CS680:
Ray Tracing and Radiosity

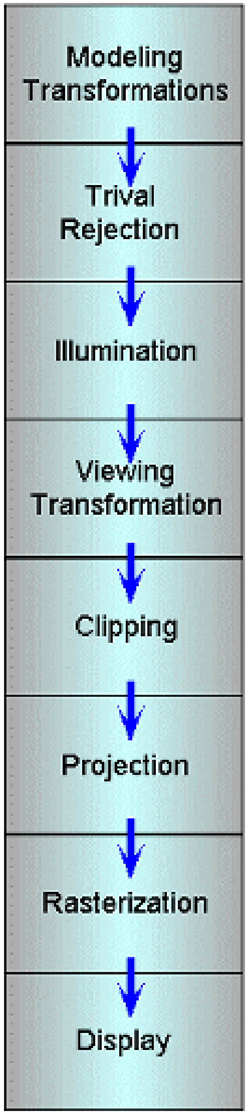
Sung-Eui Yoon
(윤성의)

Course URL:
http://jupiter.kaist.ac.kr/~sungeui/SGA/

Topics Today

- Classic ray tracing
- Classic radiosity

The Classic Rendering Pipeline

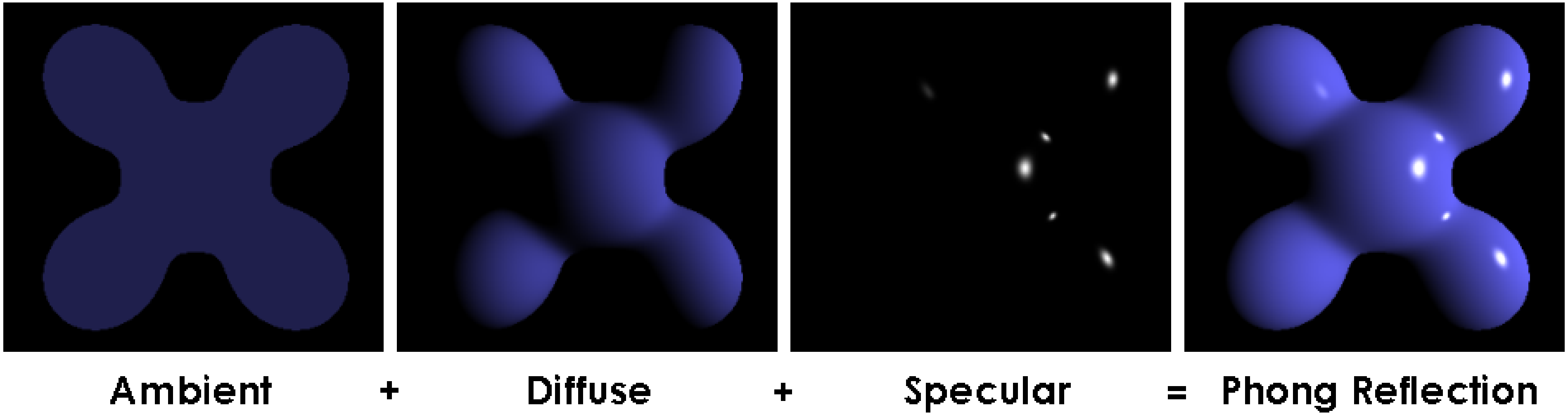


- **Object primitives defined by vertices fed in at the top**
- **Pixels come out in the display at the bottom**

- **Transformation**
  - **Transform objects into screen space**
- **Lighting**
  - **Local shading model**

OpenGL’s Illumination Model

$$I_r = \sum_{j=1}^{\text{numLights}} (k_a^j I_a^j + k_d^j I_d^j \max((\hat{N} \bullet \hat{L}_j),0) + k_s^j I_s^j \max((\hat{V} \bullet \hat{R})^{n_s},0))$$

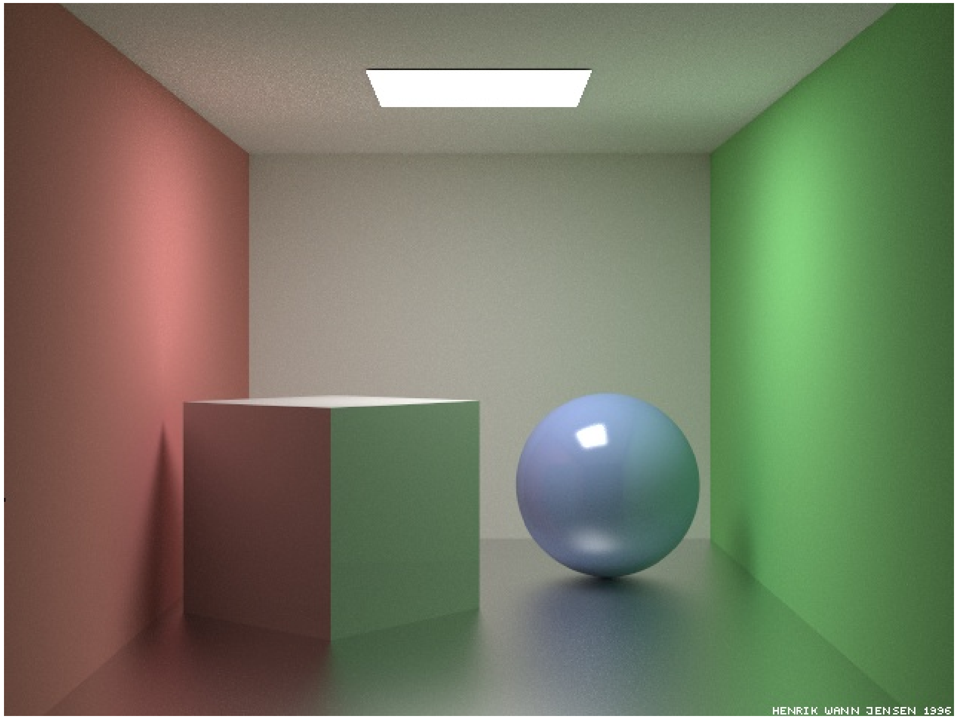


From Wikipedia

- **With programmable GPUs**
  - **Can have arbitrary shading models**

But what about other visual cues?

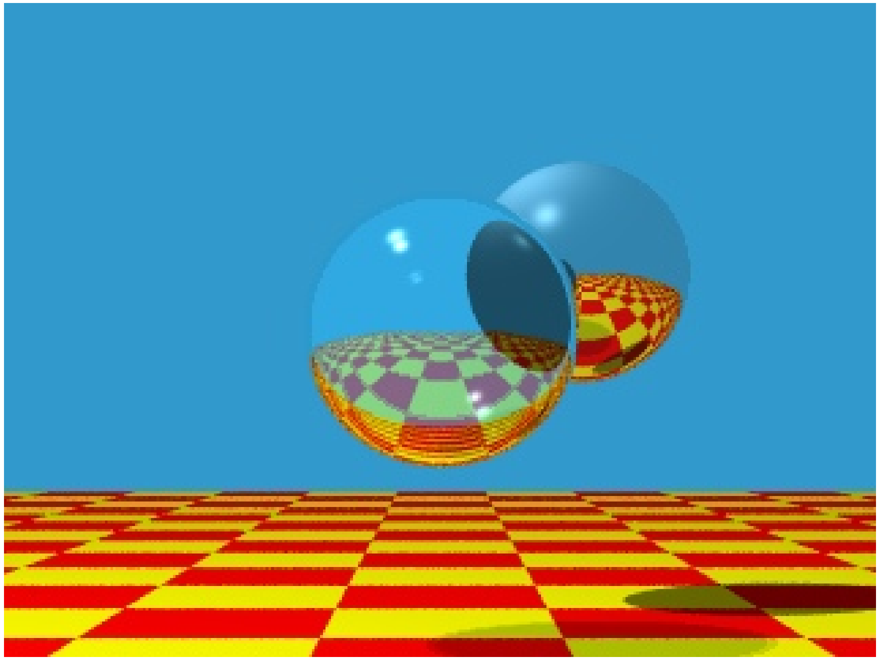
- Lighting
  - Shadows
  - Shading: glossy, transparency
- Color bleeding, etc



from Henrik's homepage

Recursive Ray Casting

- **Gained popularity in when Turner Whitted (1980) recognized that *recursive* ray casting could be used for global illumination effects**

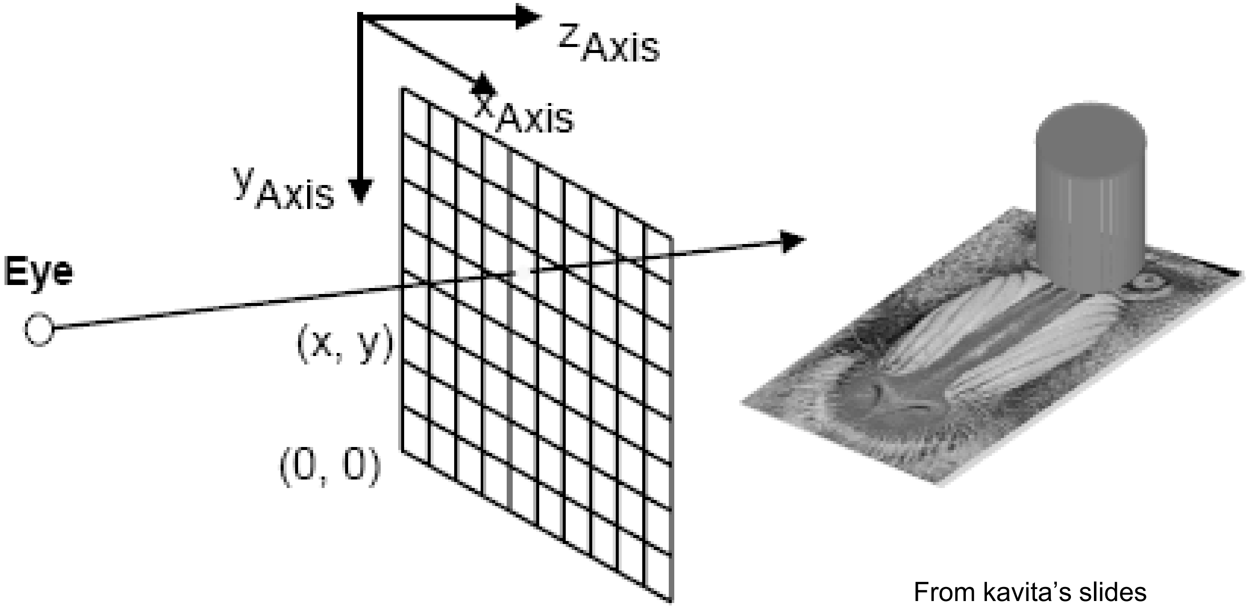


Ray Casting and Ray Tracing

- Trace rays from eye into scene
  - Backward ray tracing
- Ray casting used to compute visibility at the eye
- Perform ray tracing for arbitrary rays needed for shading
  - Reflections
  - Refraction and transparency
  - Shadows

Basic Algorithms

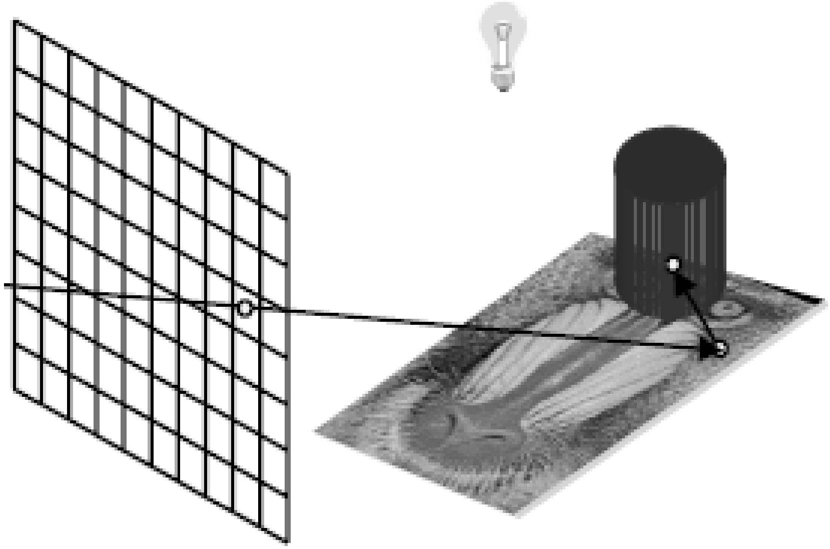
- **Rays are cast from the eye point through each pixel in the image**

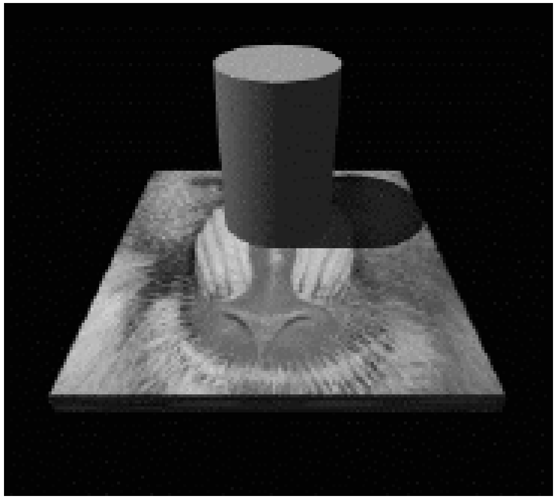


From kavita's slides

Shadows

- Cast ray from the intersection point to each light source
  - Shadow rays

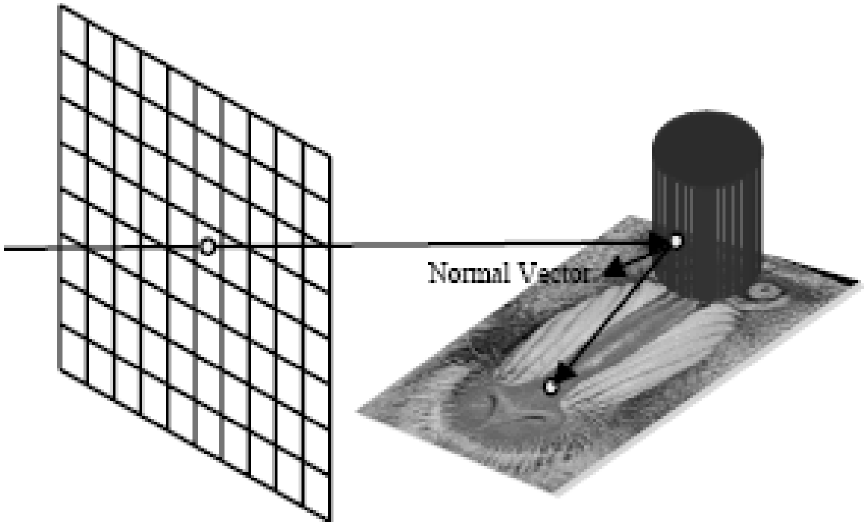


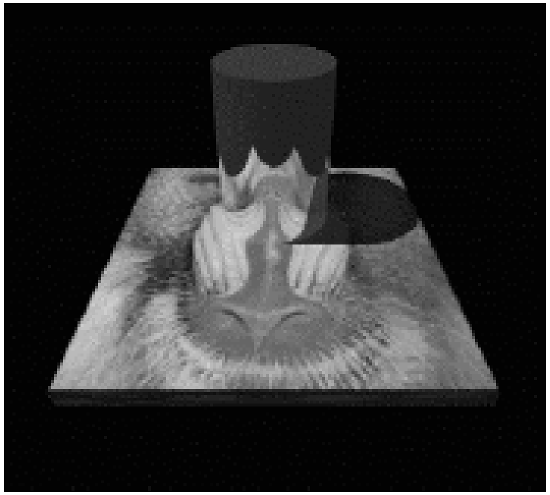


From kavita's slides

Reflections

- **If object specular, cast secondary reflected rays**

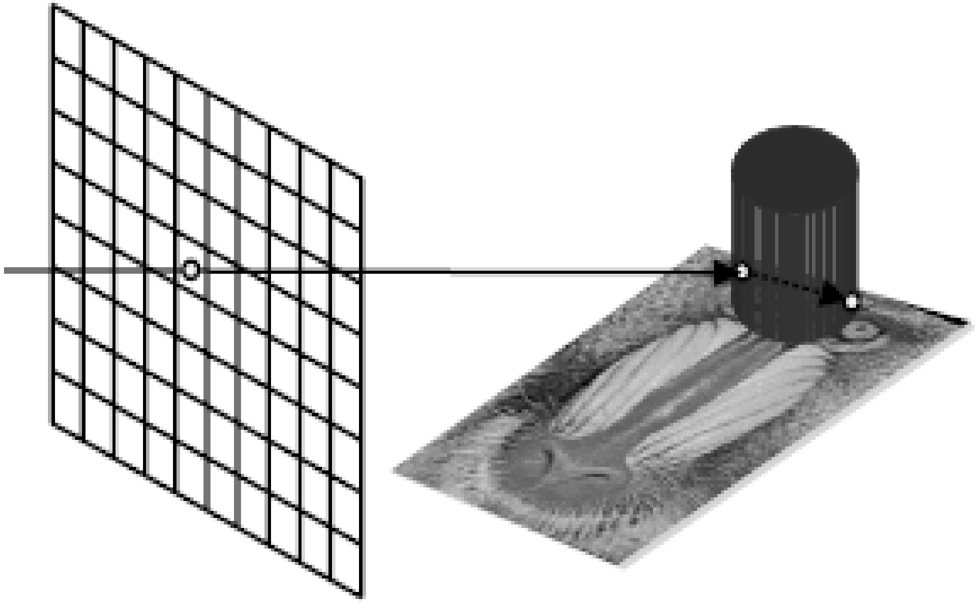


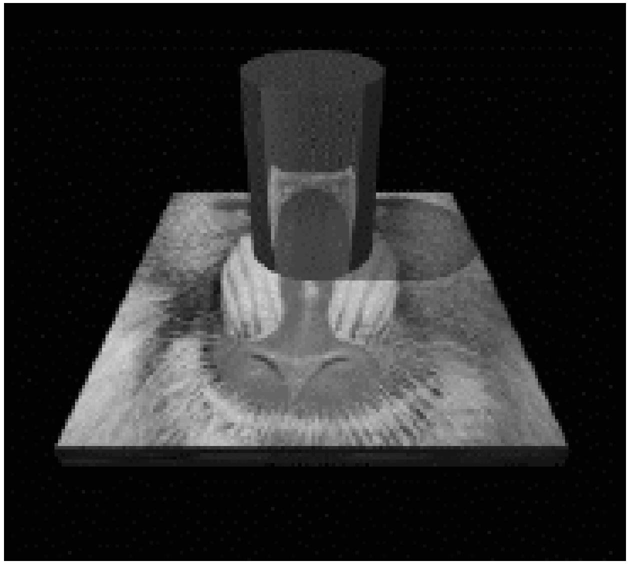


From kavita's slides

Refractions

- **If object tranparent, cast secondary refracted rays**





From kavita's slides

An Improved Illumination Model [Whitted 80]

- **Phong model**

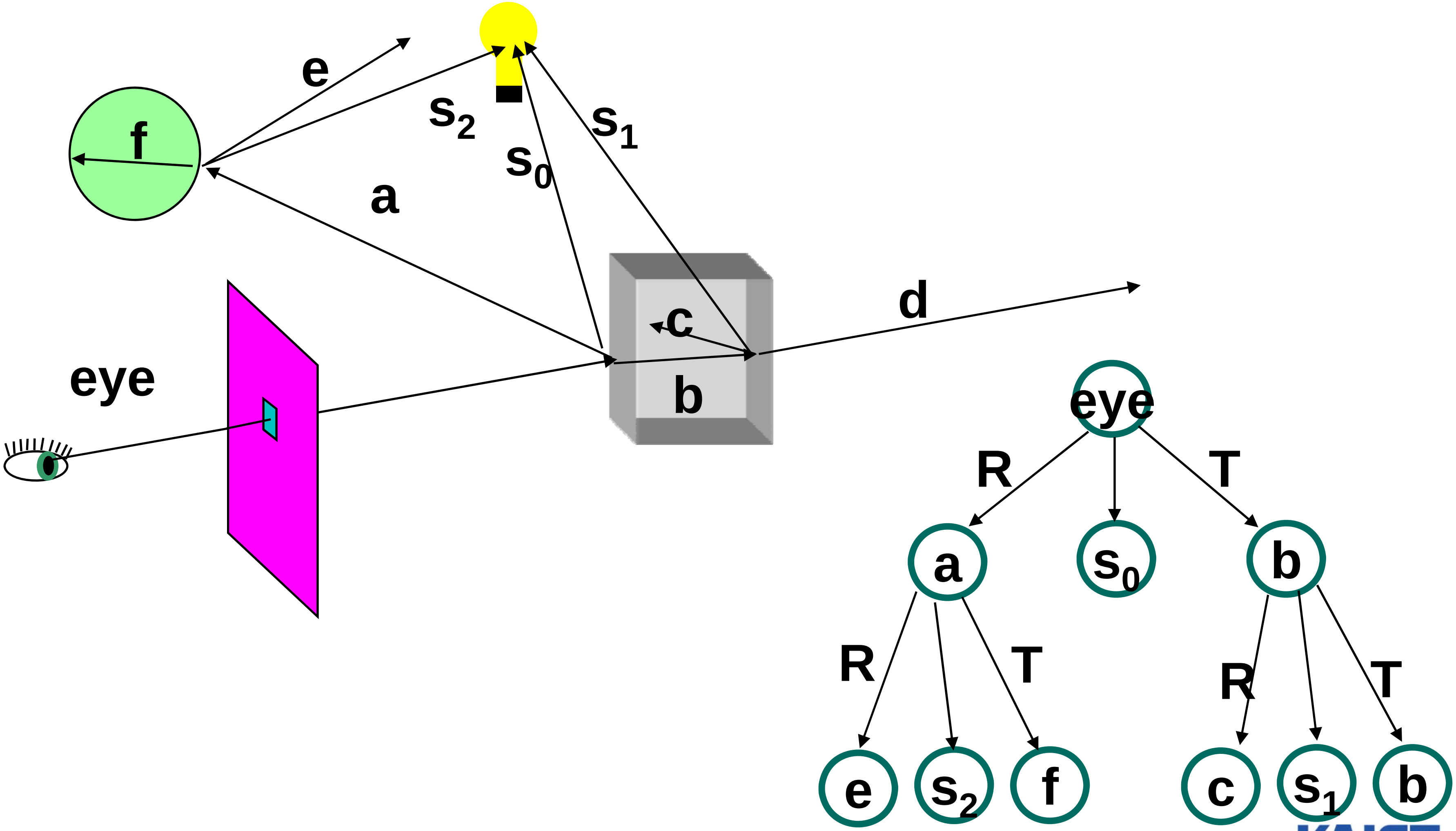
$$I_r = \sum_{j=1}^{numLights} (k_a^j I_a^j + k_d^j I_d^j (\hat{N} \bullet \hat{L}_j) + k_s^j I_s^j (\hat{V} \bullet \hat{R})^{n_s})$$

- **Whitted model**

$$I_r = \sum_{j=1}^{numLights} (k_a^j I_a^j + k_d^j I_d^j (\hat{N} \bullet \hat{L}_j)) + k_s S + k_t T$$

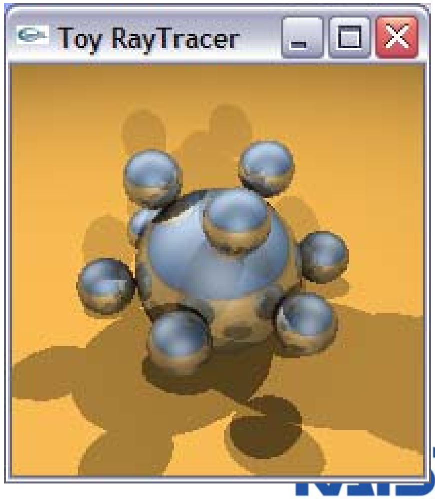
  - **S and T are intensity of light from reflection and transmission rays**
  - **Ks and Kt are specular and transmission coefficient**

Ray Tree



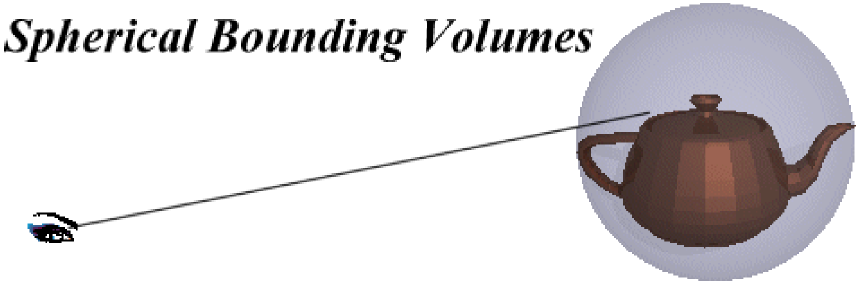
Acceleration Methods for Ray Tracing

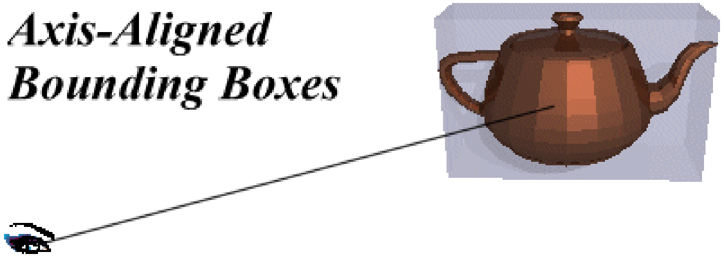
- **Rendering time for a ray tracer depends on the number of ray intersection tests per pixel**
  - **The number of pixels X the number of primitives in the scene**
- **Early efforts focused on accelerating the ray-object intersection tests**
- **More advanced methods required to make ray tracing practical**
  - **Bounding volume hierarchies**
  - **Spatial subdivision**



Bounding Volumes

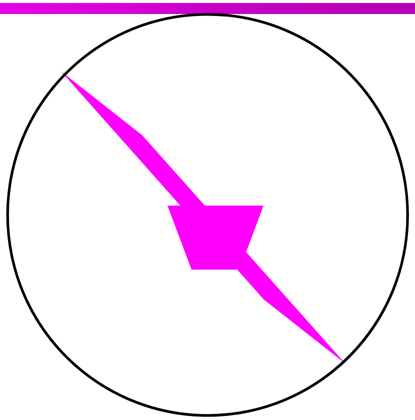
- Enclose complex objects within a simple-to-intersect objects
  - If the ray does not intersect the simple object then its contents can be ignored
  - The likelihood that it will strike the object depends on how tightly the volume surrounds the object.
- Spheres are simple, but not tight
- Axis-aligned bounding boxes often better
  - Can use nested or hierarchical bounding volumes



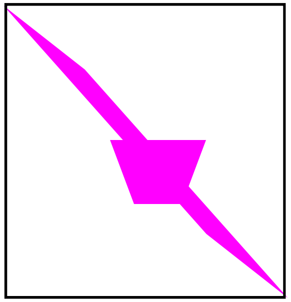


Bounding Volumes

- Sphere [Whitted80]
  - Cheap to compute
  - Cheap test
  - Potentially very bad fit

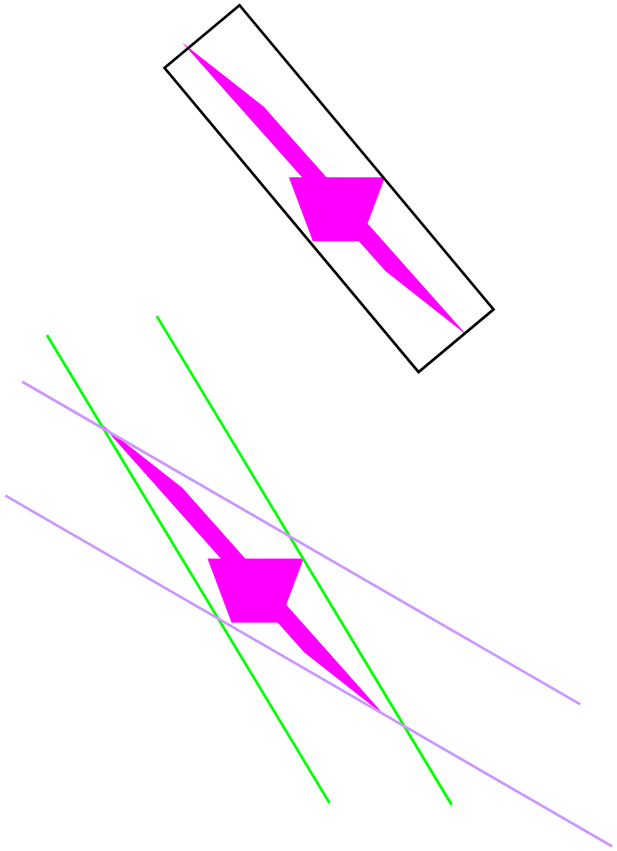


- Axis-Aligned Bounding Box
  - Very cheap to compute
  - Cheap test
  - Tighter than sphere



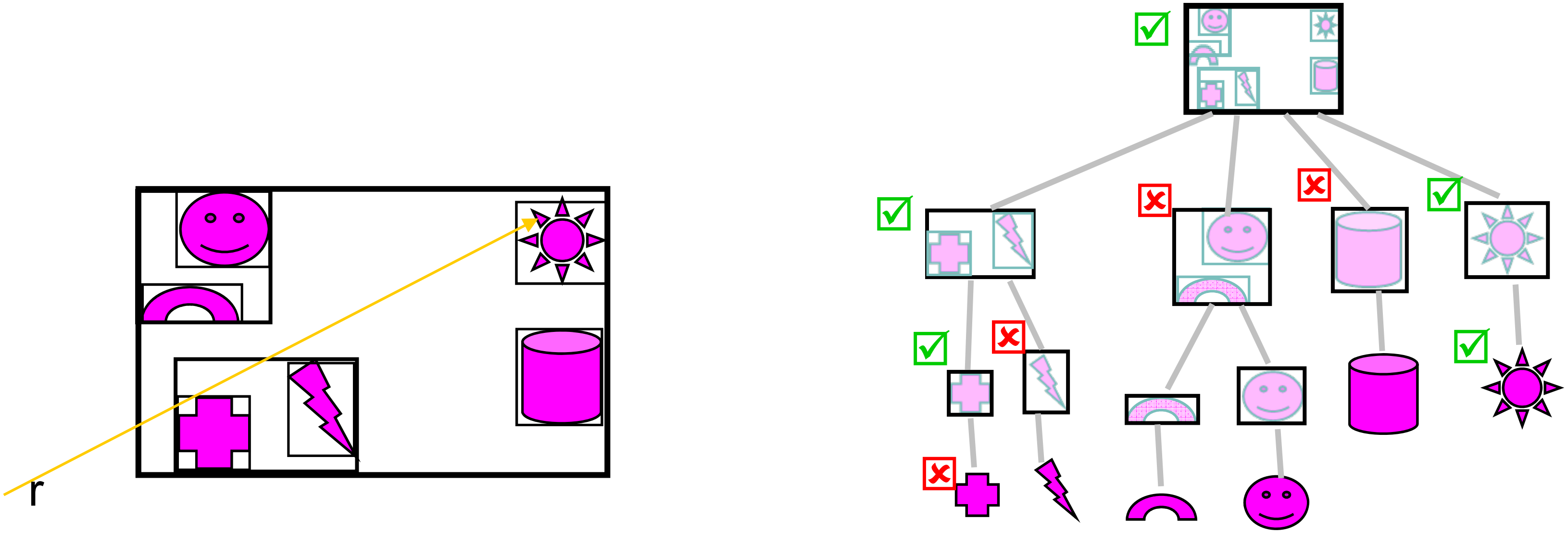
Bounding Volumes

- Oriented Bounding Box
  - Fairly cheap to compute
  - Fairly Cheap test
  - Generally fairly tight
- Slabs / K-dops
  - More expensive to compute
  - Fairly cheap test
  - Can be tighter than OBB



Hierarchical Bounding Volumes

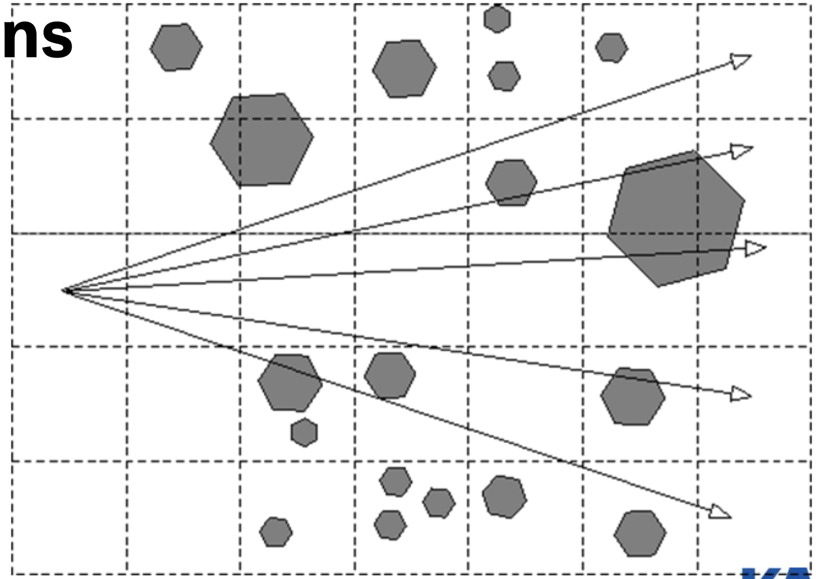
- Organize bounding volumes as a tree
- Each ray starts with the scene BV and traverses down through the hierarchy



Spatial Subdivision

Idea: Divide space in to subregions

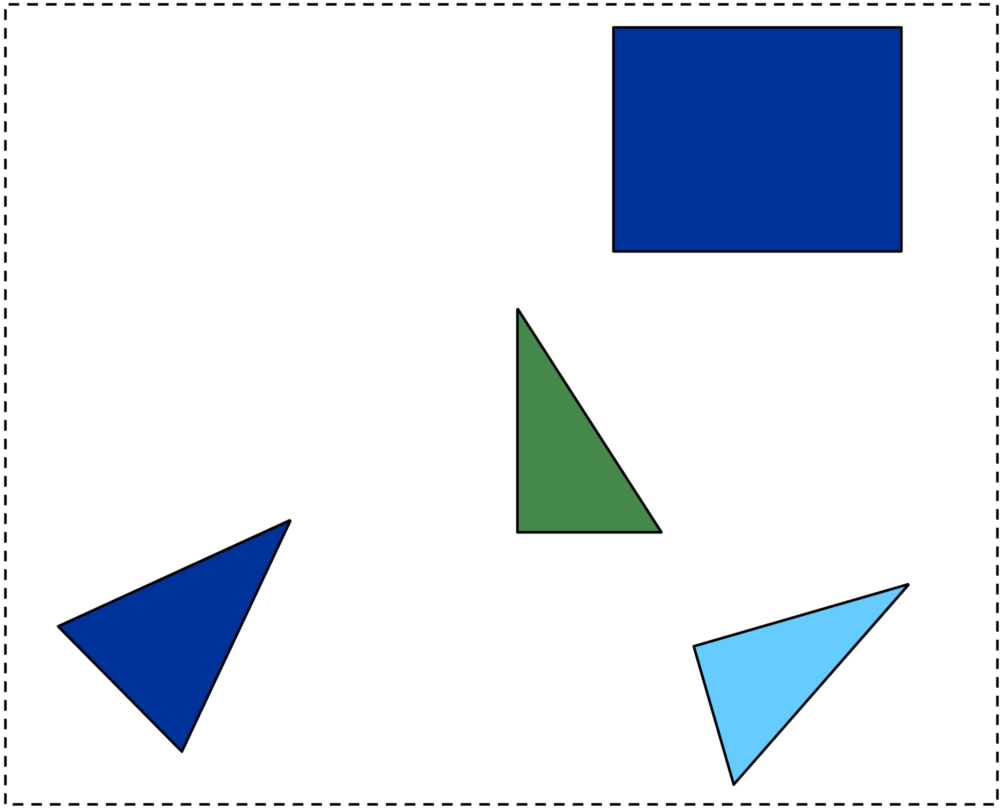
- Place objects within a subregion into a list
- Only traverse the lists of subregions that the ray passes through
- "Mailboxing" used to avoid multiple test with objects in multiple regions
- Many types
  - Regular grid
  - Octree
  - BSP tree
  - kd-tree



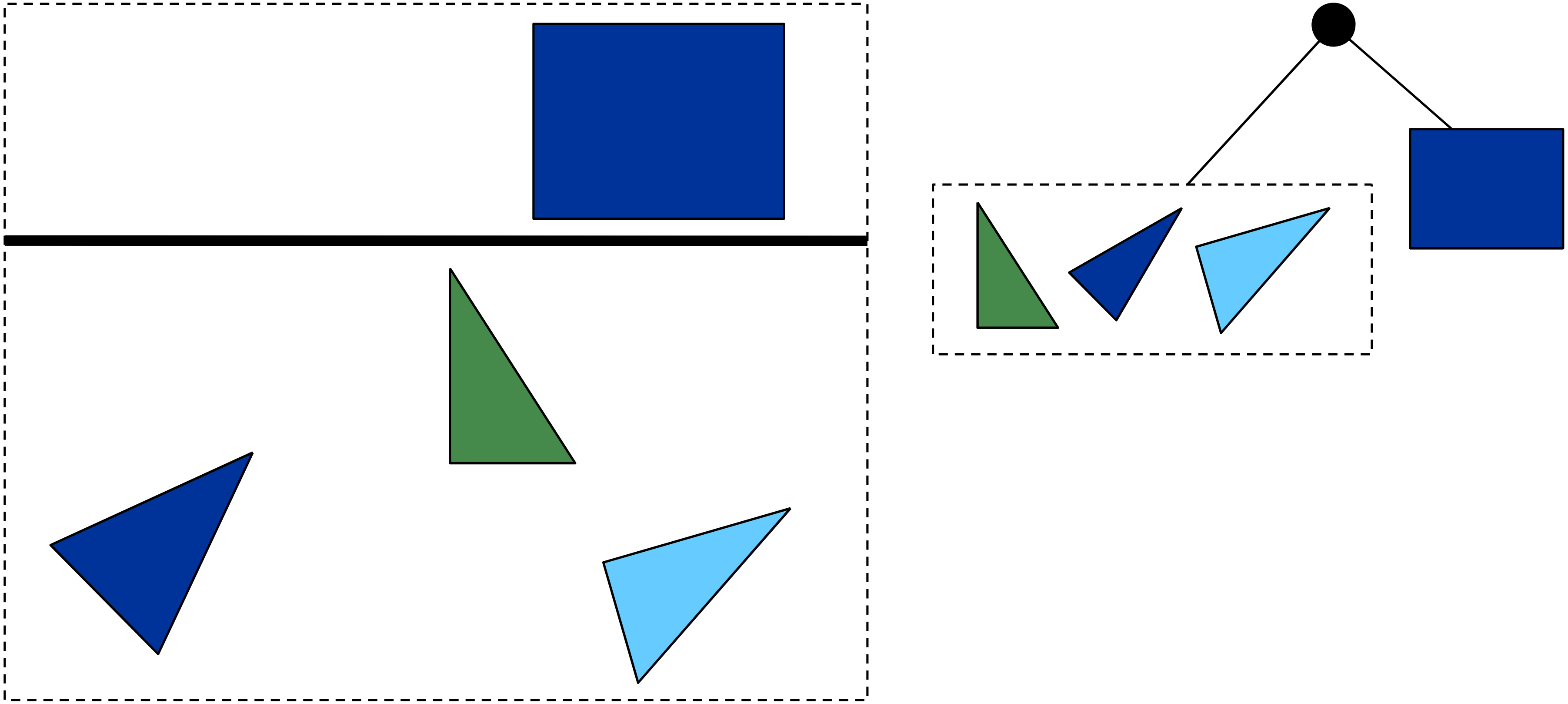
Overview of kd-Trees

- Binary spatial subdivision (special case of BSP tree)
- Split planes aligned on main axis
- Inner nodes: subdivision planes
- Leaf nodes: triangle(s)

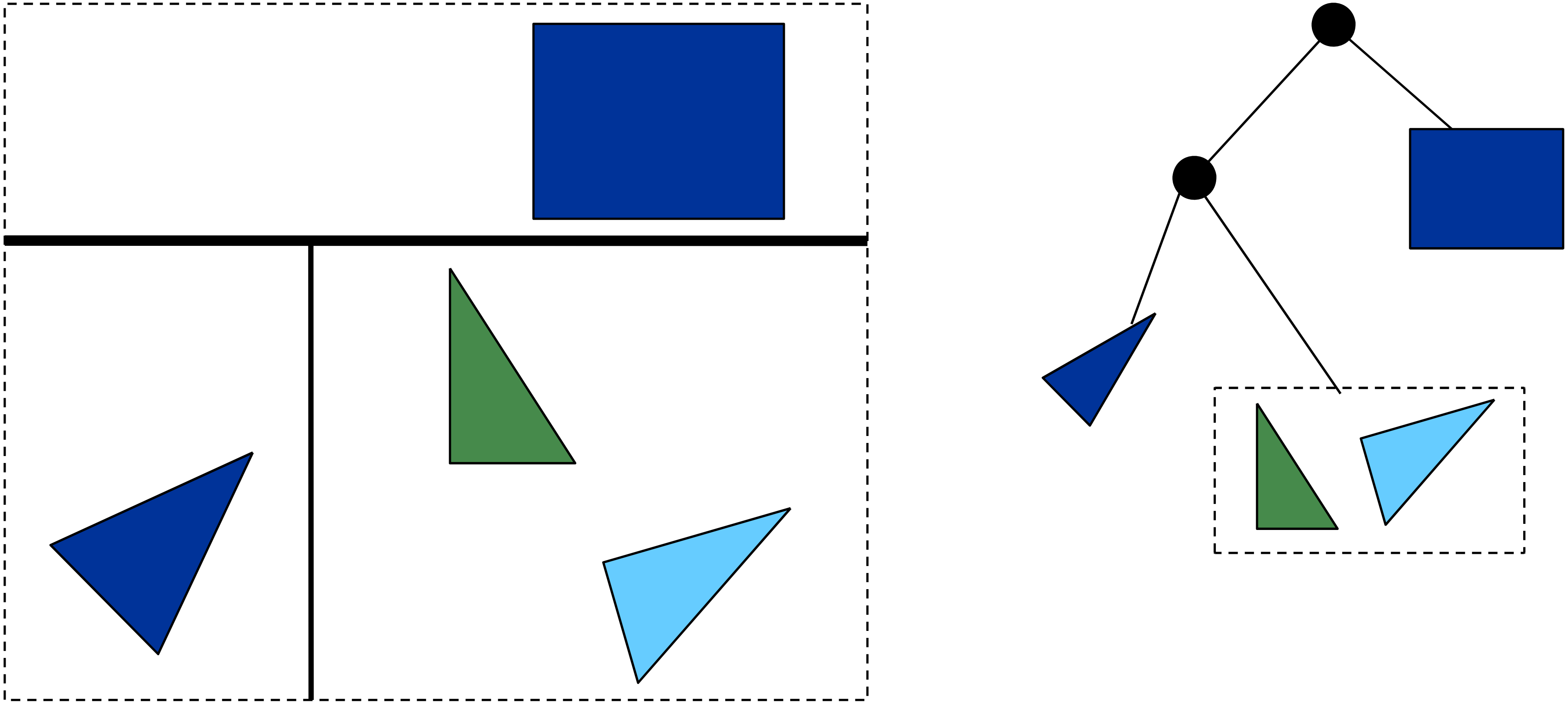
Example



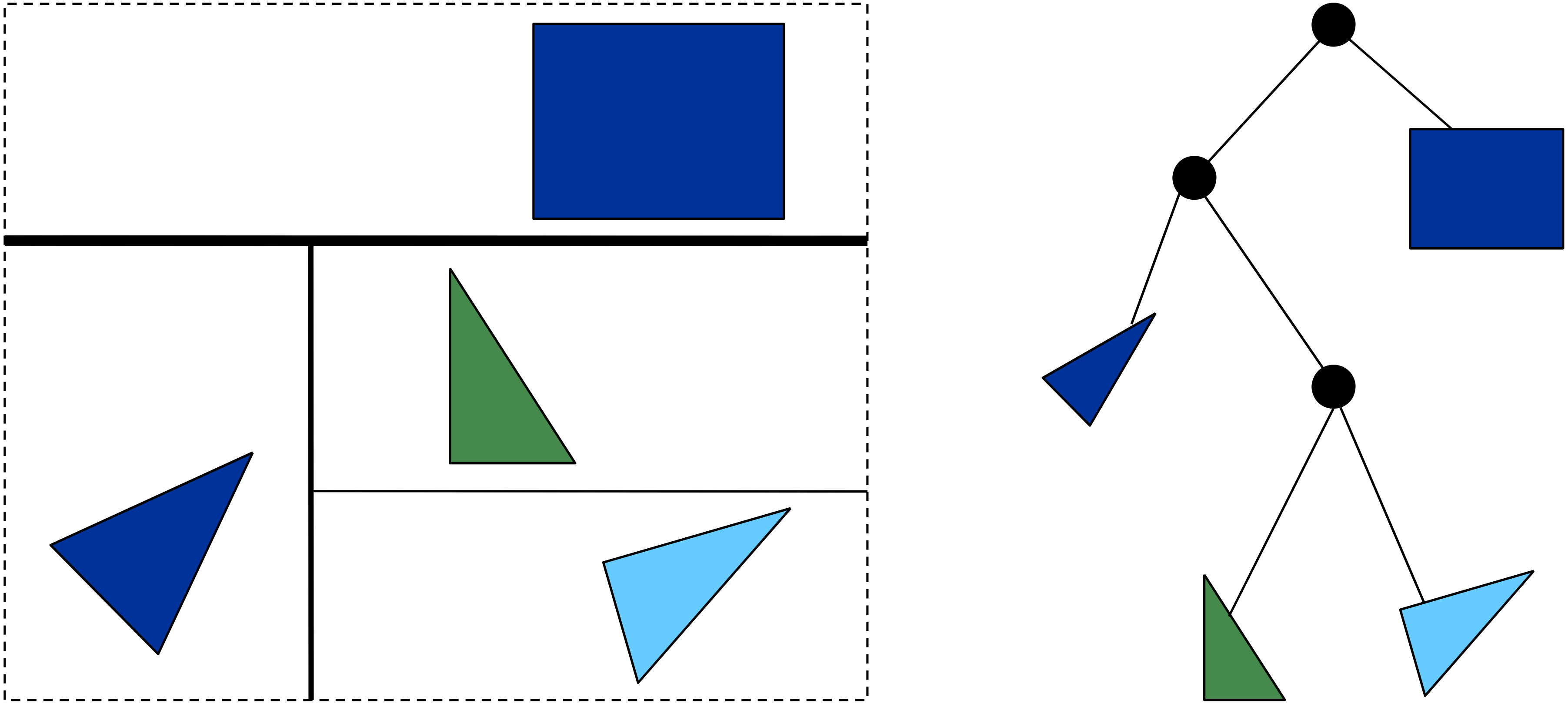
Example



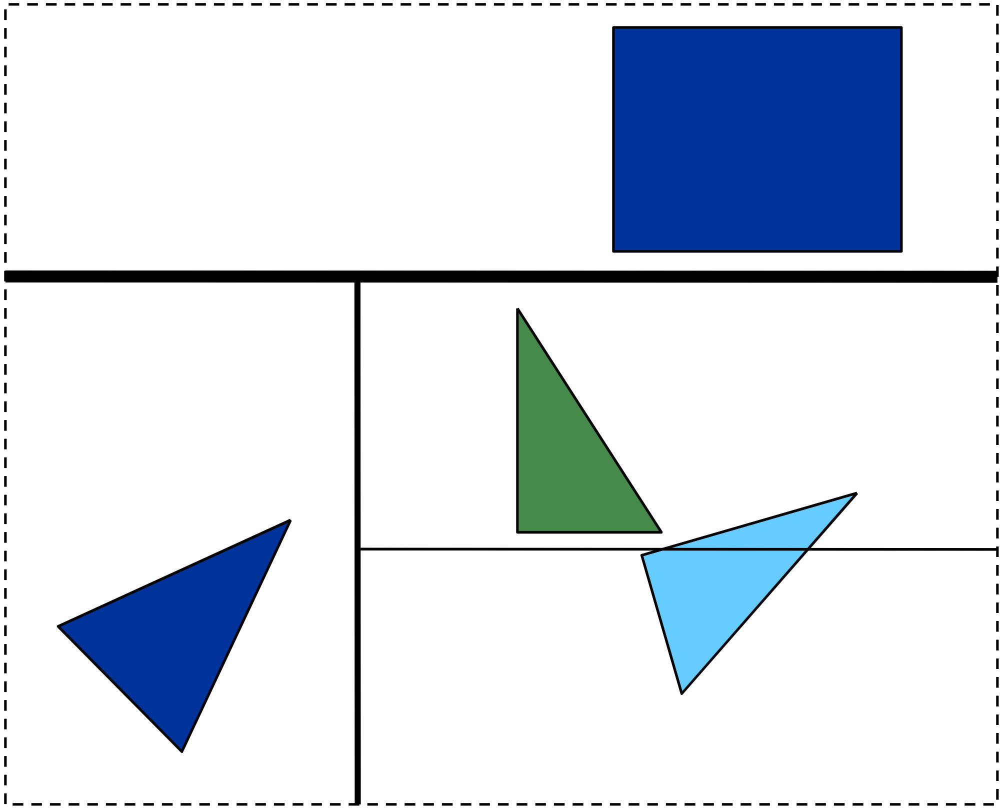
Example



Example

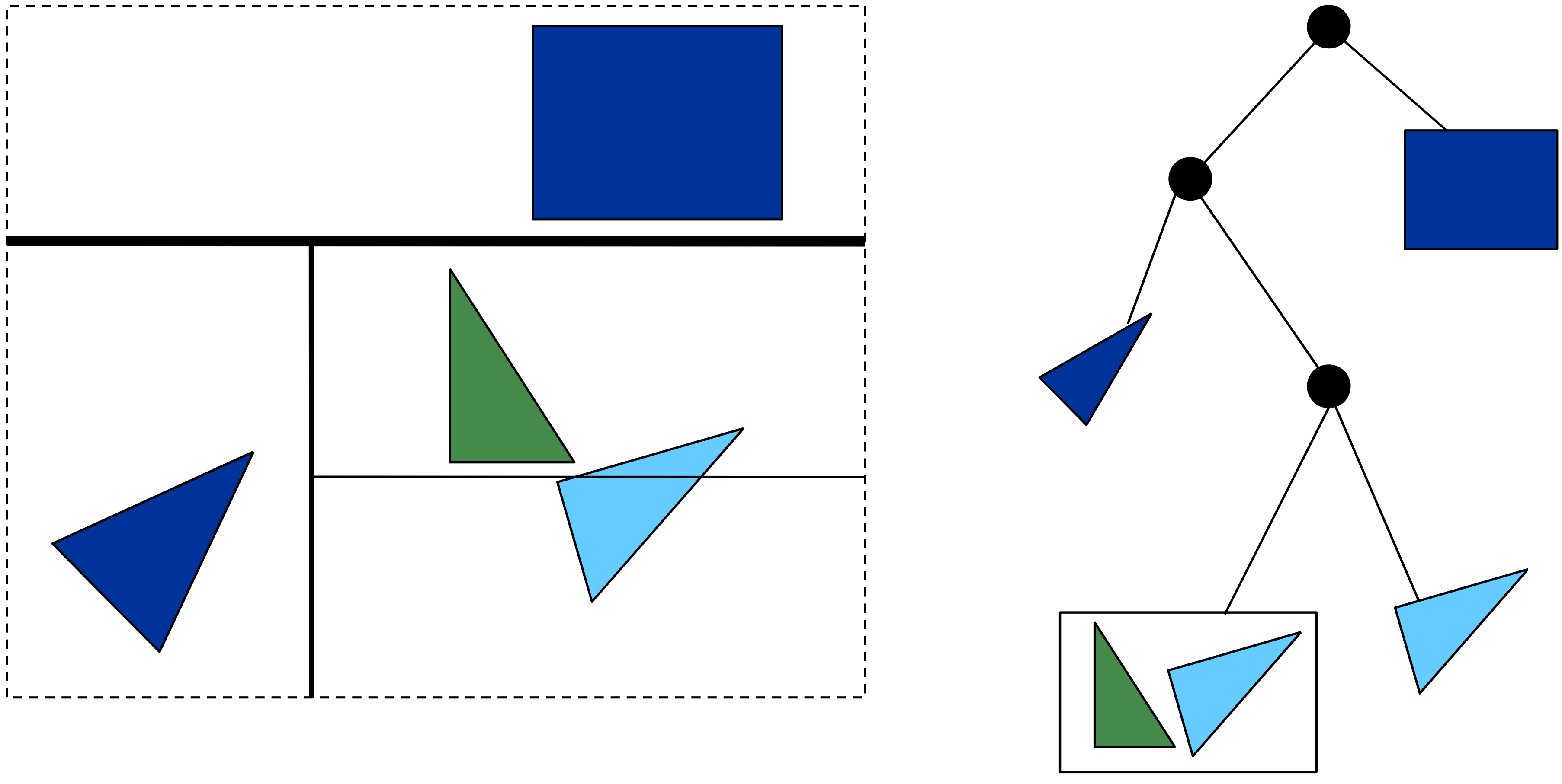


Example



What about triangles overlapping the split?

Example

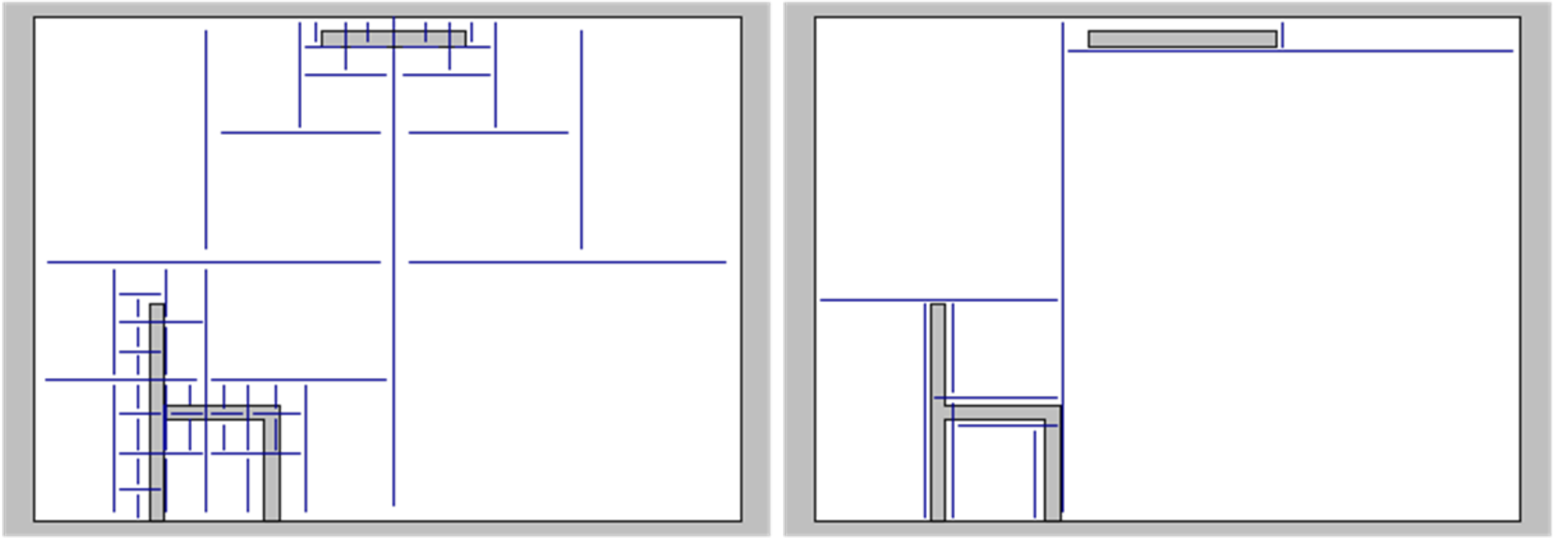


Split Planes

- How to select axis & split plane?
  - Largest dimension, subdivide in middle
  - More advanced:
    - Surface area heuristic

- Makes large difference
  - 50%-100% higher *overall* speed

Median vs. SAH



(from [Wald04])

Ray Tracing with kd-tree

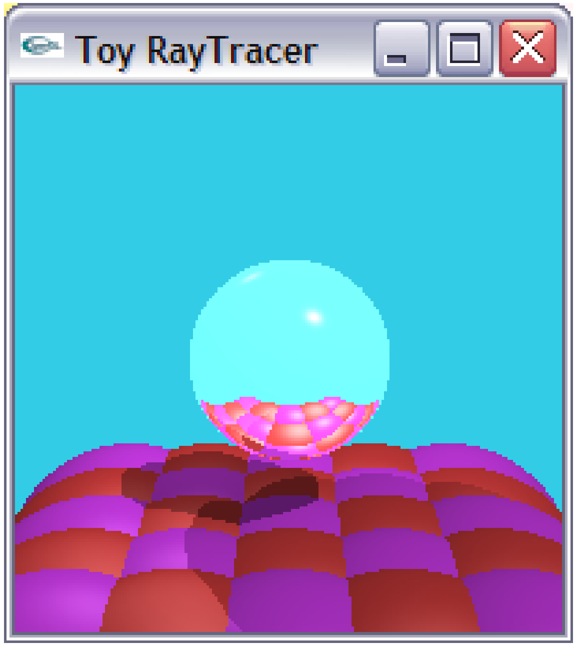
- Goal: find closest hit with scene
- Traverse tree front to back (starting from root)
- At each node:
    - If leaf: intersect with triangles
    - If inner: traverse deeper

Other Optimizations

- Shadow cache
- Adaptive recursion depth control
- Lazy geometry loading/creation

Classic Ray Tracing

- Gathering approach
  - From lights, reflected, and refracted directions
- Pros of ray tracing
  - Simple and improved realism over the rendering pipeline



- Cons:
  - Simple light model, material, and light propagation
  - Not a complete solution
  - Hard to accelerate with special-purpose H/W

History

- Problems with classic ray tracing
  - Not realistic
  - View-dependent

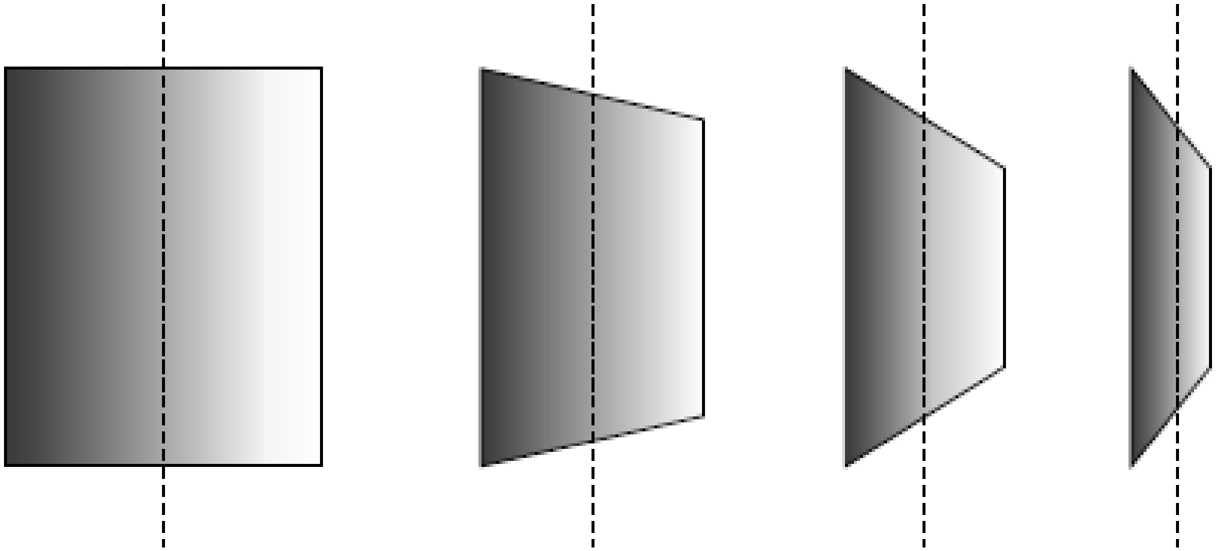
- Radiosity (1984)
  - Global illumination in diffuse scenes

- Monte Carlo ray tracing (1986)
  - Global illumination for any environment

Radiosity

- Physically based method for diffuse environments
  - Support diffuse interactions, color bleeding, indirect lighting and penumbra
  - Account for very high percentage of total energy transfer
  - Finite element method

Key Idea #1: Diffuse Only

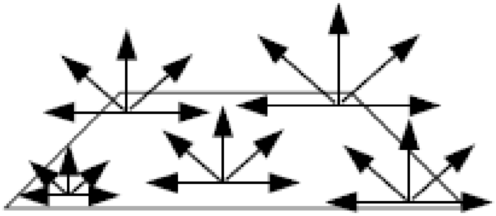


From kavita’s slides

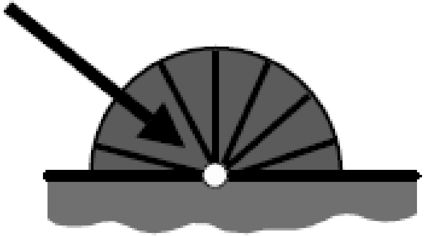
- Radiance independent of direction
  - Surface looks the same from any viewpoint
  - No specular reflection

Diffuse Surfaces

- **Diffuse emitter**
  - $L(x \to \Theta)$ **= constant over** Θ



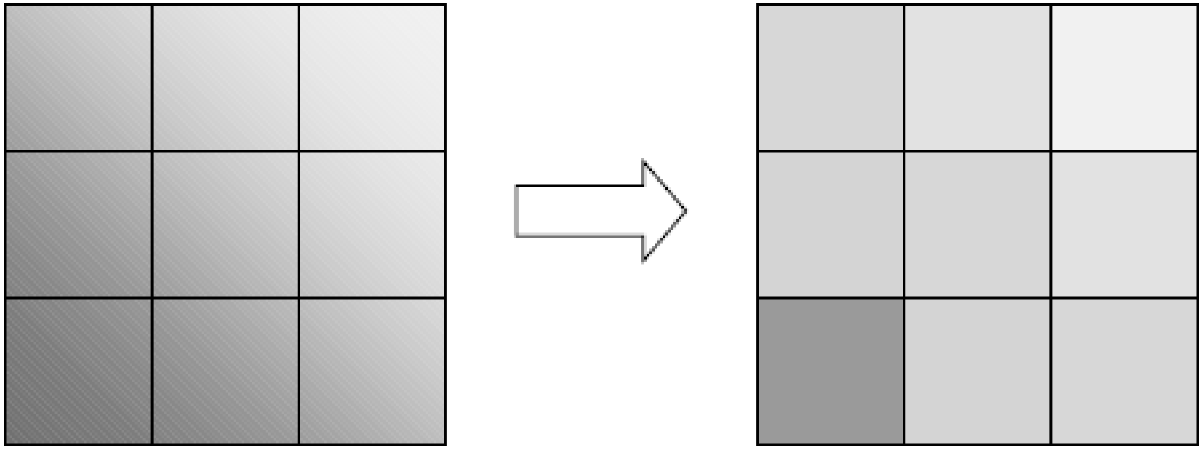
- **Diffuse reflector**
  - **Constant reflectivity**



From kavita's slides

Key Idea #2: Constant Polygons

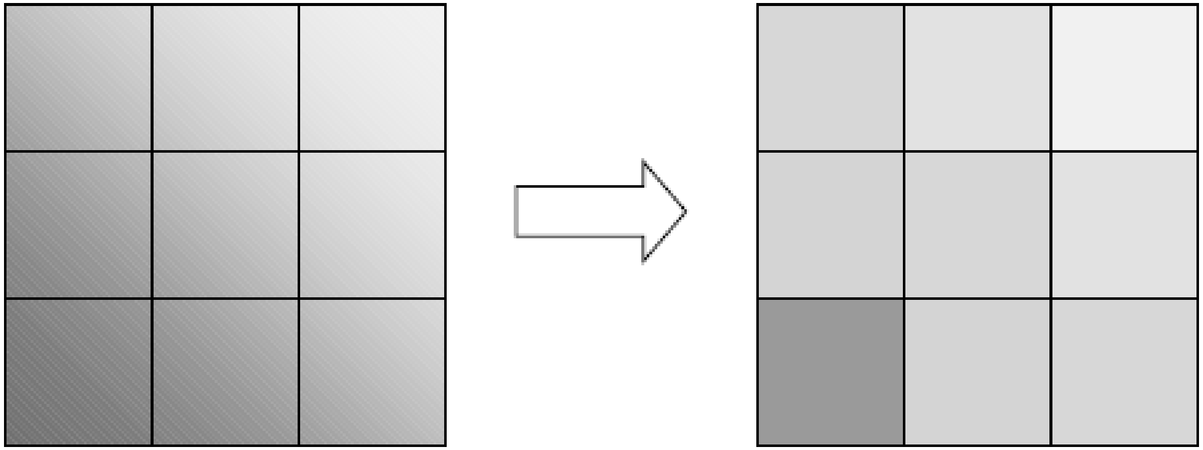
- Radiosity is an approximation
  - Due to discretization of scene into patches



From kavita's slides

  - Subdivide scene into small polygons

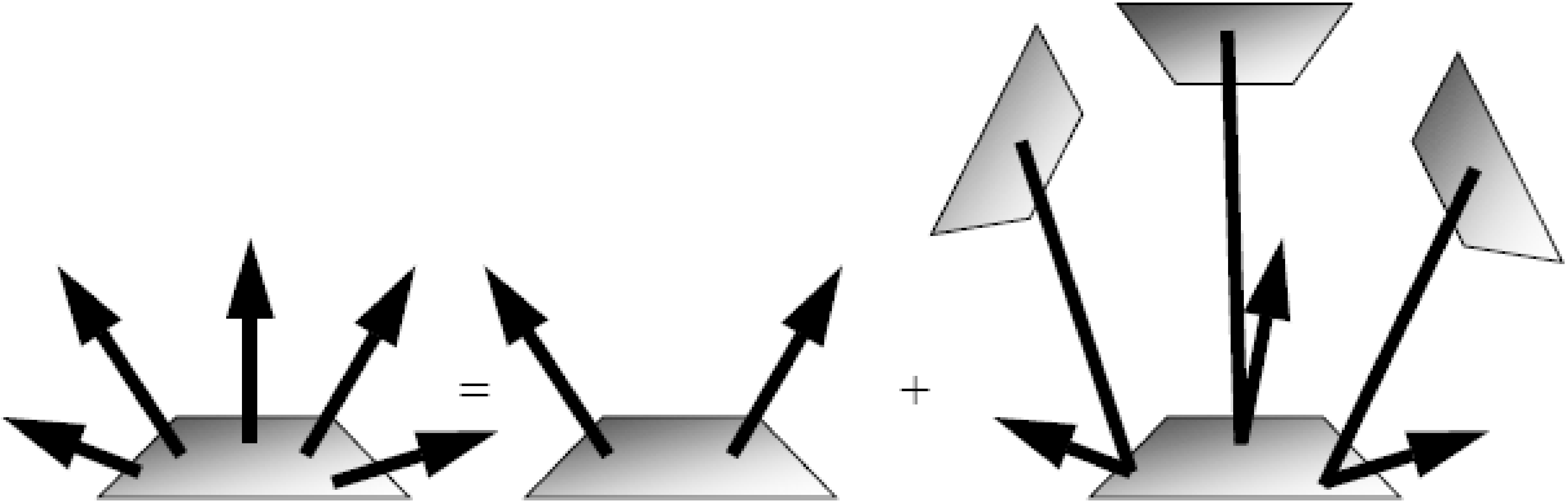
Constant Radiance Approximation



From kavita’s slides

- Radiance is constant over a surface element
  - L(x) = constant over x
- Surface element i: L(x) = L_i

Radiosity Equation



Emitted radiosity = self-emitted radiosity + received & reflected radiosity

$$Radiosity_i = Radiosity_{self,i} + \sum_{j=1}^{N} a_{j \to i} Radiosity_j$$

Radiosity Equation

- Radiosity equation for each polygon i

$$Radiosity_1 = Radiosity_{self,1} + \sum_{j=1}^{N} a_{j\to 1} Radiosity_j$$

$$Radiosity_2 = Radiosity_{self,2} + \sum_{j=1}^{N} a_{j\to 2} Radiosity_j$$

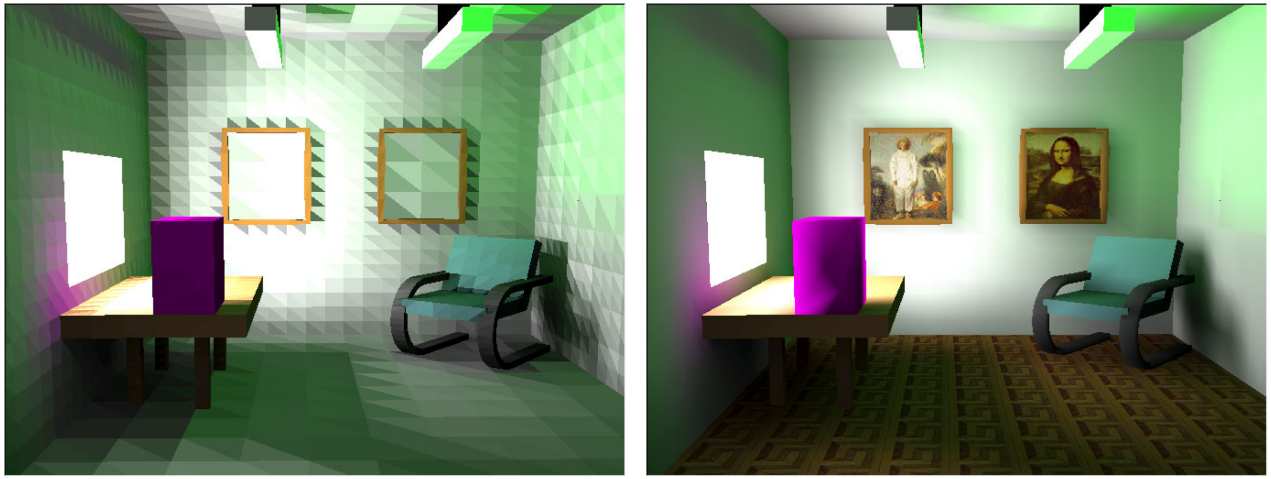
$$\ldots$$

$$Radiosity_N = Radiosity_{self,N} + \sum_{j=1}^{N} a_{j\to N} Radiosity_j$$

- N equations; N unknown variables

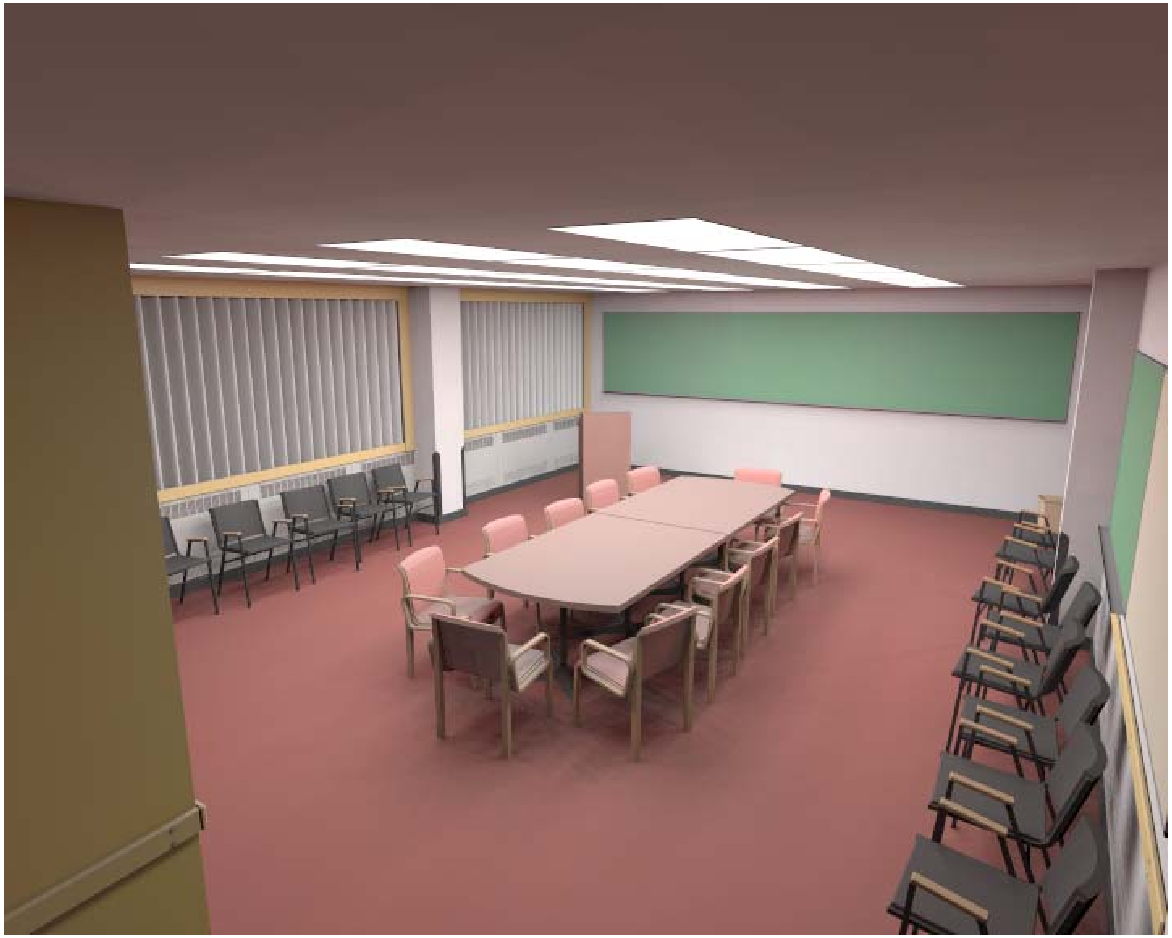
Radiosity Algorithm

- Subdivide the scene in small polygon
- Compute a constant illumination value for each polygon
- Choose a viewpoint and display the visible polygon
  - Keep doing this process



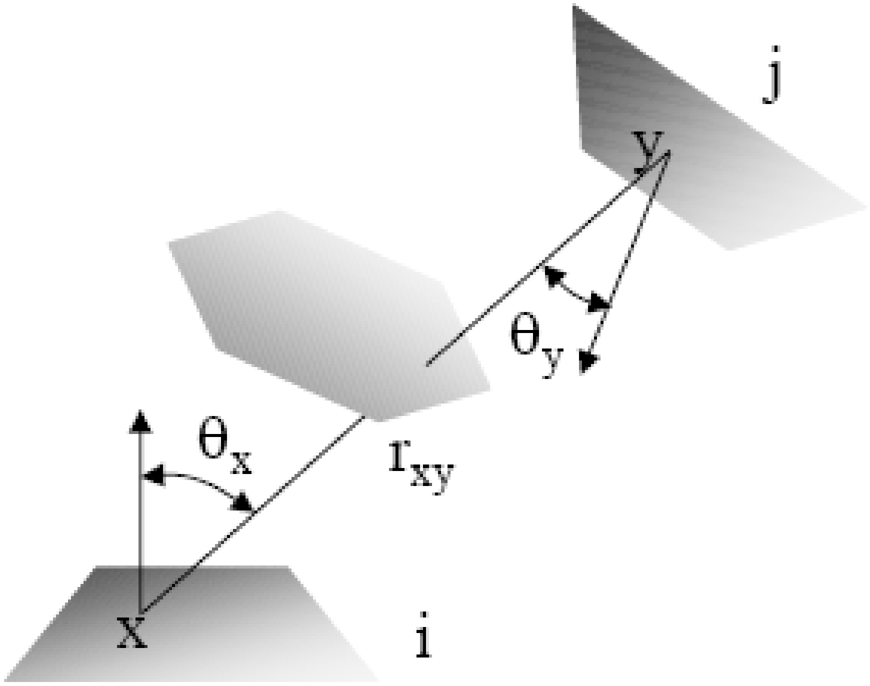
From Donald Fong's slides

Radiosity Result



Compute Form Factors

$$F(j \rightarrow i) = \frac{1}{A_j} \int_{A_i} \int_{A_j} \frac{\cos\theta_x \cdot \cos\theta_y}{\pi \cdot r_{xy}^2} \cdot V(x,y) \cdot dA_y \cdot dA_x$$



Radiosity Equation

- Radiosity for each polygon i

$$\forall i : B_i = B_{e,i} + \rho_i \sum_{j=1}^{N} B_j F(i \rightarrow j)$$

- Linear system
  - B_i : radiosity of patch i (unknown)
  - $B_{e,i}$: emission of patch i (known)
  - ρ_I : reflectivity of patch i (known)
  - $F(i \rightarrow j)$: form-factor (coefficients of matrix)

Linear System of Radiosity Equations

$$\underbrace{\begin{bmatrix} 1-\rho_1 F_{1\to 1} & -\rho_1 F_{1\to 2} & \cdots & -\rho_1 F_{1\to n} \\ -\rho_2 F_{2\to 1} & 1-\rho_2 F_{2\to 2} & \cdots & -\rho_2 F_{2\to n} \\ \cdots & \cdots & \cdots & \cdots \\ -\rho_n F_{n\to 1} & -\rho_n F_{n\to 2} & \cdots & 1-\rho_n F_{n\to n} \end{bmatrix}}_{\text{known}} \underbrace{\begin{bmatrix} B_1 \\ B_2 \\ \cdots \\ B_n \end{bmatrix}}_{\text{unknown}} = \underbrace{\begin{bmatrix} B_{e,1} \\ B_{e,2} \\ \cdots \\ B_{e,n} \end{bmatrix}}_{\text{known}}$$

How to Solve Linear System

- Matrix inversion
  - Takes O(n^3)

- Gather methods
  - Jacobi iteration
  - Gauss-Seidel

- Shooting
  - Southwell iteration

Iterative Approaches

- Jacobi iteration
  - Start with initial guess for energy distribution (light sources)
  - Update radiosity of all patches based on the previous guess

$$B_i = B_{e,i} + \rho_i \sum_{j=1}^{N} B_j F(i \to j)$$

new value

old values

  - Repeat until converged
- Guass-Seidel iteration
  - New values used immediately

Hybrid and Multipass Methods

- Ray tracing
  - Good for specular and refractive indirect illumination
  - View-dependent
- Radiosity
  - Good for diffuse
  - Allows interactive rendering
  - Does not scale well for massive models
- Hybrid methods
  - Combine both of them in a way

Some of Topic Lists

- Ray tracing
- Radiosity
- Rendering equations
- Monte Carlo method
- Levels-of-detail or multi-resolution techniques
- Many light problems
- Coherent ray tracing
- Shadow maps
- Dynamic and massive models
- Precomputed radiance transfer
- Real-time rendering
- Irradiance caching
- Sampling and reconstruction
- Data compression
- Parallel computation
- Realistic rendering

Next Time

- Radiometry