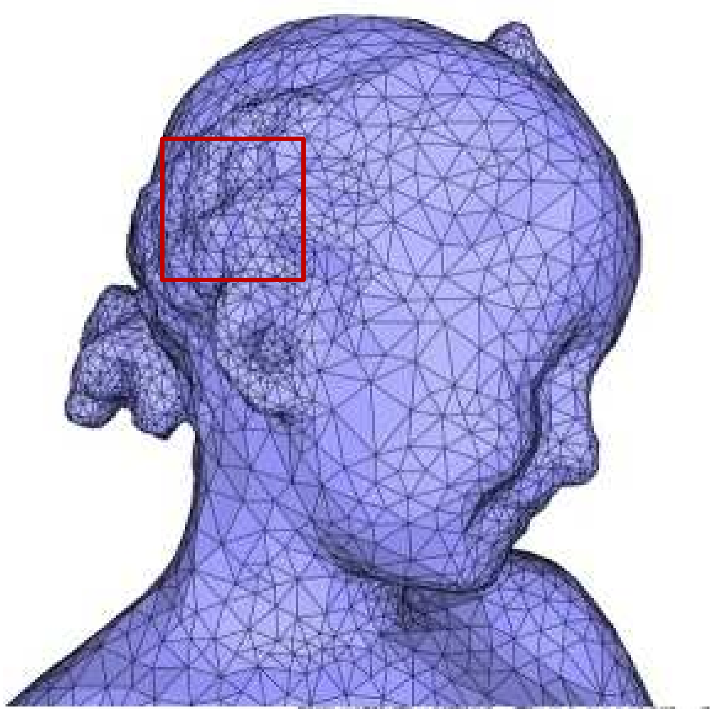
- **Heuristic**
  - **Choose number of voxels ~ total number of primitives**
    (constant prims per voxel — assuming uniform distribution of primitives)



Intersection cost: $O(\sqrt[3]{N})$

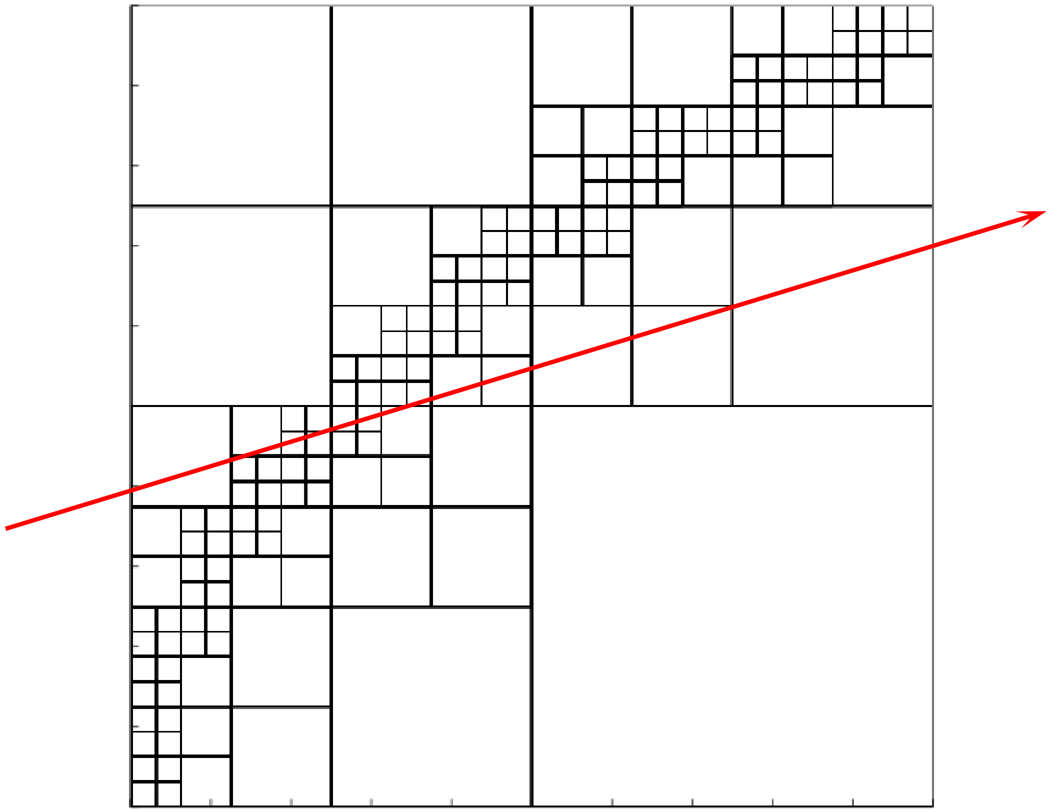
Ray-mesh intersection

- **Problems**
  - For local regions where primitives are extremely dense within a cell, the search will be slow
  - Detailed local features often have dense primitive representations



Ray-mesh intersection

- **Multi-level grid**
  - The grid cells are subdivided into subgrids which form multiple levels of grids



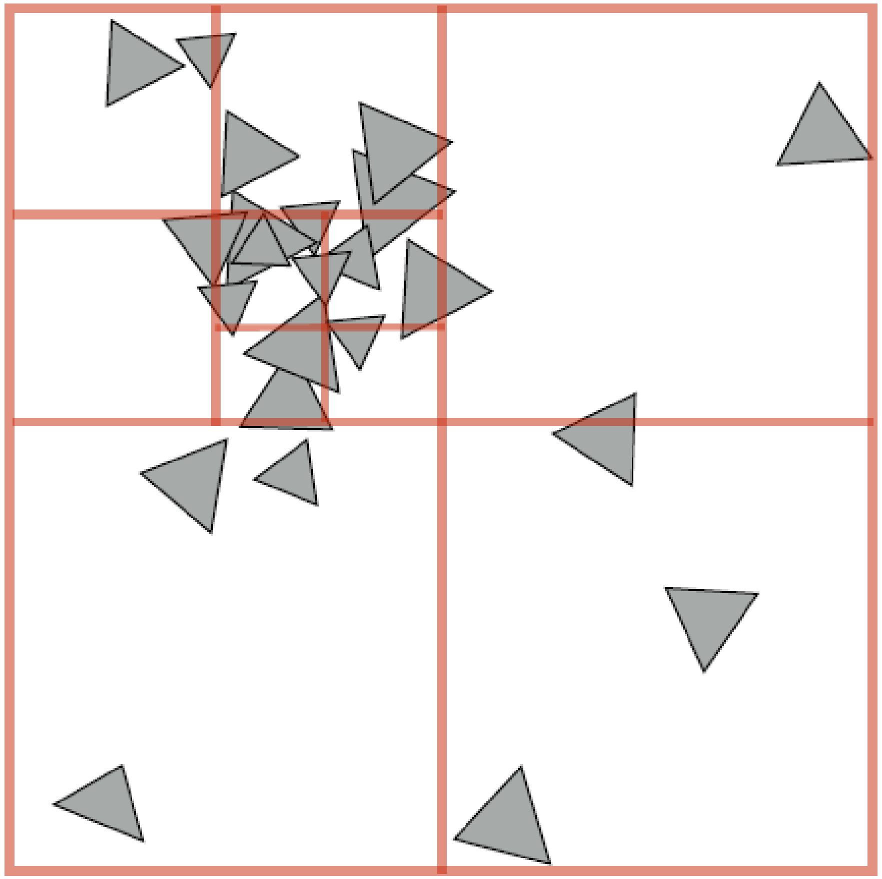
Ray-mesh intersection

- **Quad-tree / octree**

Like uniform grid: easy to build (don't have to choose partition planes)

Has greater ability to adapt to location of scene geometry than uniform grid.

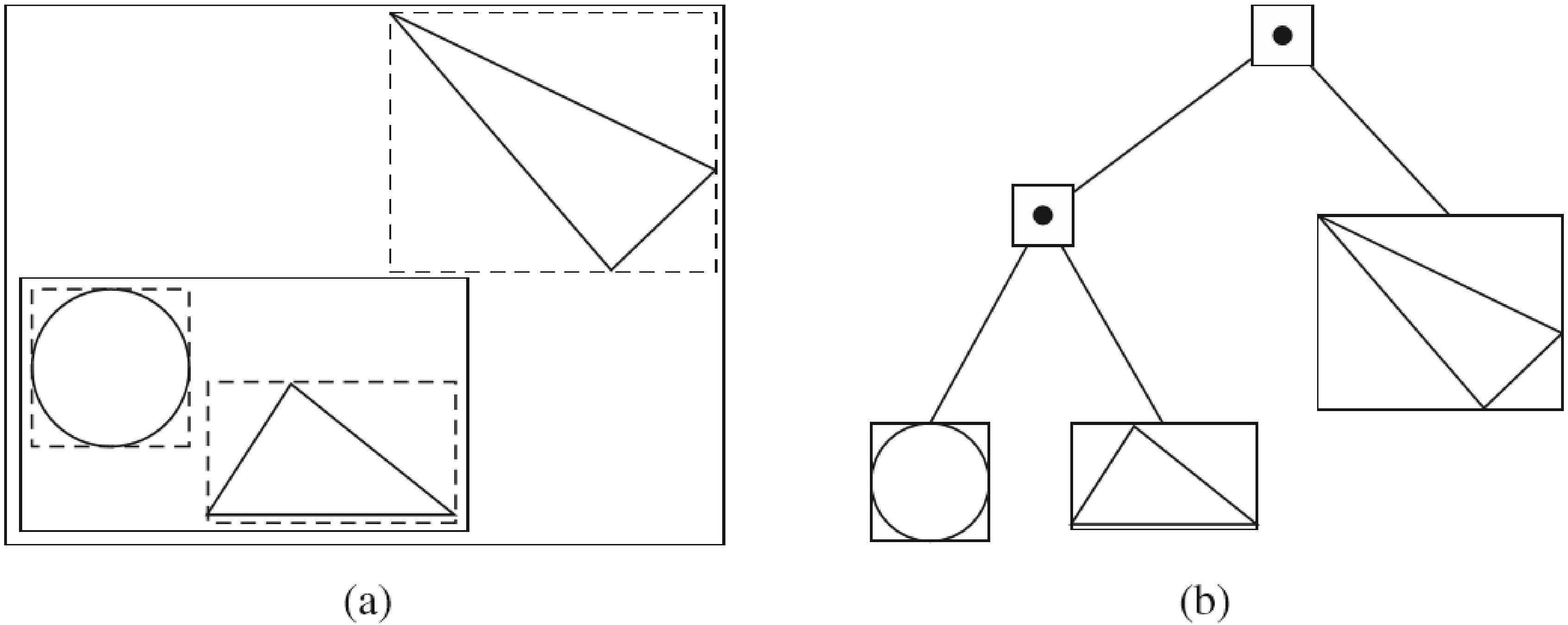
But lower intersection performance than K-D tree (only limited ability to adapt)



Quad-tree: nodes have 4 children (partitions 2D space)
Octree: nodes have 8 children (partitions 3D space)

Bounding volume hierarchies

- **Bounding volume hierarchies (BVHs)**
  - An approach for ray intersection acceleration based on primitive subdivision

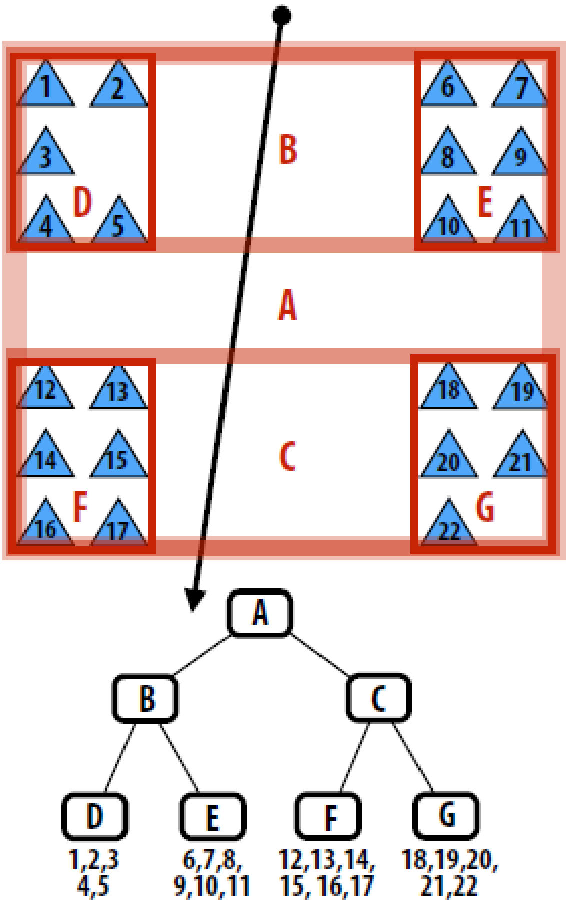


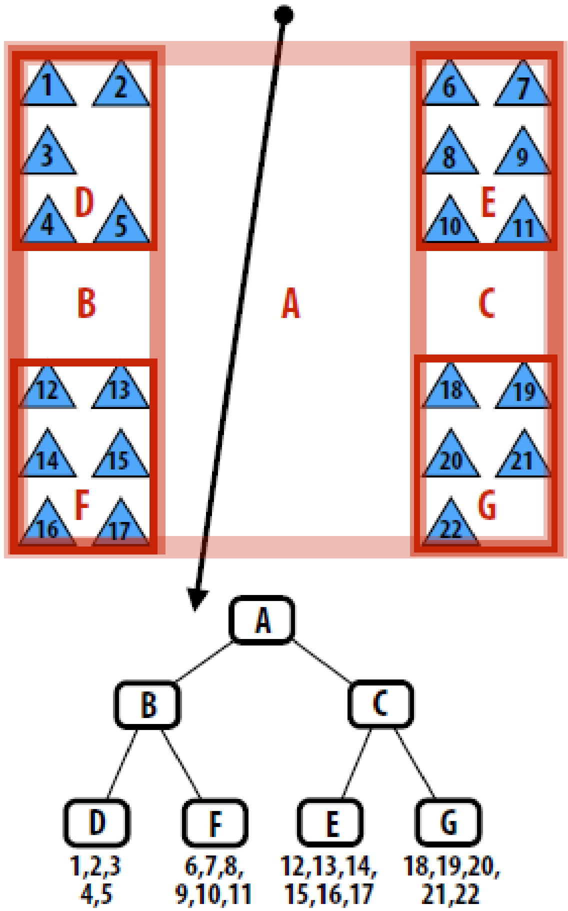
A small collection of primitives, with bounding boxes

The corresponding bounding volume hierarchy

Bounding volume hierarchies

- Interior nodes:
  - Represents subset of primitives in scene
  - Stores aggregate bounding box for all primitives in subtree
- Leaf nodes:
  - Contain list of primitives



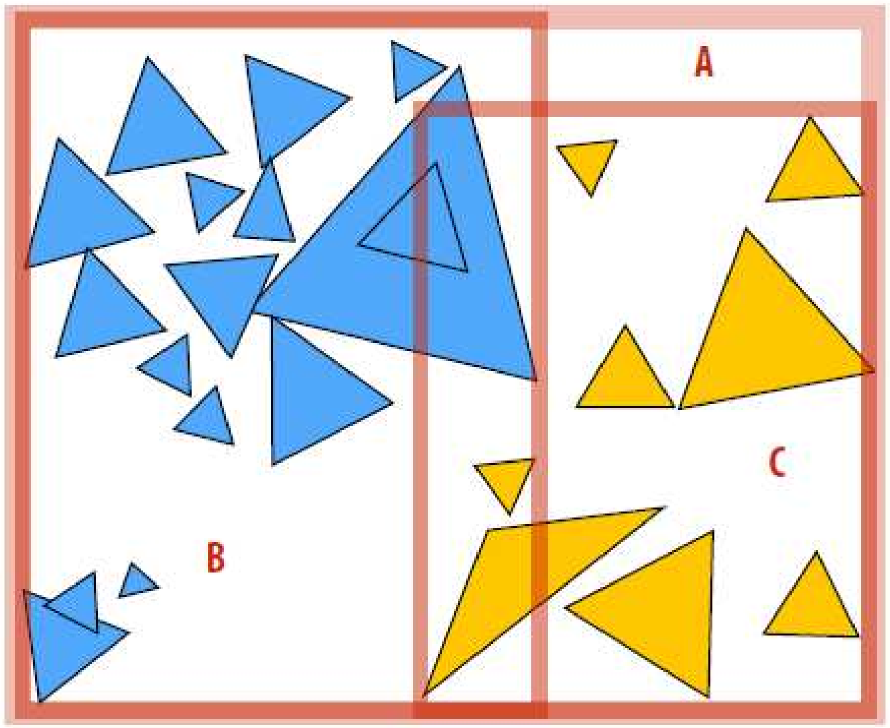


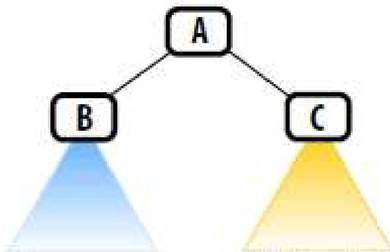
Left: two different BVH organizations of the same scene containing 22 primitives.

Is one BVH better than the other?

Bounding volume hierarchies

- **Another example**
  - **BVH partitions each node's primitives into disjoints sets**
    - **Note: The sets can still be overlapping in space (below: child bounding boxes may overlap in space)**



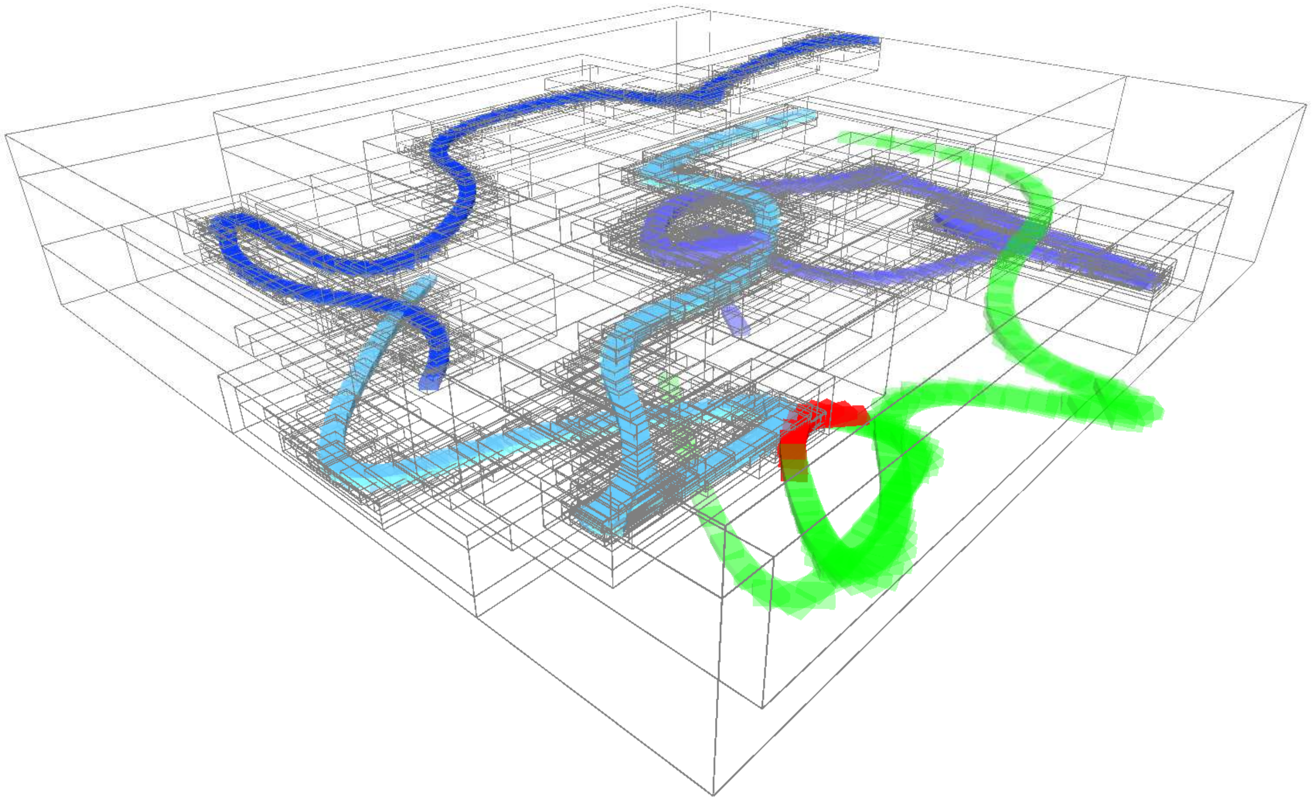


Bounding volume hierarchies

- **BVH construction**
  - **Three stages**
    - Stage 1: bounding information about each primitive is computed
    - Stage 2: The tree is built by a procedure that splits the primitives into two subsets (top-down), or merges the subsets of primitives into a larger set (bottom-up)
    - Stage 3: The tree is converted to a more compact pointer-less representation

Bounding volume hierarchies

- **Example of BVH in 3D**



2. Shading at the intersection point

Shading at intersection point

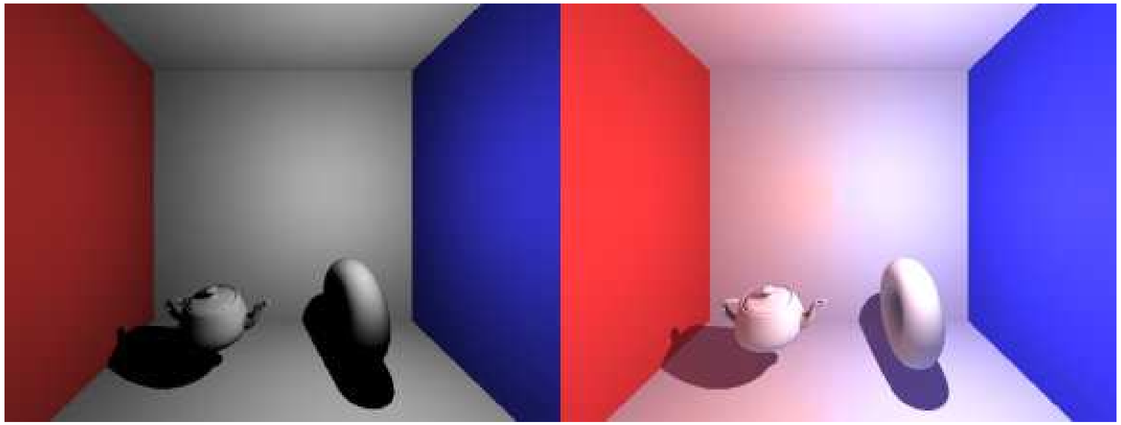
- **How to do shading?**
  - Lighting model
  - Determine reflected light intensity

- **Phong reflection model**
  - Diffuse + specular component
  - For point light source

- **More advanced BRDF reflection model**

Shading computation

- **Light sources**
  - **Local illumination**
    - Surface is illuminated by only direct light sources
    - Light sources can be point, directional, or area lights
  - **Global illumination**
    - Consider inter-reflections
    - Light sources can be direct or reflected lights



Direct light sources only

Direct light sources with inter-reflection

Shading computation

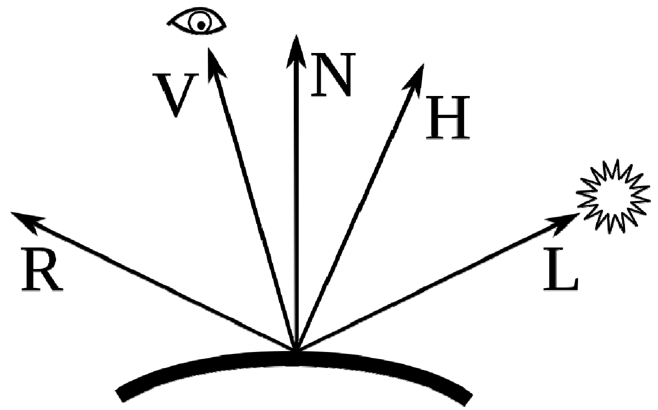
- **Determine surface reflection**
  - Diffuse surface?
  - Specular surface?
  - Mirror?
  - Combined?
  - More general? From how many light sources?

- **Compute only light intensities**
  - The ray intensity is computed for each R,G,B channel
  - Apply appropriate reflection law

Shading: Local illumination

- **Diffuse and specular reflection surface**
  - Specular lights are reflected in particular directions
  - Apply Phong reflection model to compute intensity
- **Phong shading model**
  - Ambient + diffuse + specular

$$I_p = k_a i_a + \sum_{m \in \text{lights}} (k_d (\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s (\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s})$$

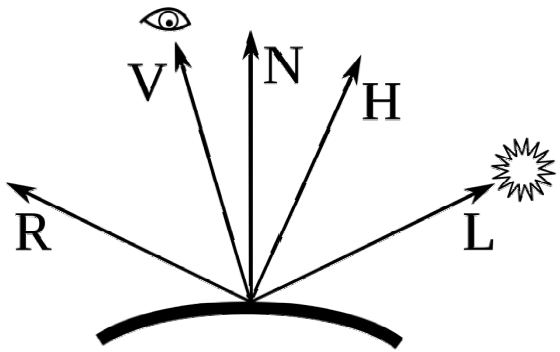


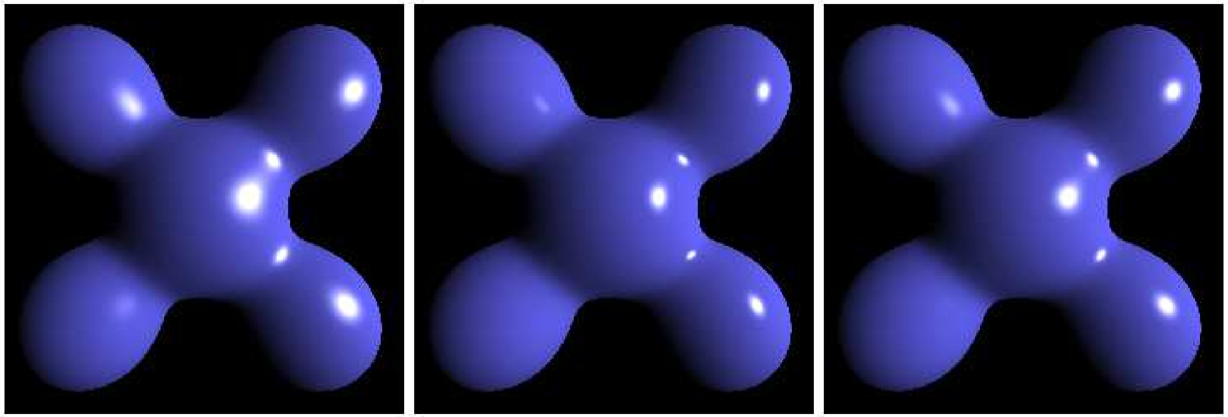
- **Blinn–Phong shading model**
  - Ambient + diffuse + specular
  - Replace ***R·V*** with ***N·H***

Shading at intersection point

- **Approximation**
  - Blinn–Phong shading model

$$H = \frac{L+V}{\|L+V\|} \qquad (N \cdot H)^{\alpha'} \Rightarrow (R \cdot V)^{\alpha}$$





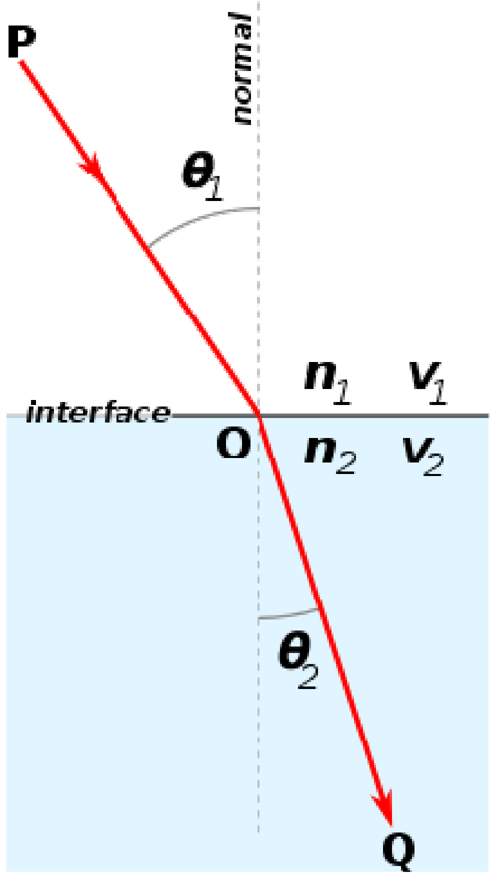
Blinn–Phong | Phong | Blinn–Phong (higher exponent)

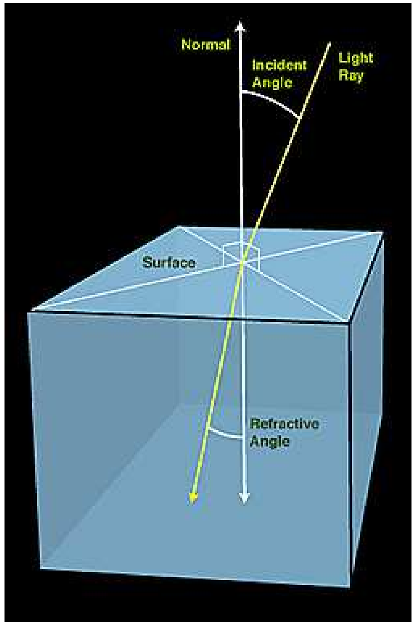
Shading: Refraction

- **Snell's law**

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{\lambda_1}{\lambda_2} = \frac{n_2}{n_1}$$

  – In 3D, we need to compute a plane determined by incident light and normal

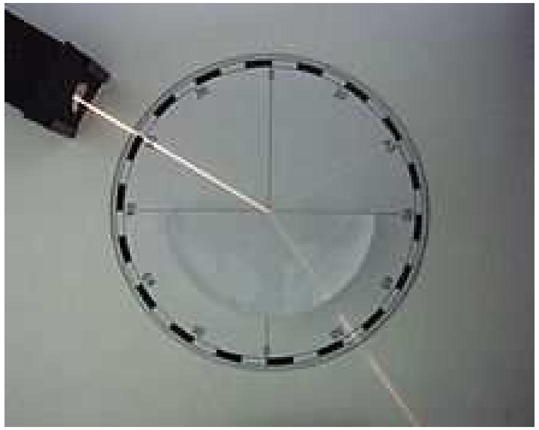


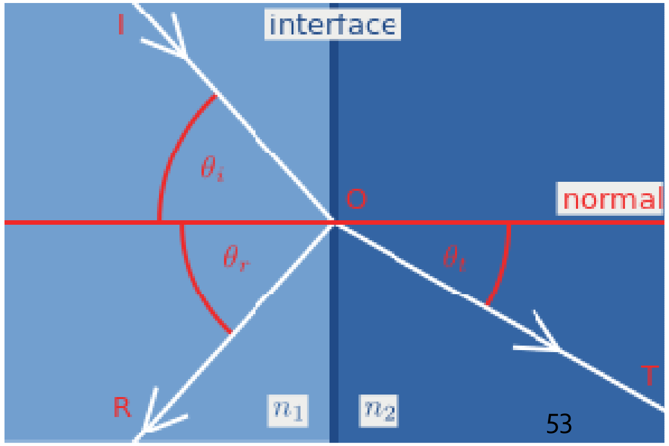


Shading: Refraction

- **At translucent interface**
  - Both reflection and refraction happen
  - How much is reflected?
    - Fresnel's law
  - Polarized light (fraction)
    - s-polarized light $R_s = \left|\frac{n_1 \cos\theta_i - n_2 \cos\theta_t}{n_1 \cos\theta_i + n_2 \cos\theta_t}\right|^2$
    - p-polarized light $R_p = \left|\frac{n_1 \cos\theta_t - n_2 \cos\theta_i}{n_1 \cos\theta_t + n_2 \cos\theta_i}\right|^2$
  - Unpolarized light (fraction)

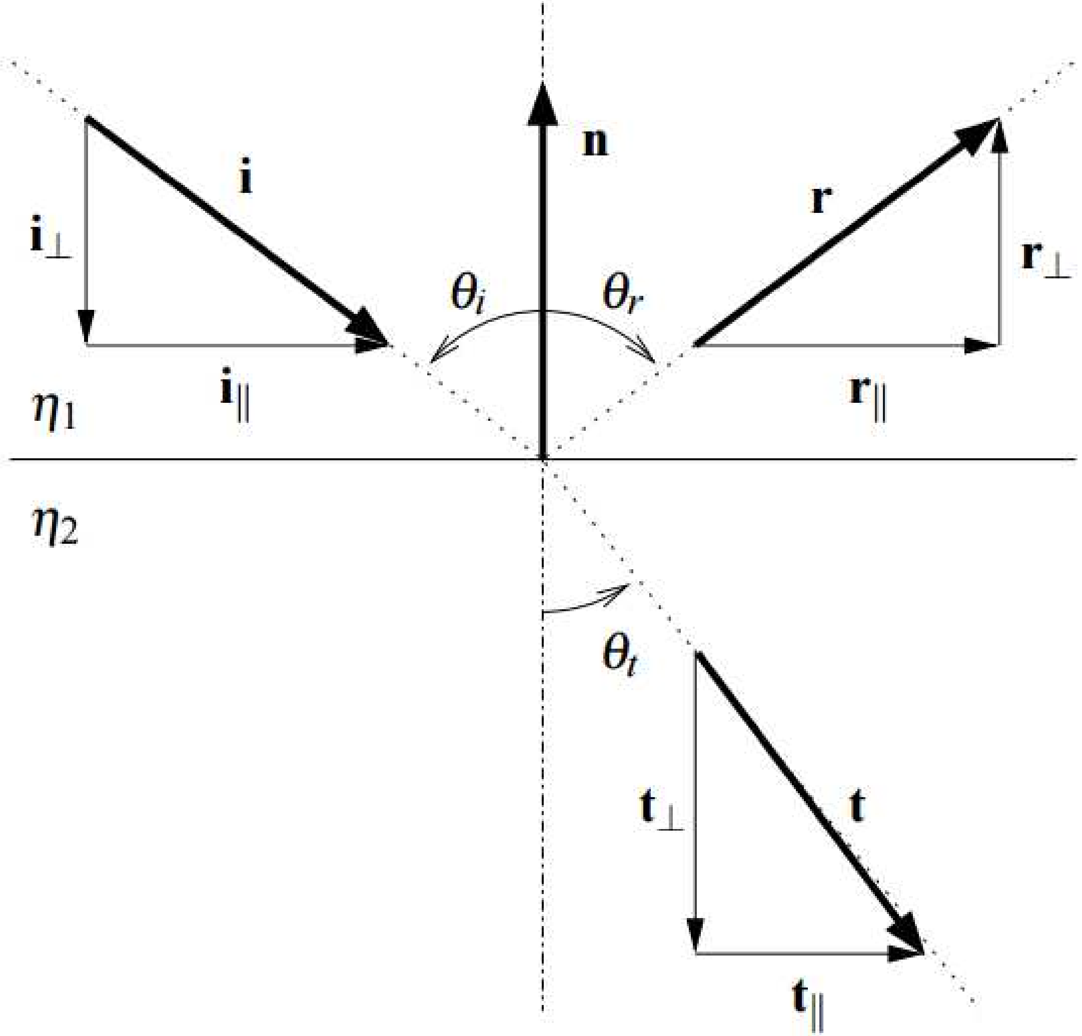
$$R = \frac{1}{2}(R_s + R_p)$$





Shading: computing the reflection & refraction rays

- **The configuration**



Shading: computing the reflection & refraction rays

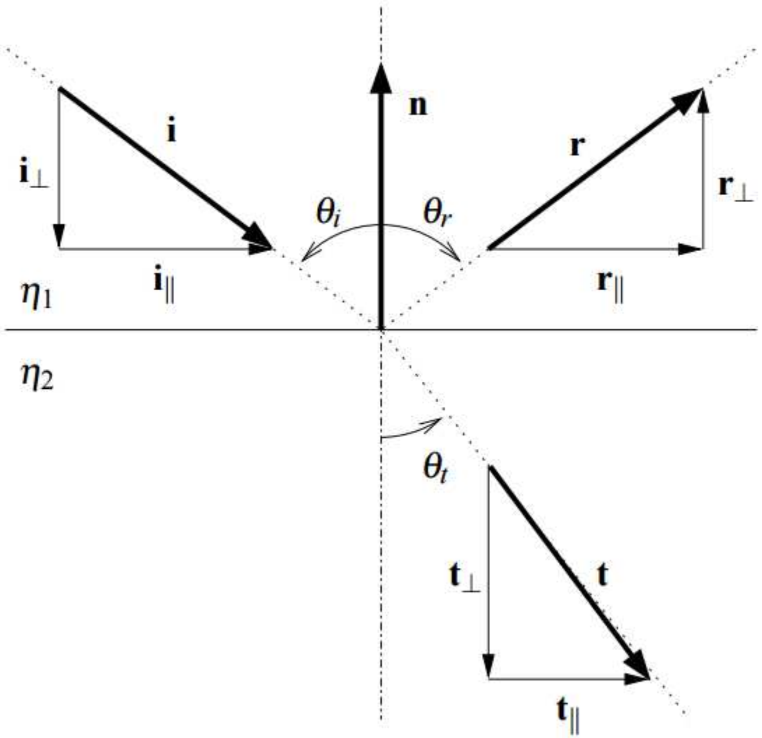
- **Computing the reflection ray**
  - Using normalized vectors

$$
\begin{aligned}
\mathbf{r}_\perp &= -\mathbf{i}_\perp \\
\mathbf{r}_\parallel &= \mathbf{i}_\parallel
\end{aligned}
$$

$$\mathbf{r} = \mathbf{r}_\parallel + \mathbf{r}_\perp = \mathbf{i}_\parallel - \mathbf{i}_\perp$$

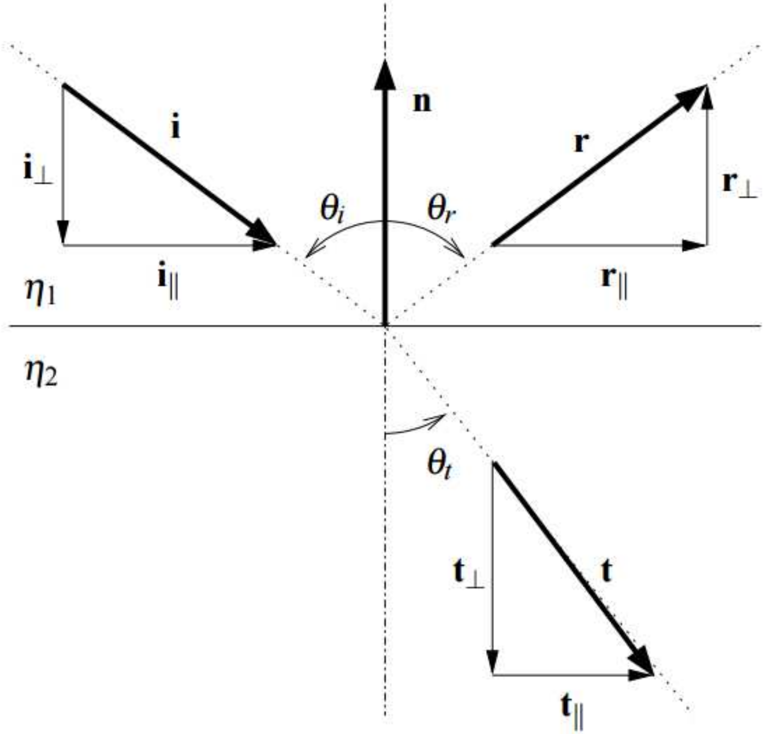
$$\Downarrow$$

$$
\begin{aligned}
\mathbf{r} &= \mathbf{i}_\parallel - \mathbf{i}_\perp \\
&= [\mathbf{i} - (\mathbf{i}\cdot\mathbf{n})\,\mathbf{n}] - (\mathbf{i}\cdot\mathbf{n})\,\mathbf{n} \\
&= \mathbf{i} - 2\,(\mathbf{i}\cdot\mathbf{n})\,\mathbf{n}
\end{aligned}
$$



Shading: computing the reflection & refraction rays

- **Computing the refraction ray**
  - Using normalized vectors



$$\eta_1 \sin\theta_i = \eta_2 \sin\theta_t$$

$$\mathbf{t} = \mathbf{t}_\parallel + \mathbf{t}_\perp$$

$$\mathbf{t}_\parallel = \frac{\eta_1}{\eta_2}\mathbf{i}_\parallel = \frac{\eta_1}{\eta_2}[\mathbf{i} + \cos\theta_i \mathbf{n}] \qquad \mathbf{t}_\perp = -\sqrt{1 - |\mathbf{t}_\parallel|^2}\,\mathbf{n}$$

$$\Downarrow$$

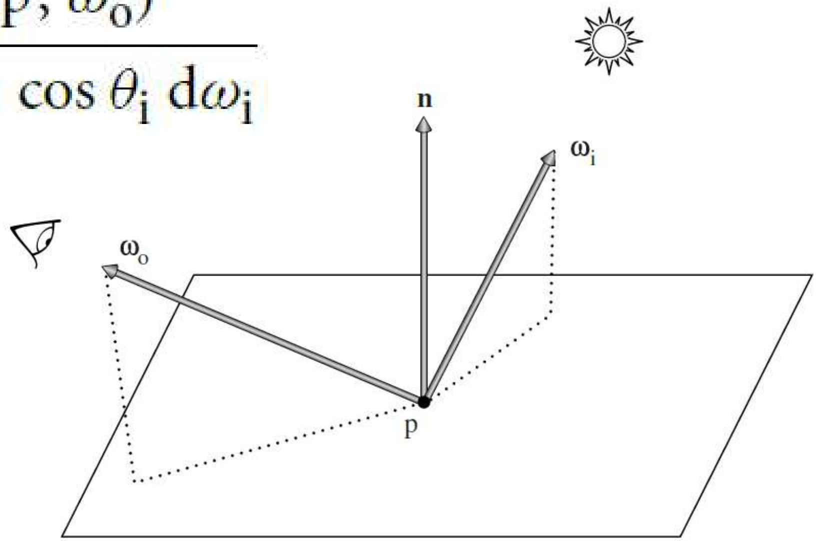
$$\mathbf{t} = \frac{\eta_1}{\eta_2}\mathbf{i} + \left(\frac{\eta_1}{\eta_2}\cos\theta_i - \sqrt{1 - |\mathbf{t}_\parallel|^2}\right)\mathbf{n} = \frac{\eta_1}{\eta_2}\mathbf{i} + \left(\frac{\eta_1}{\eta_2}\cos\theta_i - \sqrt{1 - \sin^2\theta_t}\right)\mathbf{n}$$

$$\sin^2\theta_t = \left(\frac{\eta_1}{\eta_2}\right)^2 \sin^2\theta_i = \left(\frac{\eta_1}{\eta_2}\right)^2 (1 - \cos^2\theta_i)$$

Shading: the rendering equation

- **Bidirectional reflectance distribution function (BRDF)**
  - A formalism for describing reflection from a surface
  - How much radiance is leaving the surface as a result of incident radiance

$$f_r(p, \omega_o, \omega_i) = \frac{dL_o(p, \omega_o)}{dE(p, \omega_i)} = \frac{dL_o(p, \omega_o)}{L_i(p, \omega_i) \cos\theta_i \, d\omega_i}$$



Phong lighting model is a special BRDF

Shading: the rendering equation

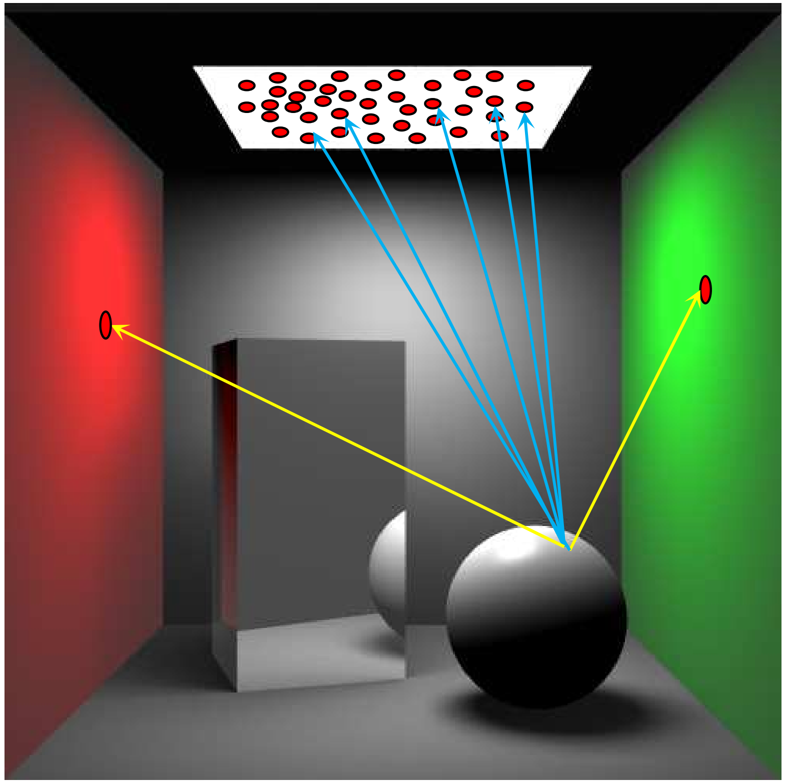
- **The fundamental rendering equation**
  - Describe how an incident distribution of light at point is transformed into an outgoing distribution

$$L_o(\mathrm{p}, \omega_o) = \int_{S^2} f(\mathrm{p}, \omega_o, \omega_i)\, L_i(\mathrm{p}, \omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i$$

  - When S^2 (the entire sphere) is the domain, it is often called the *scattering equation*
  - When upper hemisphere is the domain, it is often called the *reflection equation*

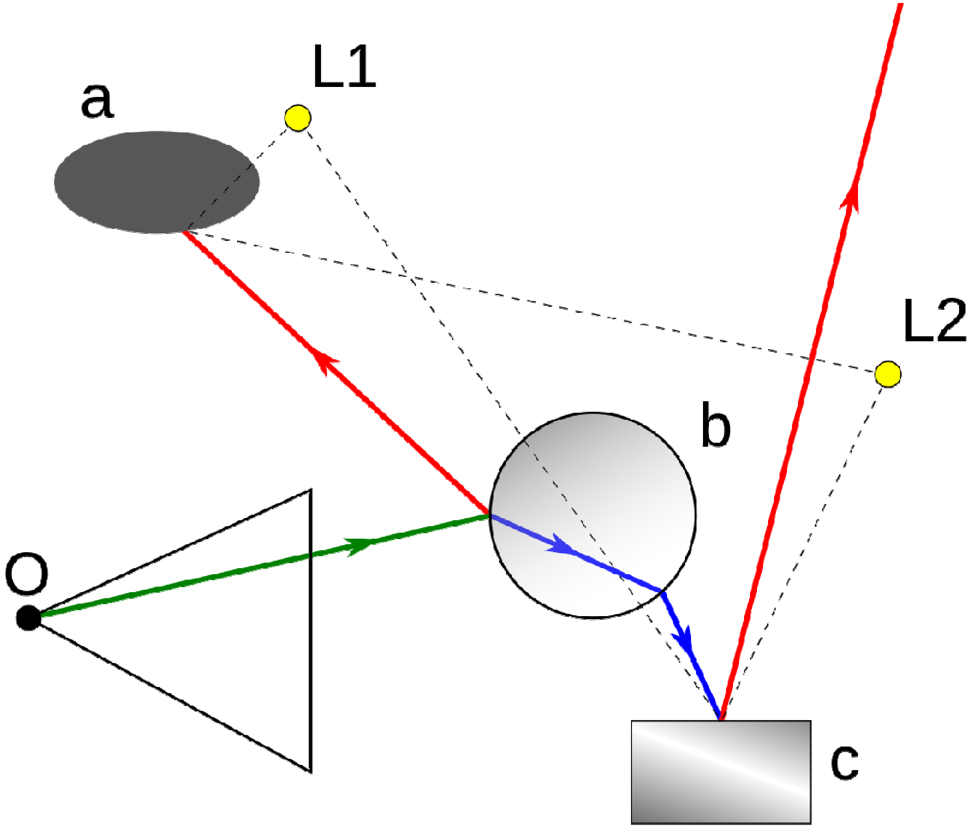
Shading: the rendering equation

- **Breaking the limit of point light source**
  - Sample light points on area light source (direct & indirect light sources)
  - Sum together all the contributions from point light sources



Light ray tracing

- **Light ray energy is accumulated along the tracing rays**



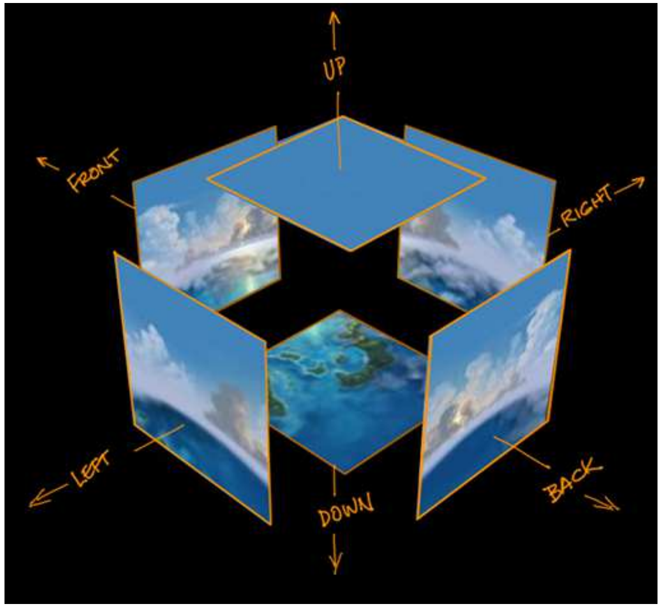
Environment map

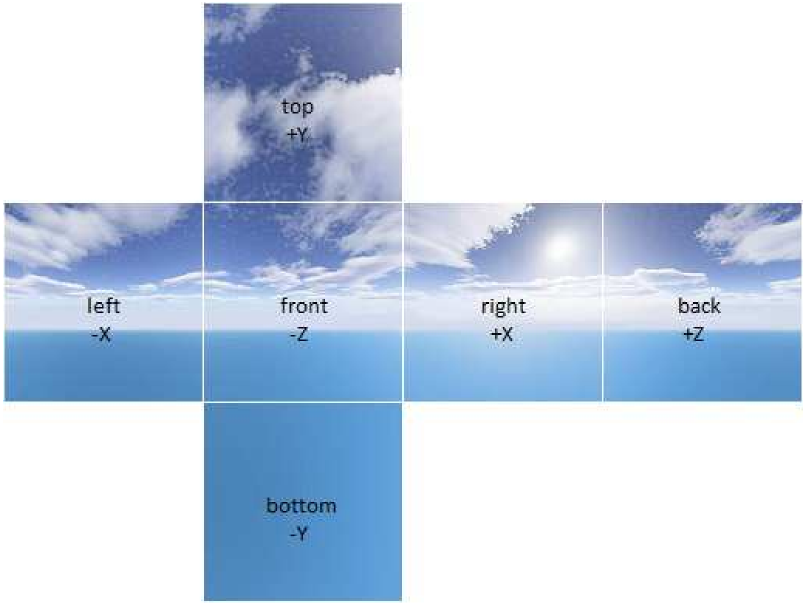
- **At any point, the environment will cast light onto that point**

- **Environment mapping**
  – An efficient image-based lighting technique for approximating the environment light source
  – Store light sources as environment textures
  – Cube-mapping, sphere mapping

Environment map

- **Environment mapping**
  - Cube mapping





Environment map

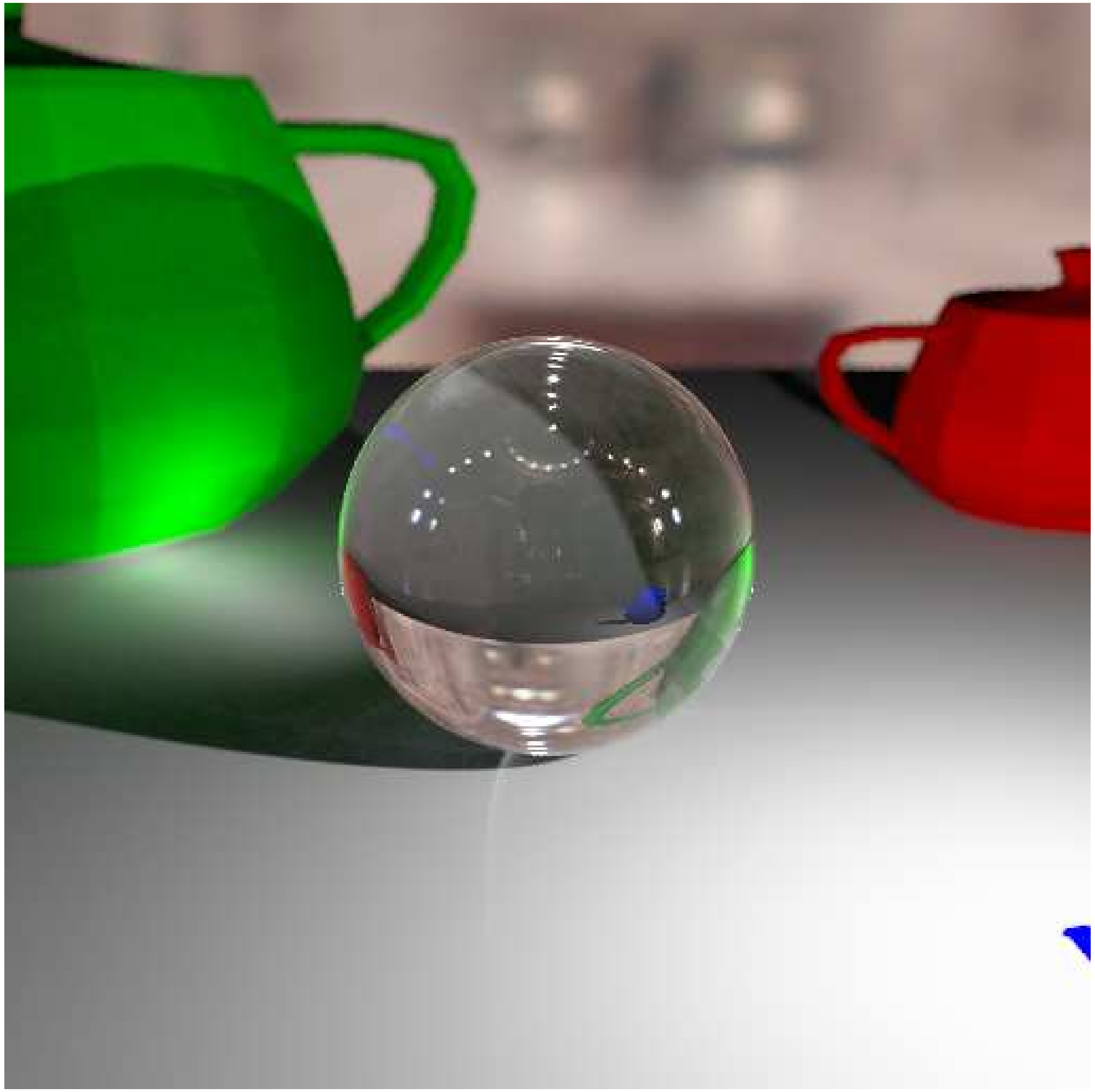
- **Environment mapping**
  - Sphere mapping

Shading with environment map

- **In principle**
  - Every pixel on the environment map is taken as a light source
- **Acceleration**
  - Importance sampling
  - Samples taken as point light sources

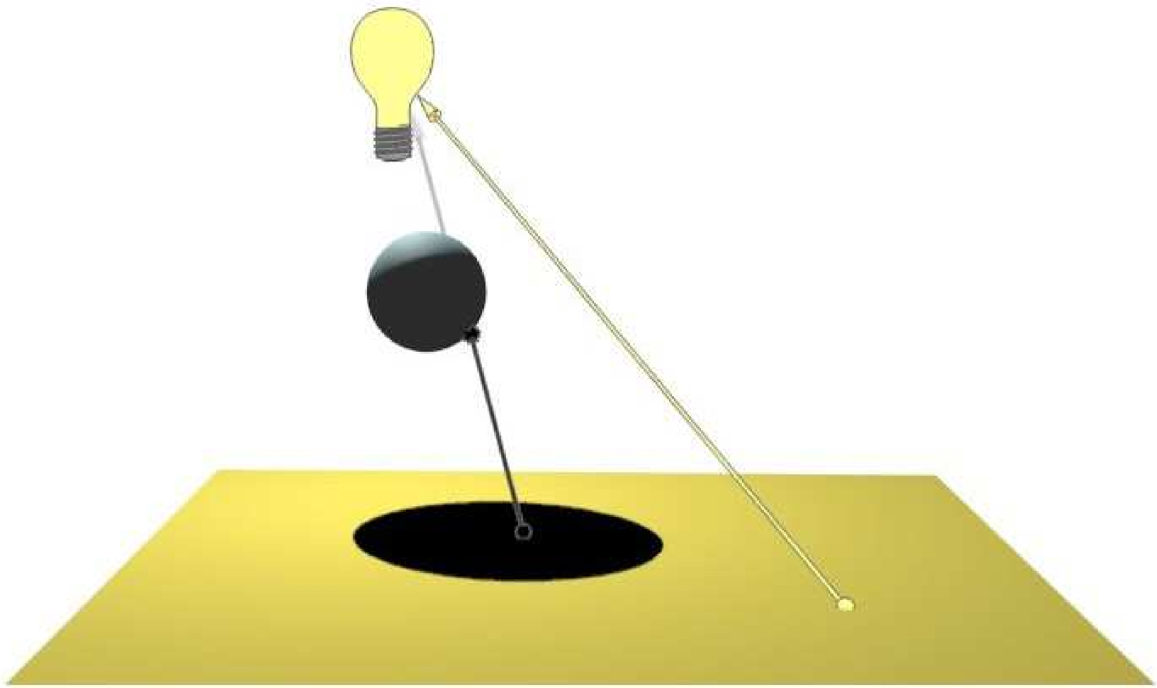
Shading with environment map

- **An example**



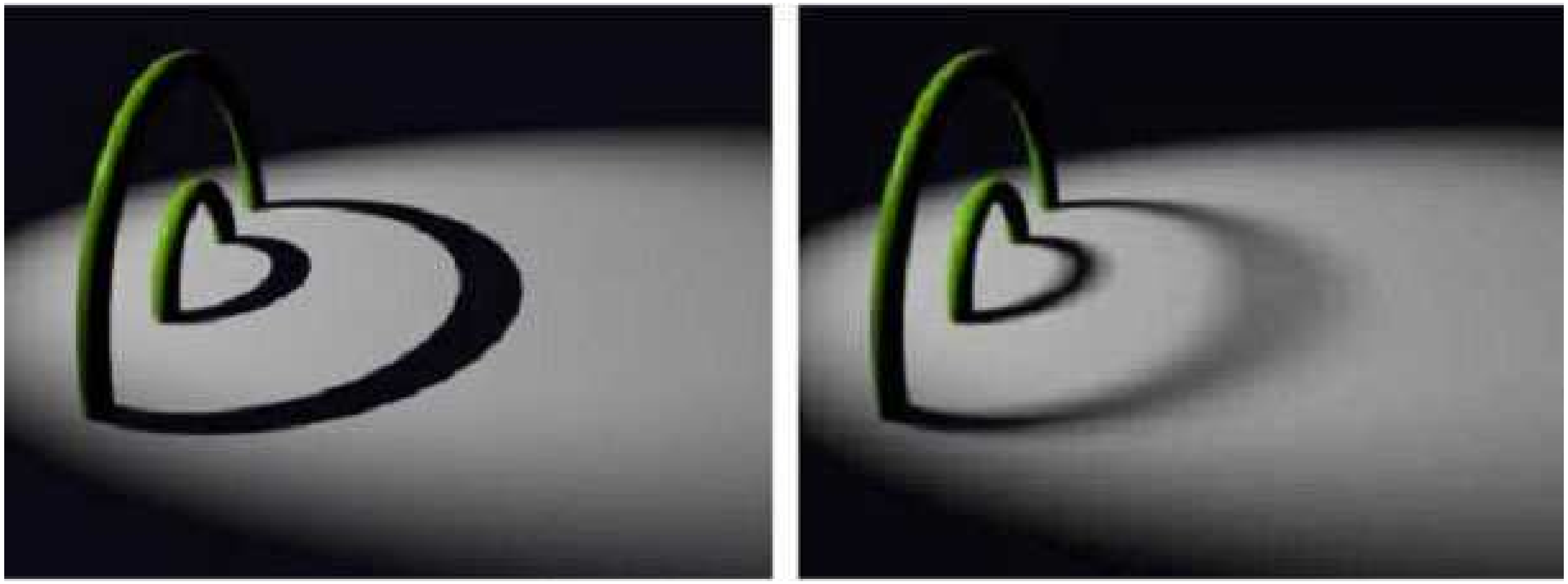
Shadow rendering

- **At each intersection point**
  - A shadow ray is generated towards light source
  - If intersection with other objects, the point is inside shadow region



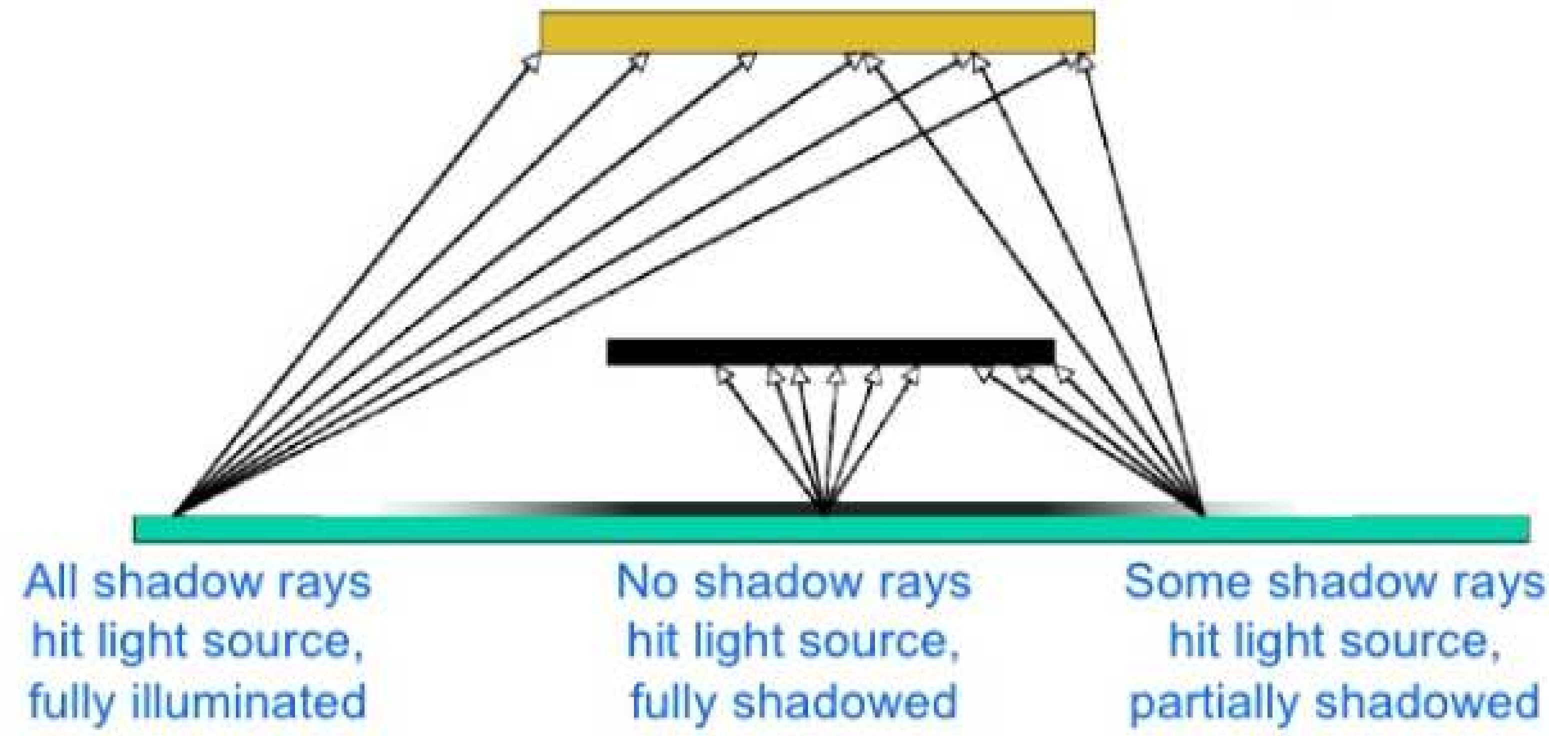
Shadow rendering

- **Hard shadow v.s. soft shadow**
  - Hard shadow is generated by point light source only
  - Soft shadow is generated by area light source



Shadow rendering

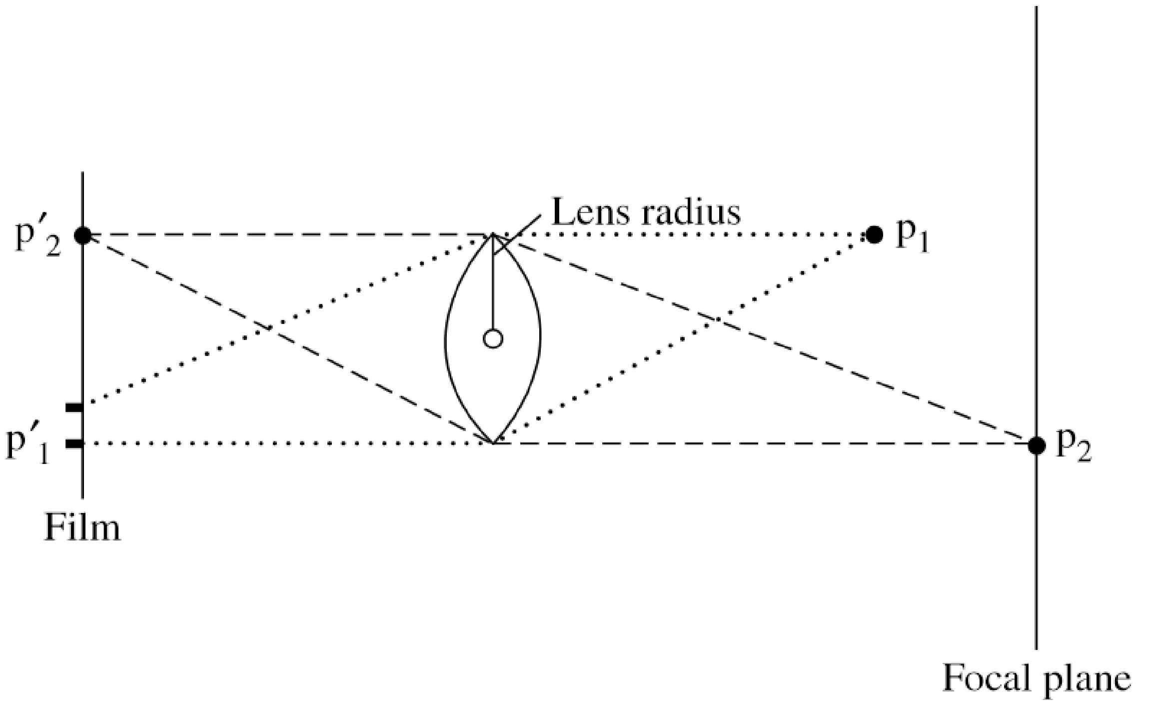
- **Shadow generation in ray tracing**
  - Cast multiple shadow rays for soft shadow



3. More advanced camera models

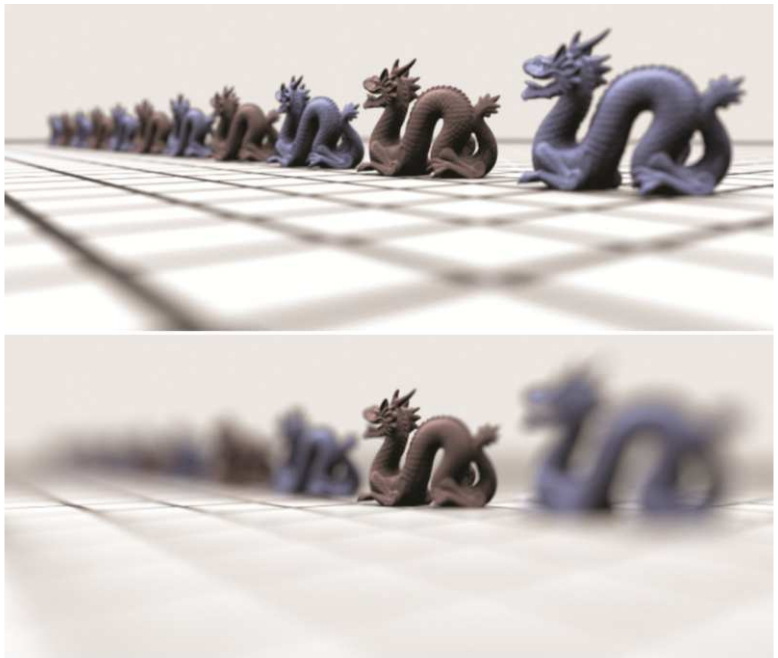
Depth of field

- **Real camera**
  - Have lens systems that focus light through a finite-sized aperture onto the film plane
  - A single point in the scene may be projected onto an area on the film plane (circle of confusion)



Depth of field

- **Real camera**
  - A finite area of the scene may be visible from a single point on the image plane, giving a blurred image
  - The size of the circle of confusion is affected by the radius of aperture and the distance between the objects and the lens

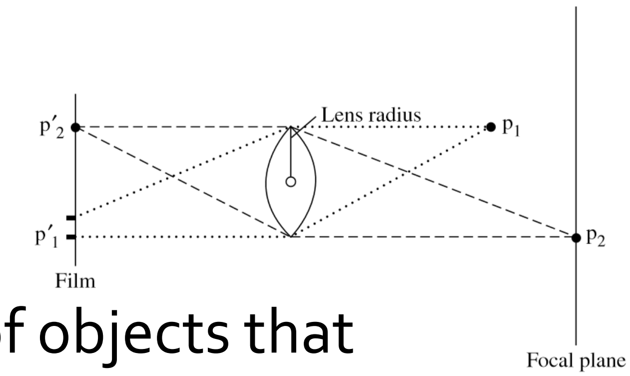


Smaller aperture size

Larger aperture size

Depth of field

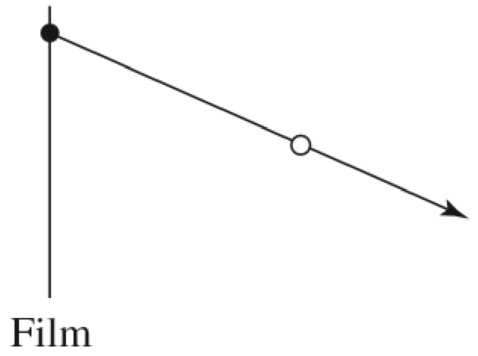
- **Real camera**
  - **Focal distance**
    - The distance from the lens to the plane of objects that project to a zero-radius circle of confusion
    - These points appear to be perfectly in focus
  - **In practice**
    - Objects do not have to be exactly on the focal plane to appear in sharp focus
    - As long as the circle of confusion is roughly smaller than a pixel
    - The range of distance from the lens where objects appear in focus in called the lens' ***depth of field***



Depth of field

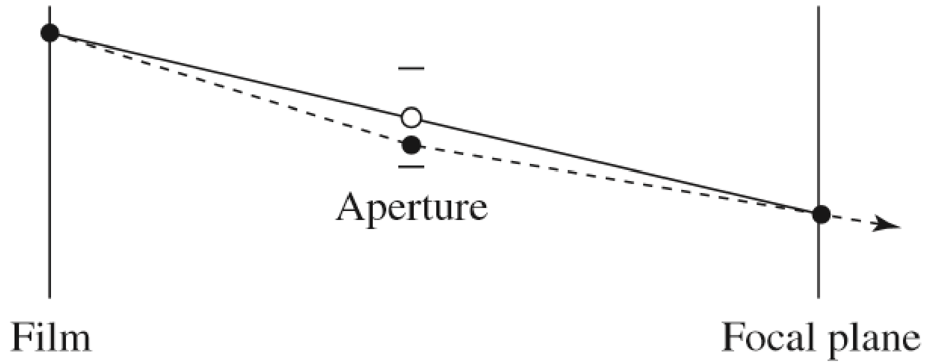
- **How to compute?**
  - Sample more rays that go through a finite lens

(a)



Zero aperture size
(Pin-hole camera)

(b)



Finite aperture size
(Real camera)

Depth of field

- **How to compute?**
  - Sampling rate is important to give plausible rendering results

4 samples per pixel

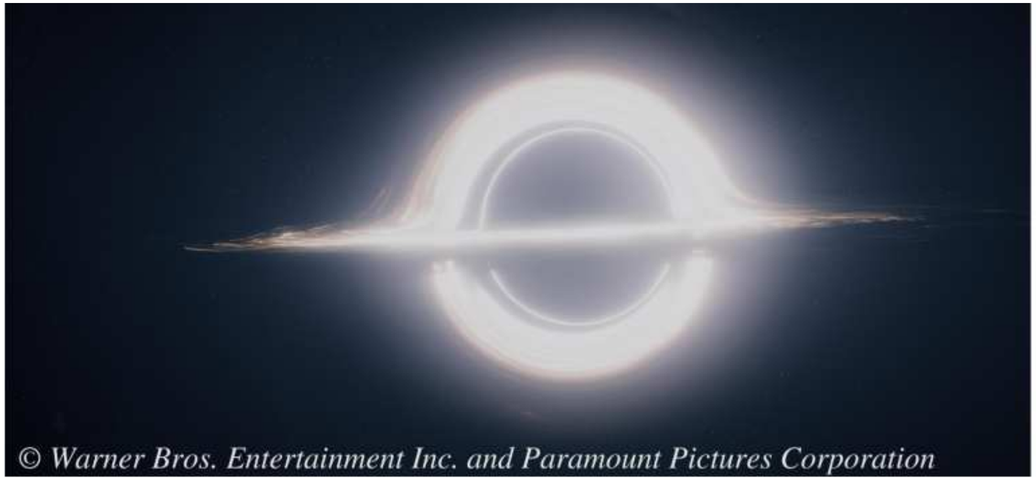
128 samples per pixel

More camera models

- **Compared to projection-based rendering**
  - Ray-tracing is easy to employ unusual image projections (sometimes highly non-linear)
  - For example, *environment camera*, which traces rays in all directions around a point in the scene

More general optical rays

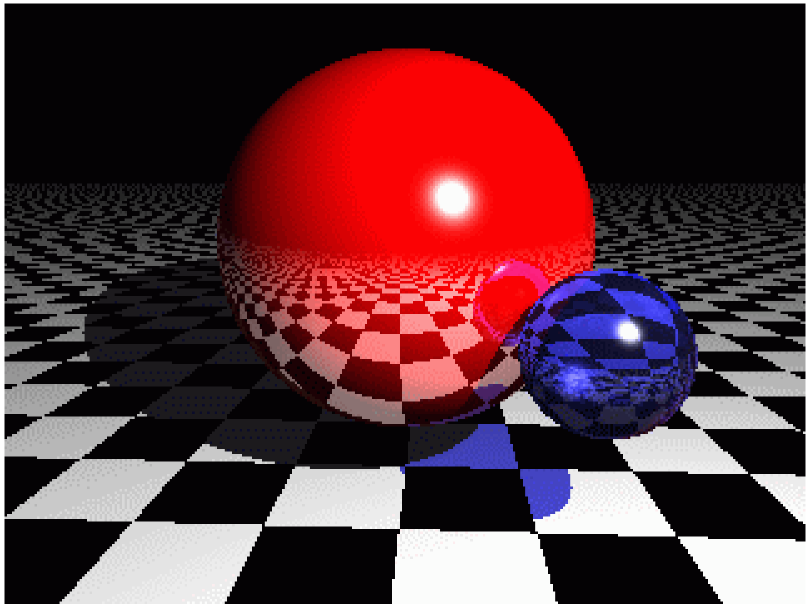
- **Optical rays can be bent over space**
  - Example: blackhole rendering in movie “Intestellar”
  - Three scientific papers, two in physics journals, one in ACM SIGGRAPH talk



4. Anti-aliasing

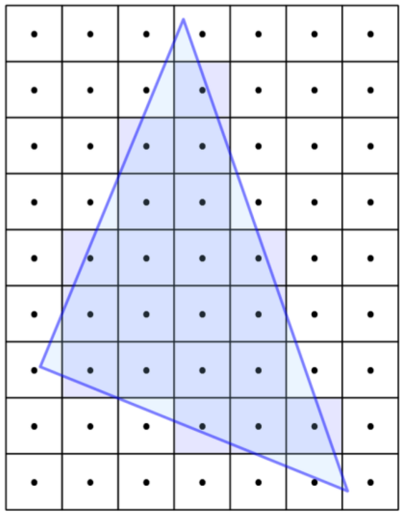
Aliasing problem

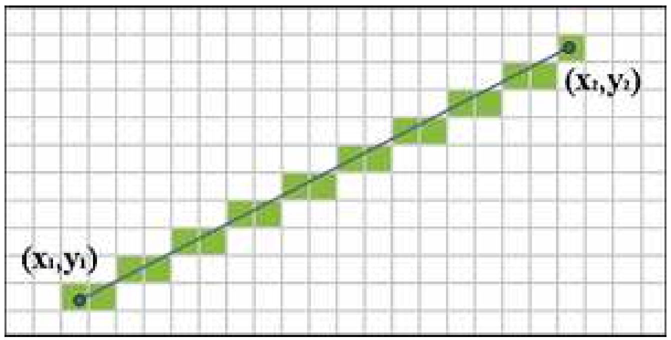
- **How does aliasing problem occur?**
  - Not enough samples to reconstruct original continuous signal
  - Zigzag artifacts usually observed



Aliasing problem

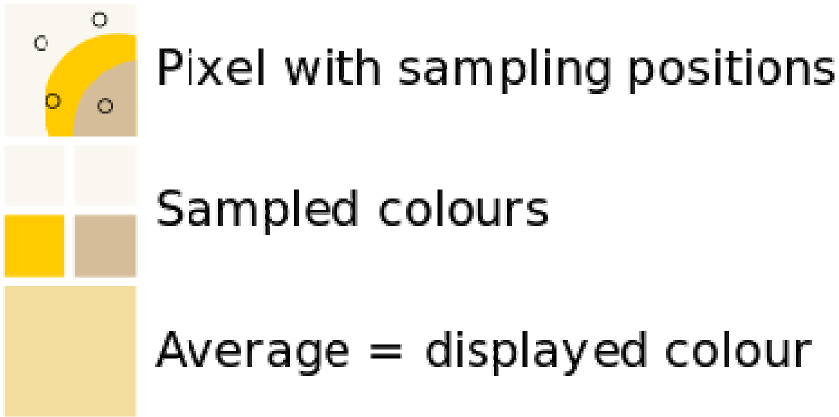
- **Where can we find aliasing problems obvious?**
  - Sharp geometric changes that create shading changes
  - Shading edges
  - Rasterization causes the problem





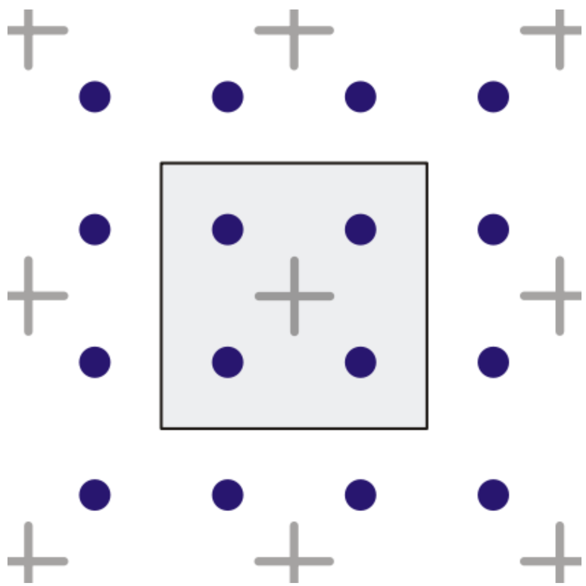
Anti-aliasing

- **Super-sampling**
  - Instead of shooting one ray per pixel, shoot multiple rays
  - This is essentially a super-sampling process
  - The final intensity is the (weighted) average of the sampled intensities



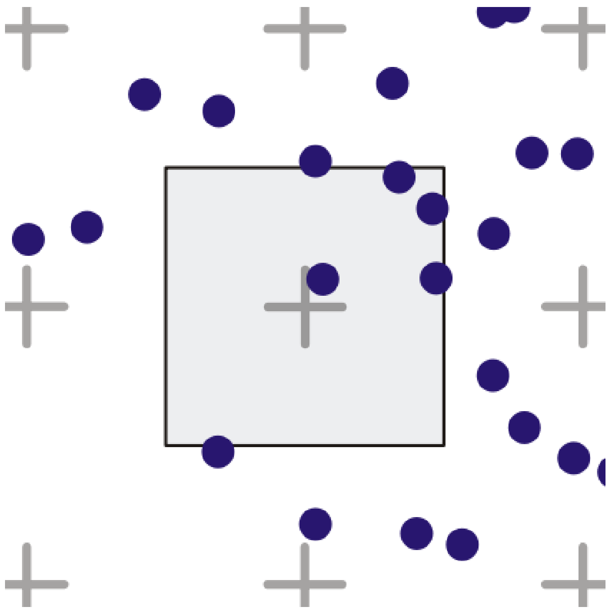
Anti-aliasing

- **Super-sampling**
  - Sampling method
    - Grid sampling pattern : aliasing can still occur if a low number of sub-pixels is used



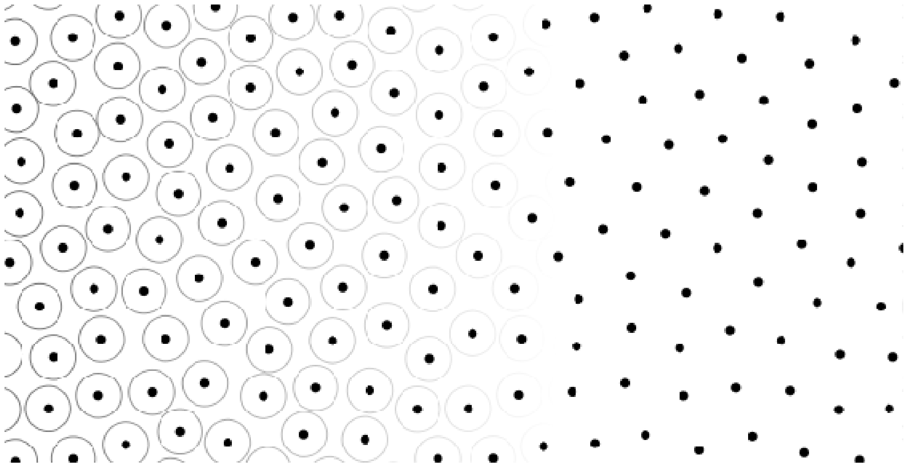
Anti-aliasing

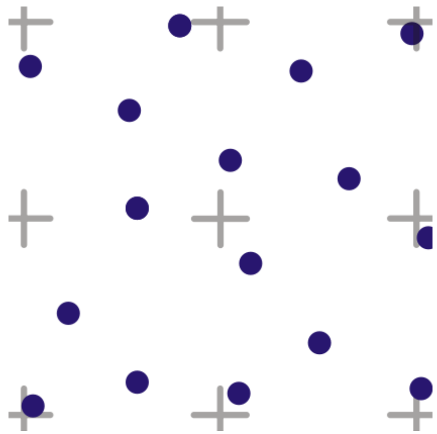
- **Super-sampling**
  - Sampling method
    - Random sampling, also known as stochastic sampling
    - Samples end up being unnecessary in some areas of the pixel and lacking in others



Anti-aliasing

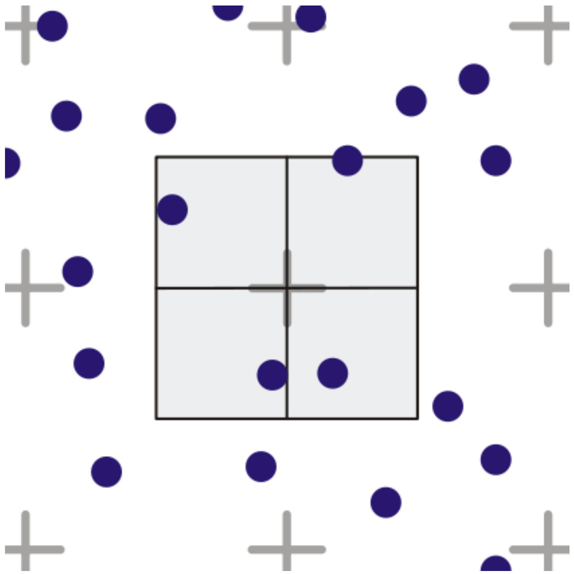
- **Super-sampling**
  - Sampling method
    - Poisson-disk : an algorithm that places the samples randomly, but make sure any two are not too close
    - The end result is an even but random distribution of samples
    - The computational time for this algorithm is great





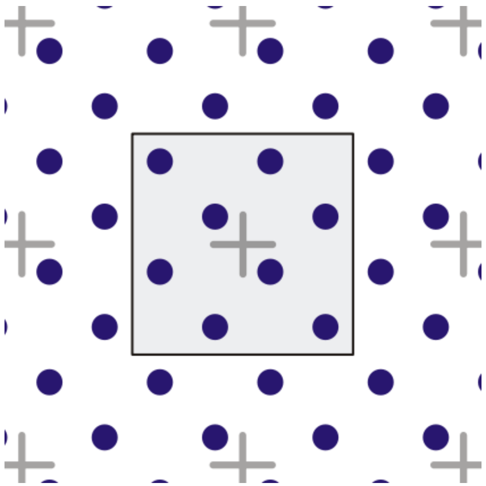
Anti-aliasing

- **Super-sampling**
  - Sampling method
    - Jittered sampling : A modification of the grid algorithm to approximate the Poisson disc
    - A pixel is split into several sub-pixels, but a sample is taken from a random point within the sub-pixel
    - Congregation can still occur, but to a lesser degree



Anti-aliasing

- **Super-sampling**
  - Sampling method
    - Rotated grid : A 2×2 grid layout is used rotated to avoid sample alignment on the axes
    - Greatly improving anti-aliasing quality for the most commonly encountered cases
    - For an optimal pattern, the rotation angle is arctan (1/2) (about 26.6°)

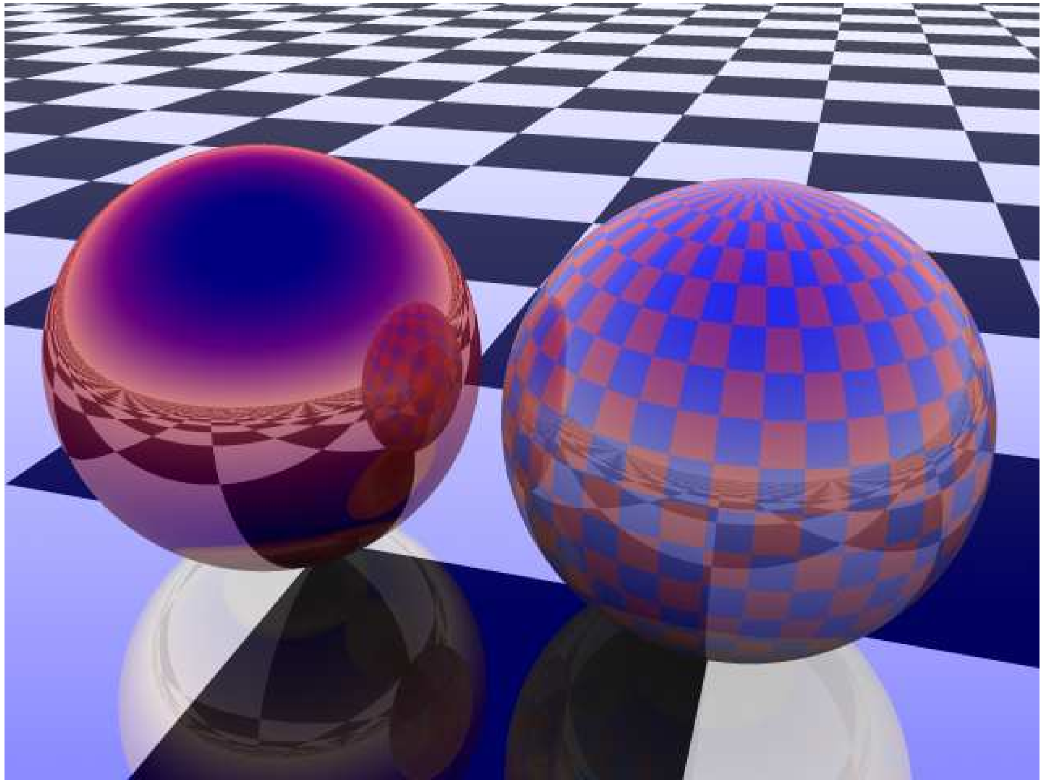


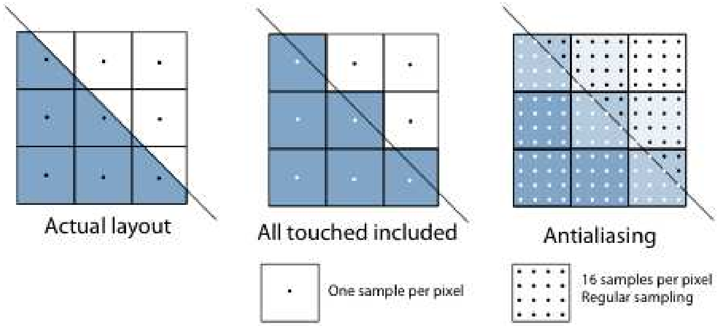
Anti-aliasing

- **Adaptive super-sampling**
  - Super-sampling is computationally expensive
  - Usually many more samples are required in order to produce good-quality image
- **Adaptivity**
  - Only pixels at the edges of objects are super-sampled with enough samples
  - Initially only a few samples are taken within each pixel
  - If these values are very similar, only these samples are used to determine the intensity (color)
  - If not, more are used

Anti-aliasing

- **Anti-aliased ray tracing image**
  - Making the edges a little bit blurry with proper blurriness





Next lecture: Efficient Ray-geometry intersection