CS580:

Ray Tracing

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/GCG/

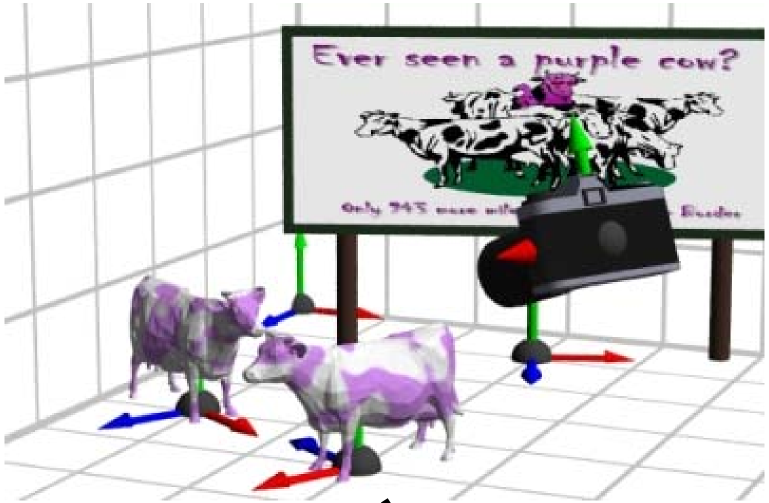
KAIST

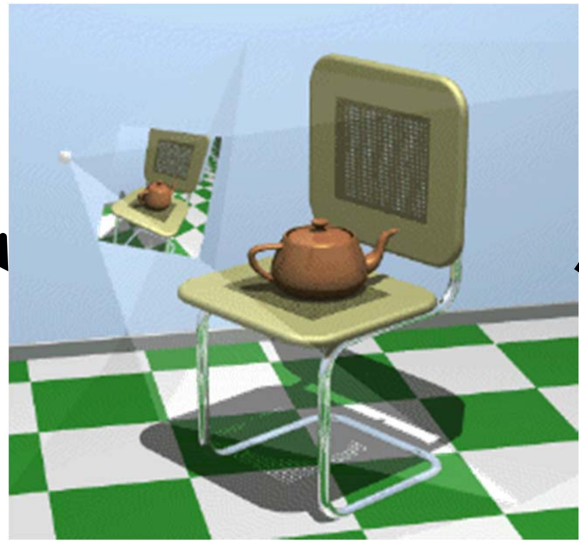
Class Objectives

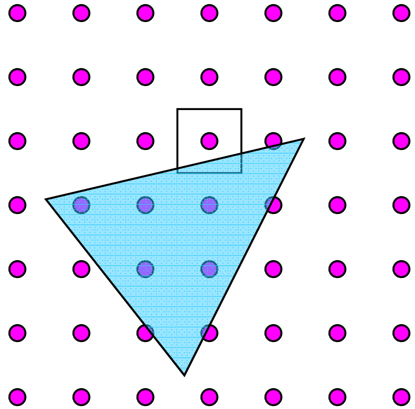
- Understand a basic ray tracing
- Know the Phong illumination model
- Implement its acceleration data structure and know how to use it

The Classic Rendering Pipeline

- Modeling Transformations
- Trival Rejection
- Illumination
- Viewing Transformation
- Clipping
- Projection
- Rasterization
- Display





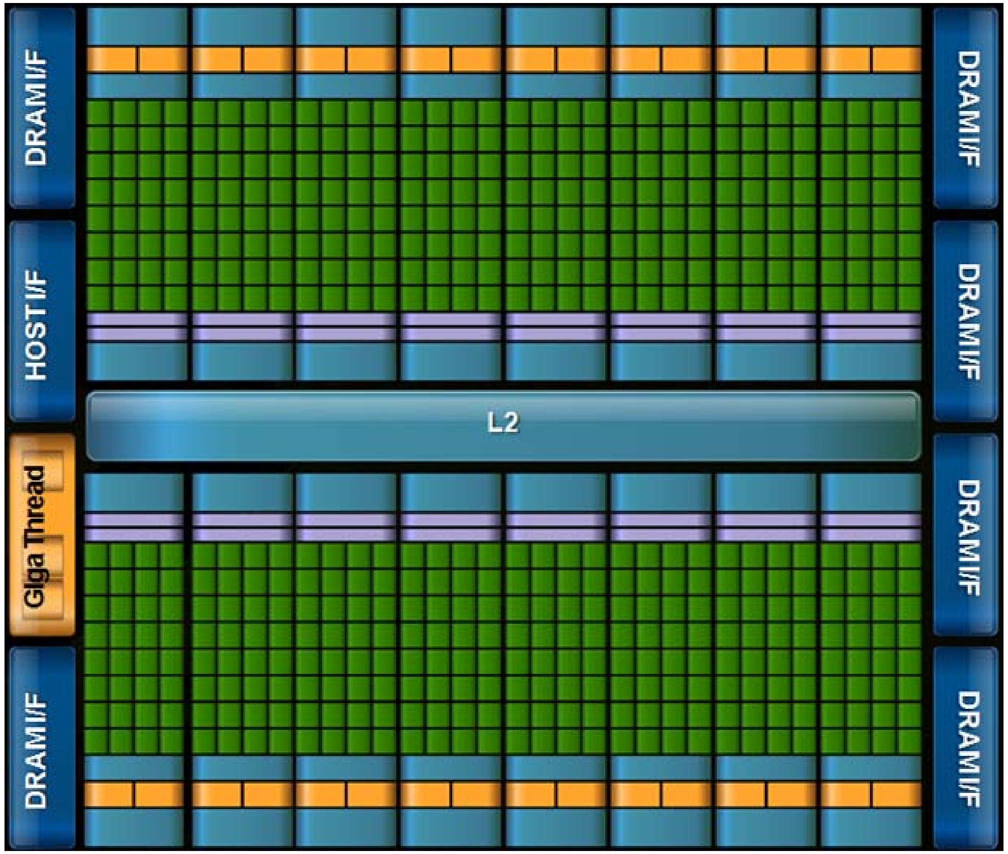


Why we are using rasterization?

- Efficiency
- Reasonably quality

Fermi GPU Architecture

16 SM (streaming processors)



512 CUDA cores

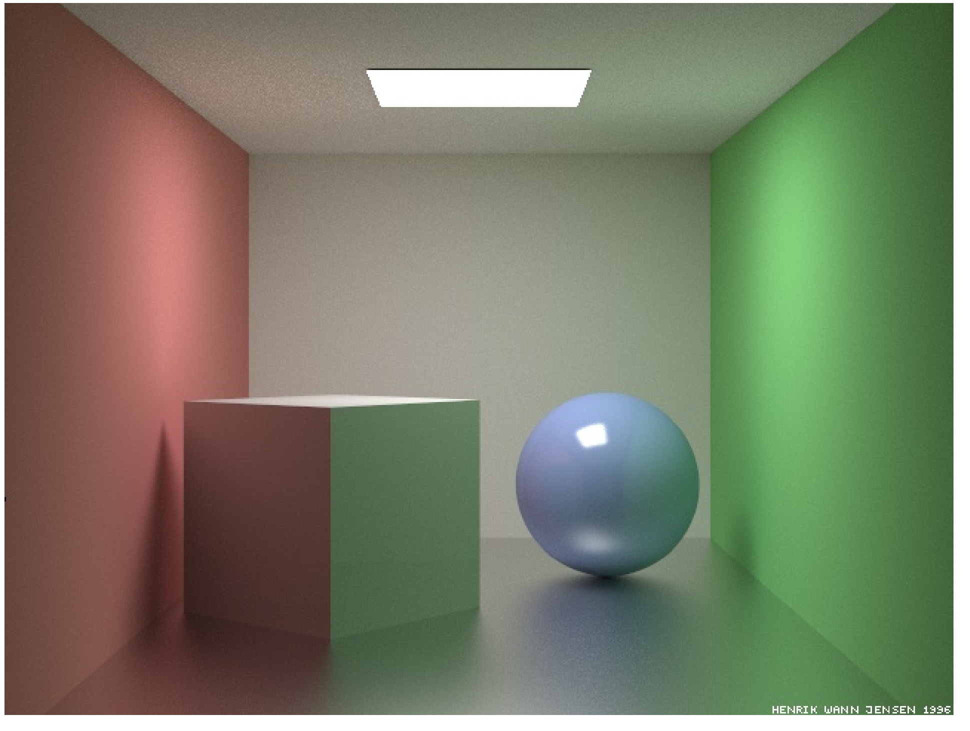
Memory interfaces

Where Rasterization Is

From Battlefield: Bad Company, EA Digital Illusions
CE AB

But what about other visual cues?

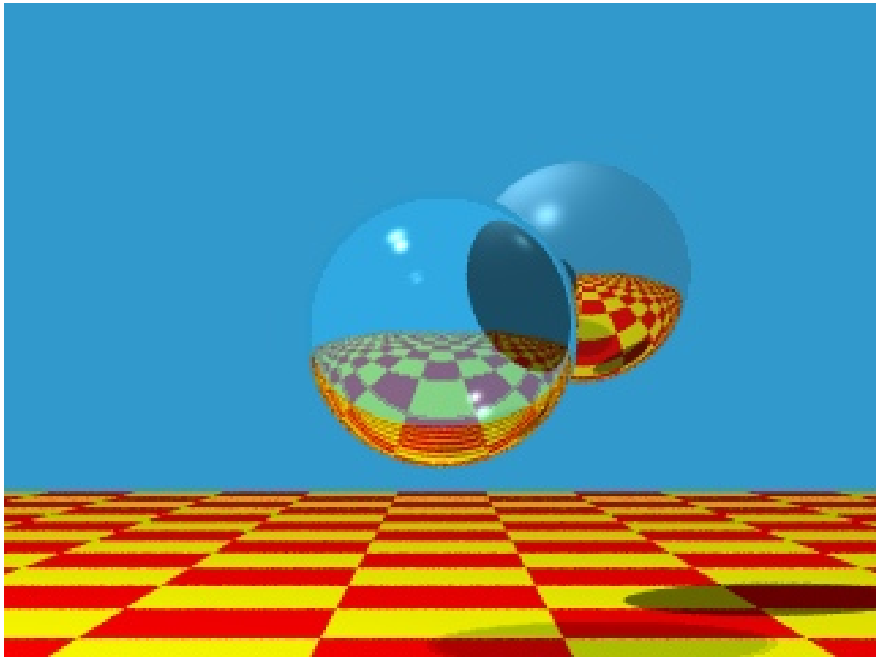
- **Lighting**
  - **Shadows**
  - **Shading: glossy, transparency**
- **Color bleeding, etc**



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**



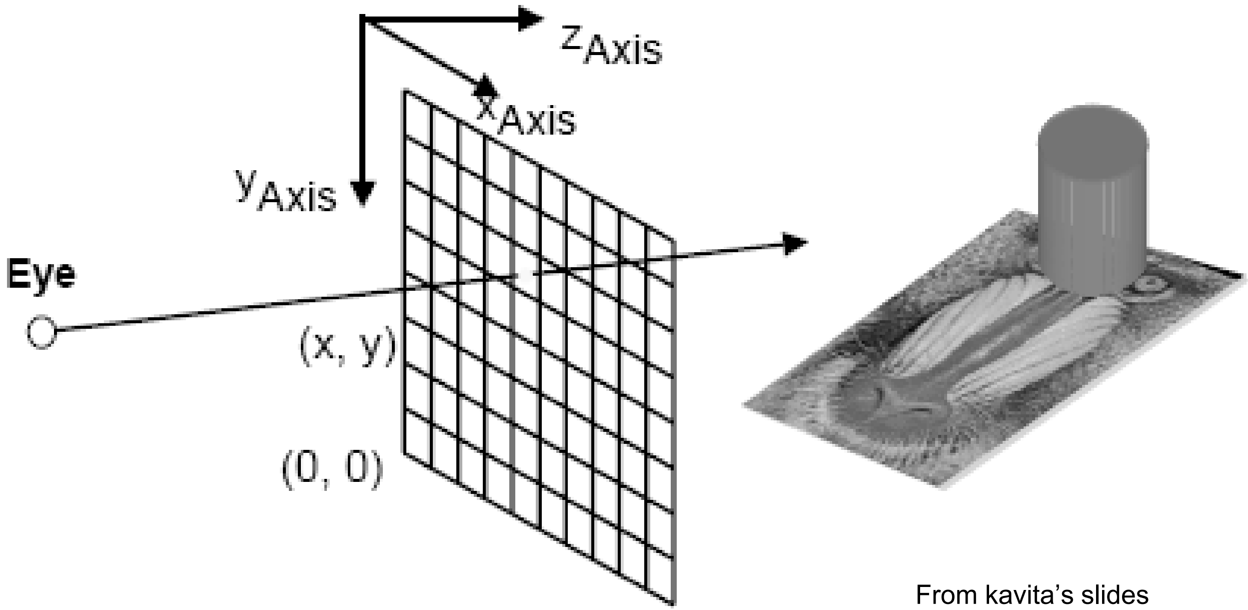
KAIST

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

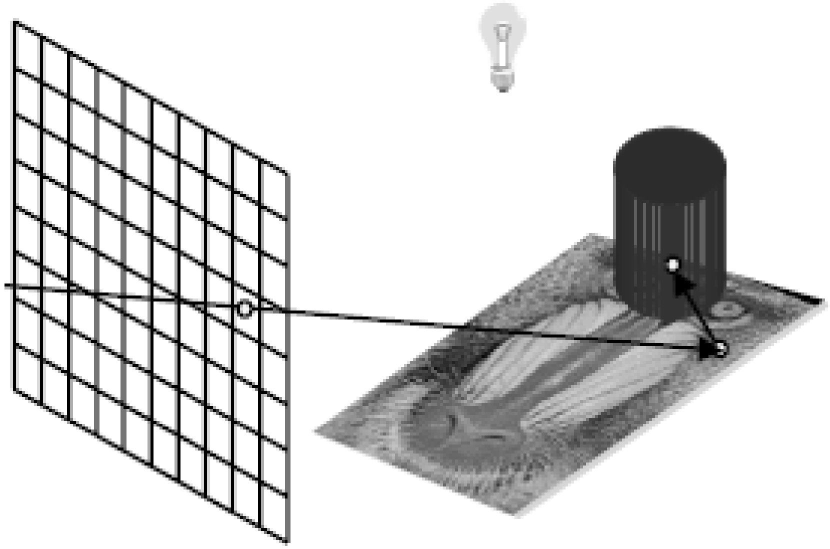
- **Rays are cast from the eye point through each pixel in the image**

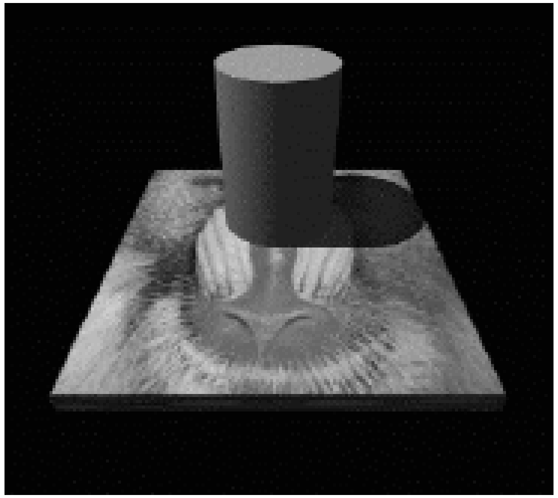


From kavita's slides

Shadows

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

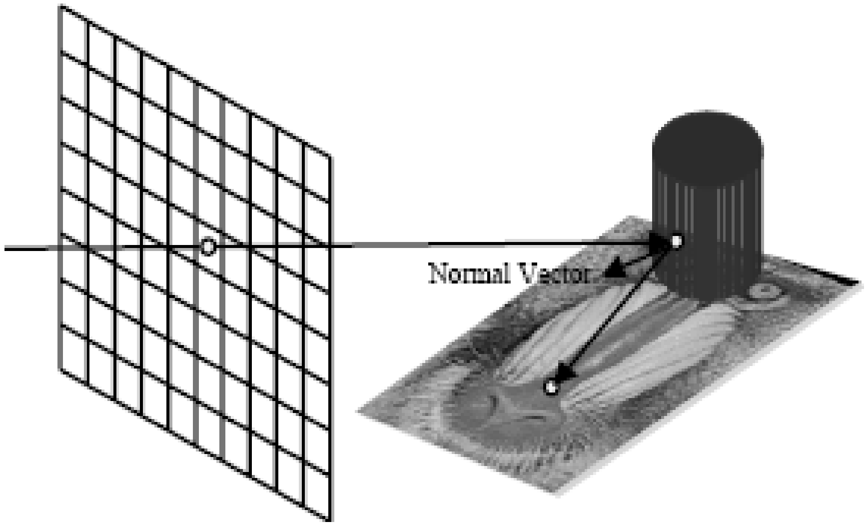


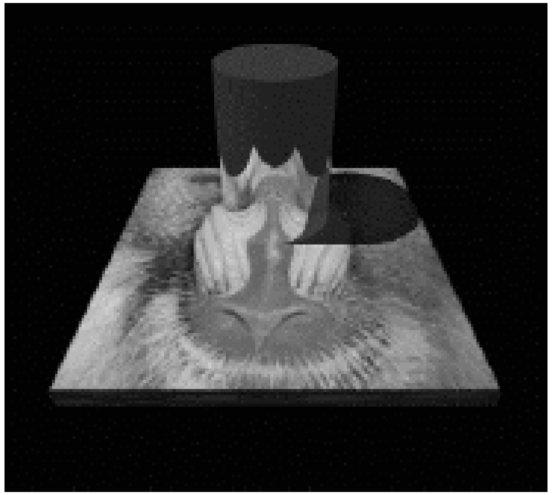


From kavita's slides

Reflections

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

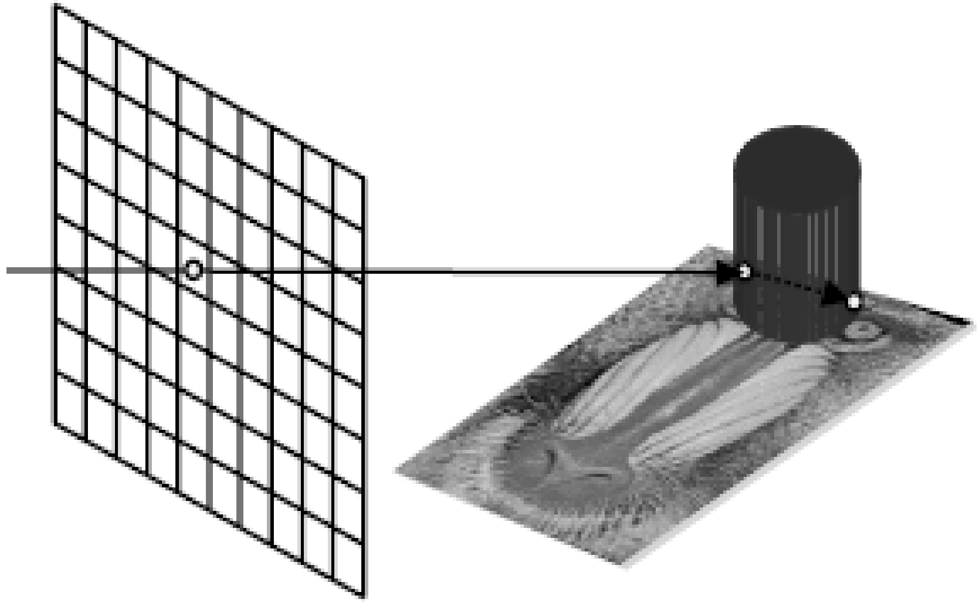


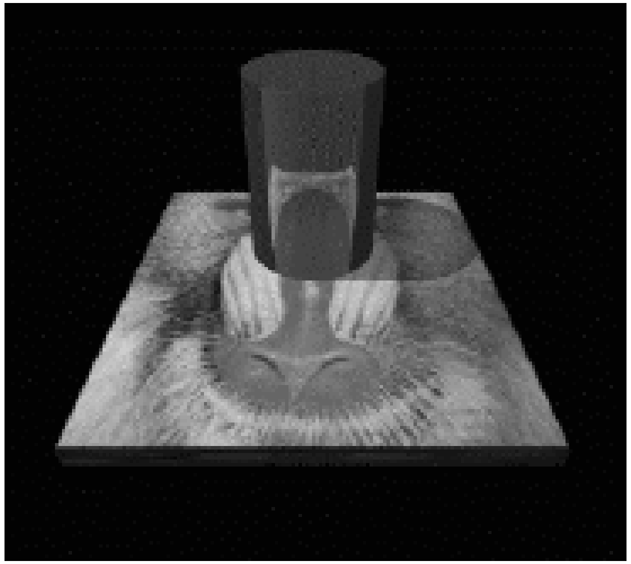


From kavita's slides

Refractions

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





From kavita's slides

An Improved Illumination Model [Whitted 80]

- **Phong illumination model**

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

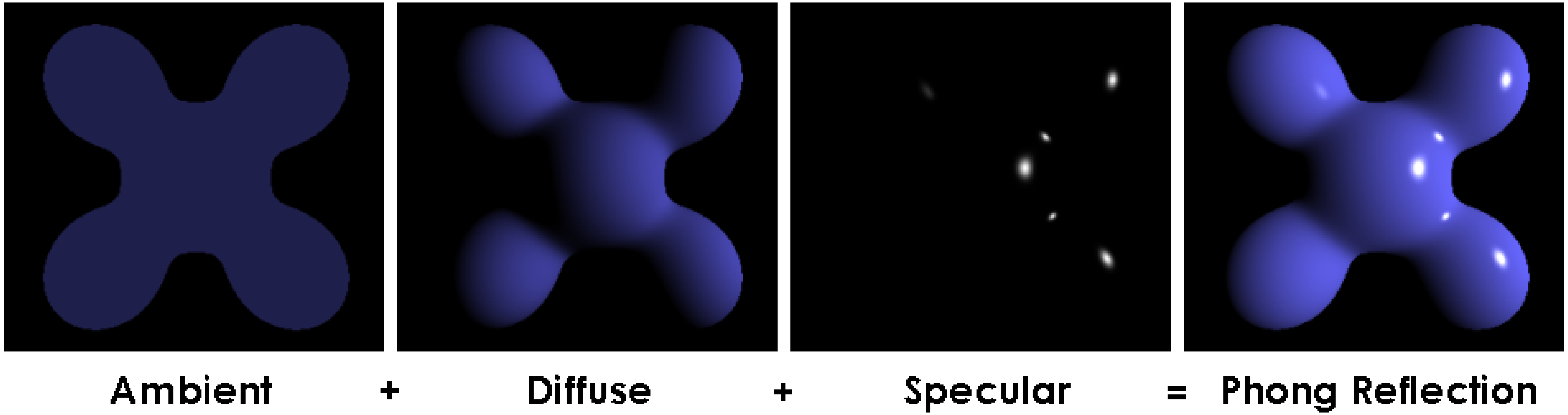
- **Whitted model**

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

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

OpenGL's Illumination Model

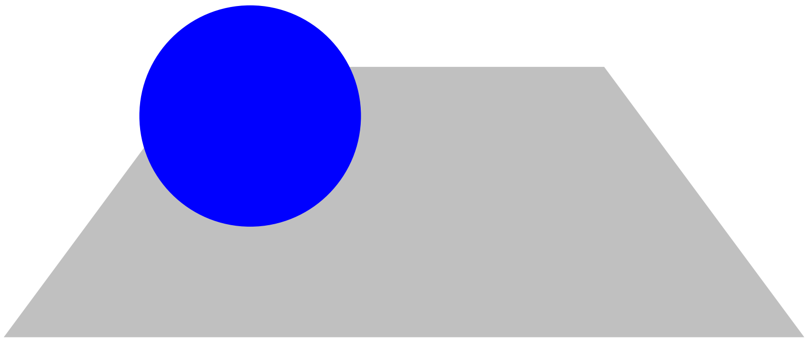
$$I_r = \sum_{j=1}^{numLights} (k_a^j I_a^j + k_d^j I_d^j \max((\hat{N} \bullet \hat{L}_j),0) + k_s^j I_s^j \max((\hat{V} \bullet \hat{R})^{n_s},0))$$



From Wikipedia

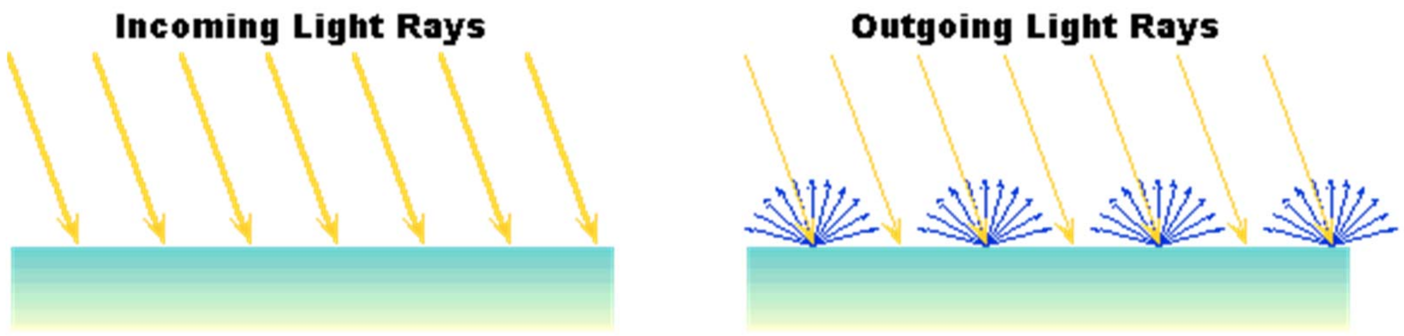
Ambient Light Source

- A simple <u>hack</u> for indirect illumination
  - Incoming ambient illumination ($I_{i,a}$) is constant for all surfaces in the scene
  - Reflected ambient illumination ($I_{r,a}$) depends only on the surface's ambient reflection coefficient (k_a) and not its position or orientation $I_{r,a} = k_a I_{i,a}$
  - These quantities typically specified as (R, G, B) triples



Ideal Diffuse Reflection

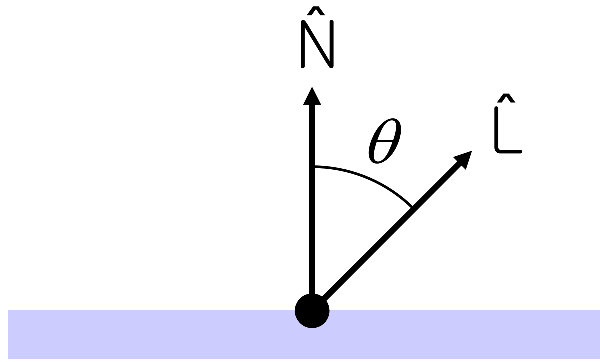
- Ideal diffuse reflectors (e.g., chalk)
  - Reflect uniformly over the hemisphere
  - Reflection is view-independent
  - Very rough at the microscopic level
- Follow Lambert's cosine law

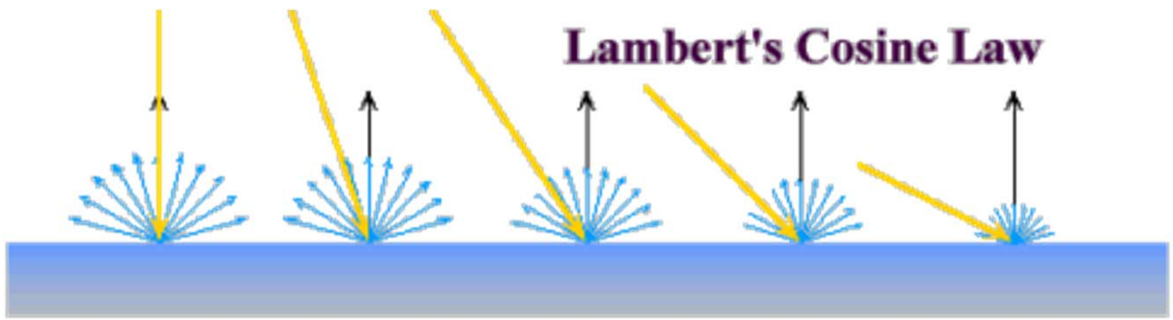


Lambert's Cosine Law

- The reflected energy from a small surface area from illumination arriving from direction $\hat{L}$ is proportional to the cosine of the angle between $\hat{L}$ and the surface normal

$$I_r \approx I_i \cos\theta$$
$$\approx I_i (\hat{N} \bullet \hat{L})$$





Computing Diffuse Reflection

- Constant of proportionality depends on surface properties

$$I_{r,d} = k_d I_i (\hat{N} \bullet \hat{L})$$

  - The constant k_d specifies how much of the incident light I_i is diffusely reflected

Diffuse reflection for varying light directions

  - When $(\hat{N} \cdot \hat{L}) < 0$ the incident light is blocked by the surface itself and the diffuse reflection is 0

Specular Reflection

- Specular reflectors have a bright, view dependent highlight
  - E.g., polished metal, glossy car finish, a mirror
  - At the microscopic level a specular reflecting surface is very smooth
  - Specular reflection obeys Snell's law

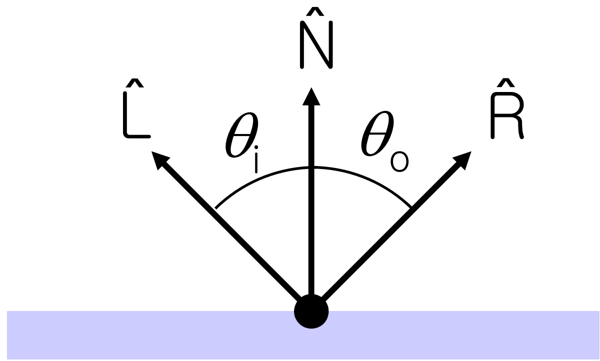
Image source: astochimp.com and wiki

KAIST

Snell's Law

- **The relationship between the angles of the incoming and reflected rays with the normal is given by:**

$$n_i \sin\theta_i = n_o \sin\theta_o$$



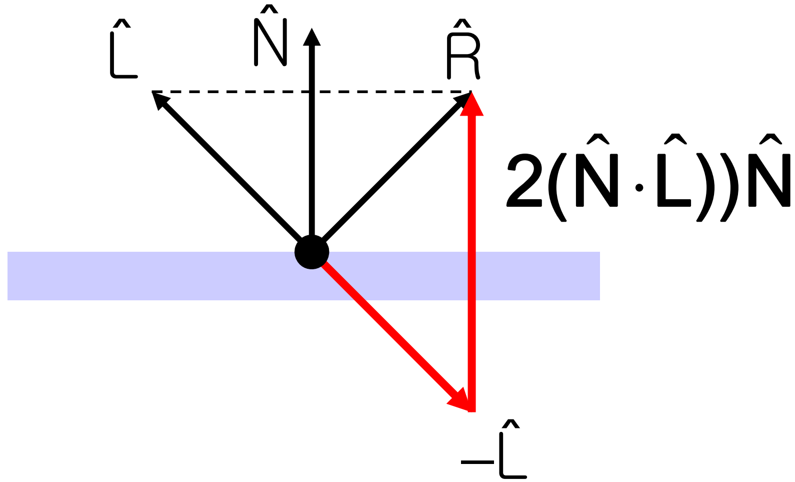
  - **n_i and n_o are the indices of refraction for the incoming and outgoing ray, respectively**
  - **Reflection is a special case where $n_i = n_o$ so $\theta_o = \theta_i$**
  - **The incoming ray, the surface normal, and the reflected ray all lie in a common plane**

Computing the Reflection Vector

- **The vector R can be computed from the incoming light direction and the surface normal as shown below:**

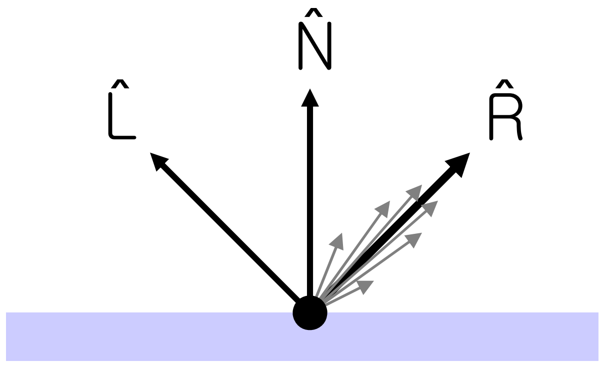
$$\hat{R} = (2(\hat{N} \cdot \hat{L}))\hat{N} - \hat{L}$$

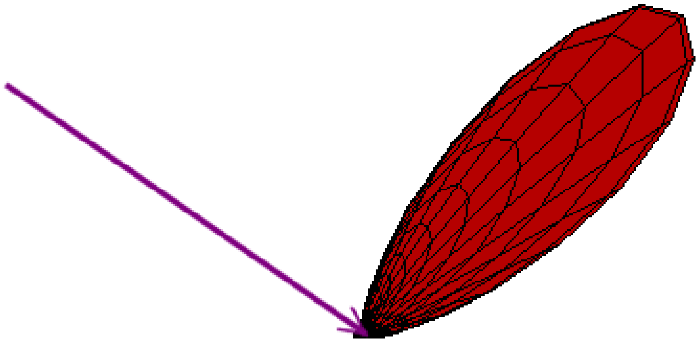
- **How?**



Non-Ideal Reflectors

- Snell's law applies only to *ideal* specular reflectors
  - Roughness of surfaces causes highlight to "spread out"
  - Empirical models try to simulate the appearance of this effect, without trying to capture the physics of it

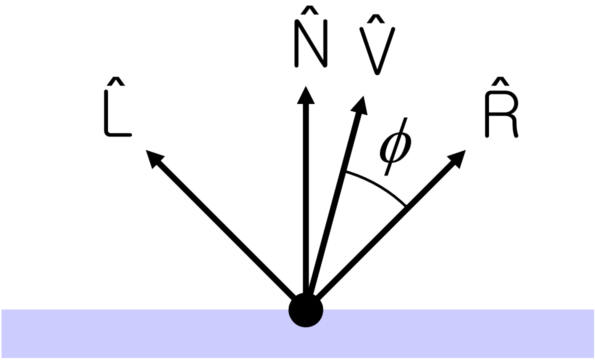




Phong Illumination

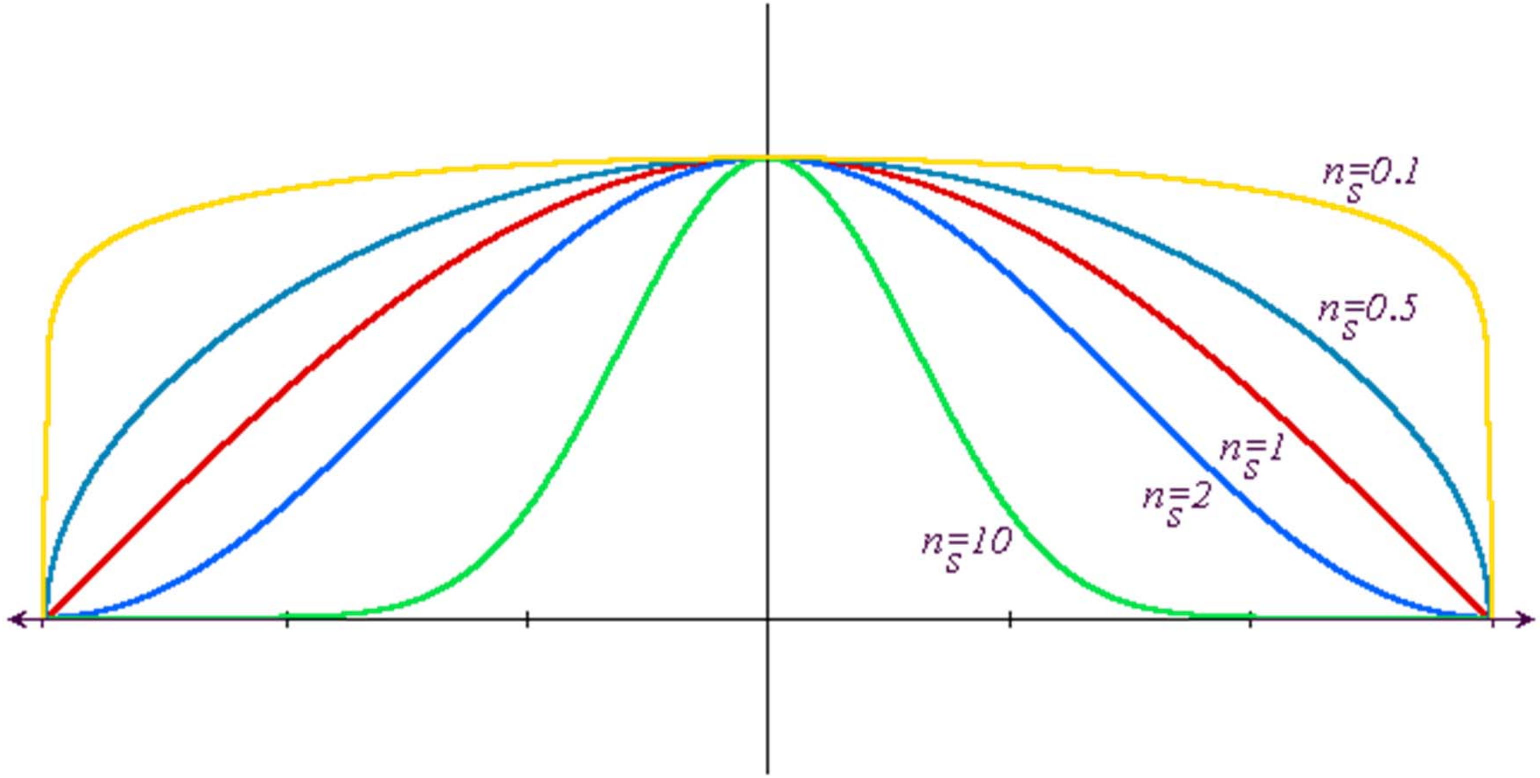
- **One of the most commonly used illumination models in computer graphics**
  - **Empirical model and does not have no physical basis**

$$I_r = k_s I_i (\cos\phi)^{n_s} = k_s I_i (\hat{V} \bullet \hat{R})^{n_s}$$



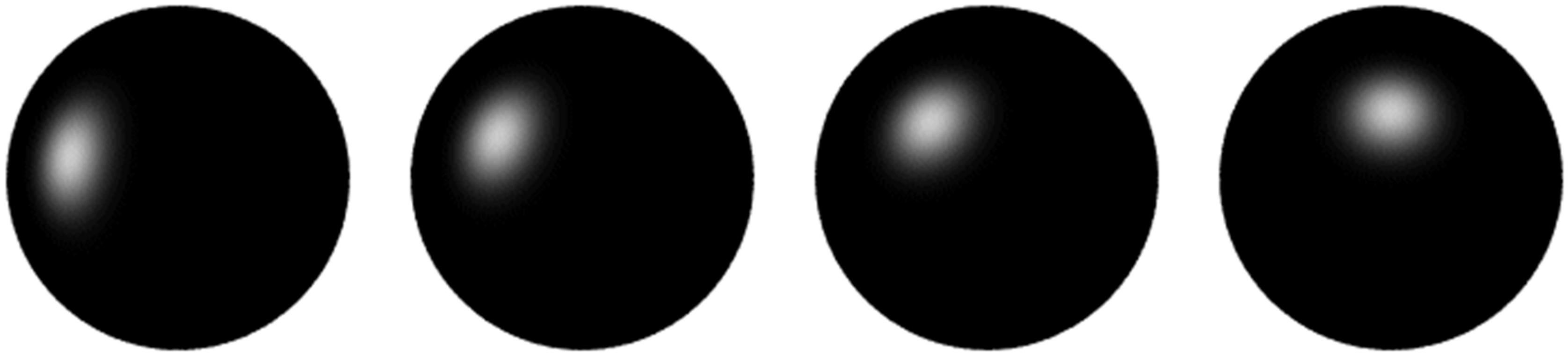
- $(\hat{V})$ **is the direction to the viewer**
  - $(\hat{V} \bullet \hat{R})$ **is clamped to [0,1]**
  - **The specular exponent n_s controls how quickly the highlight falls off**

Effect of Specular Exponent

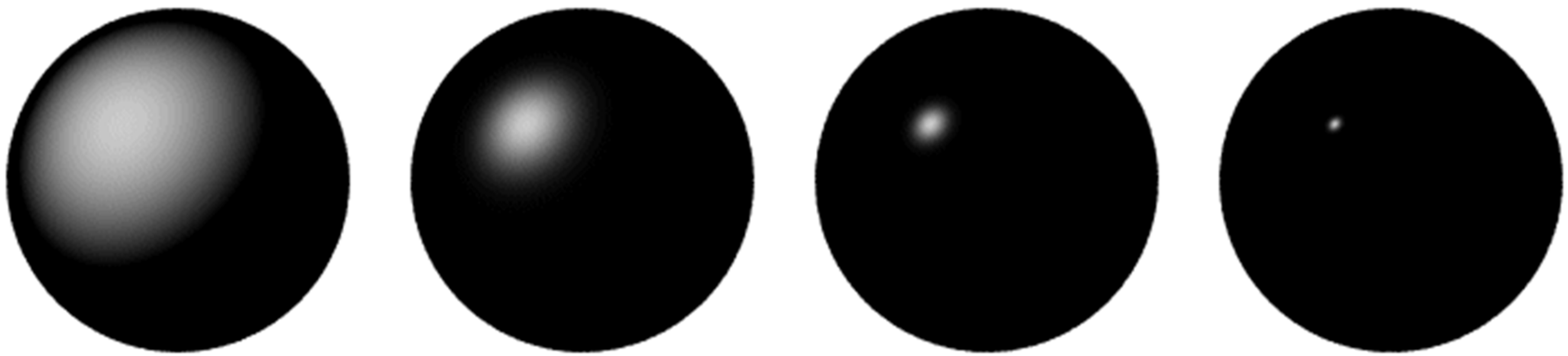


- **How the shape of the highlight changes with varying n_s**

Examples of Phong



varying light direction



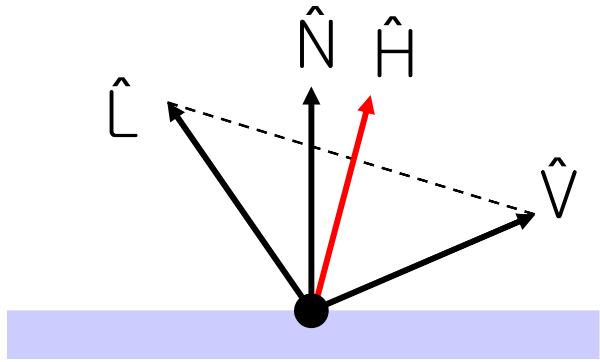
varying specular exponent

Blinn & Torrance Variation

- **Jim Blinn introduced another approach for computing Phong-like illumination based on the work of Ken Torrance:**

$$\hat{H} = \frac{\hat{L} + \hat{V}}{\left|\hat{L} + \hat{V}\right|}$$

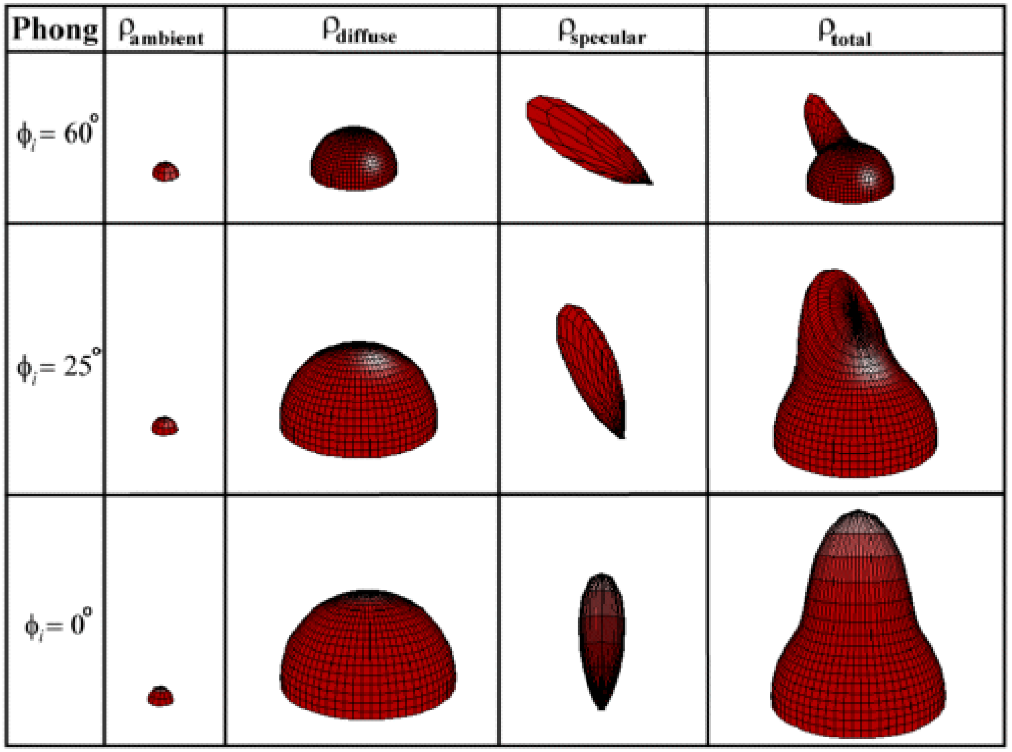
$$I_{r,s} = k_s I_i (\hat{N} \cdot \hat{H})^{n_s}$$



- $\hat{H}$ **is the half-way vector that bisects the light and viewer directions**

Putting it All Together

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



An Improved Illumination Model [Whitted 80]

- **Phong illumination model**

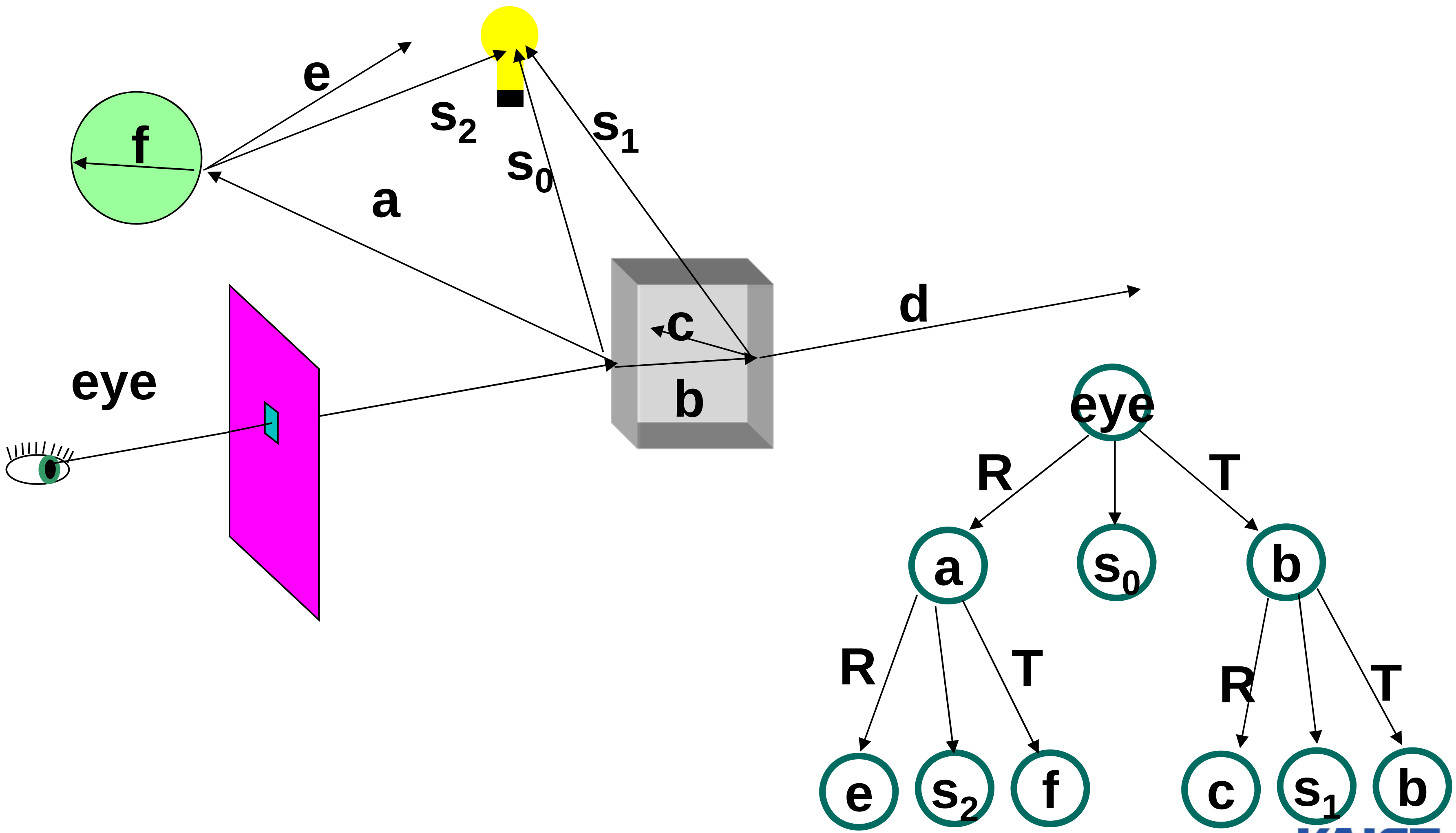
$$I_r = \sum_{j=1}^{numLights} (k_a^j I_a^j + k_d^j I_d^j (\hat{N} \bullet \hat{L}_j) + k_s^j I_s^j (\hat{V} \bullet \hat{R})^{n_s})$$

- **Whitted model**

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

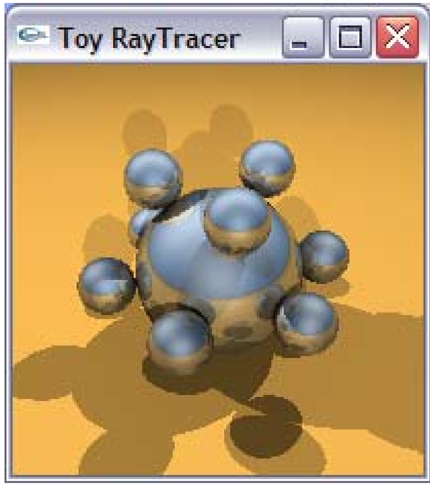
  - **S and T are intensity of light from reflection and transmission rays**
  - **Ks and Kt are specular and transmission coefficient**

Ray Tree



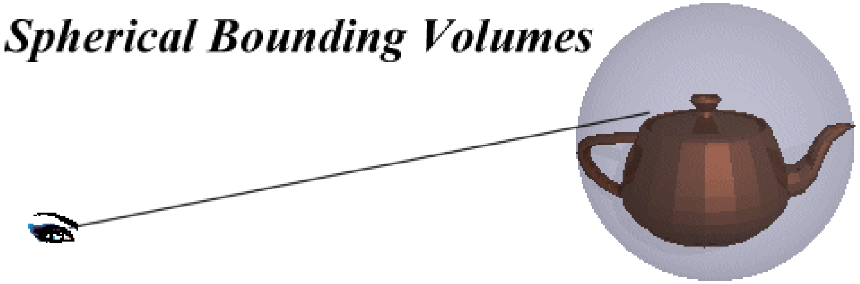
Acceleration Methods for Ray Tracing

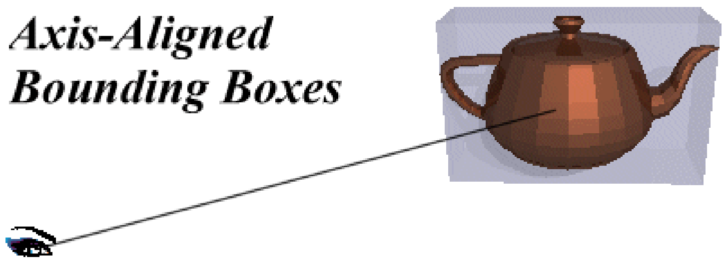
- Rendering time for a ray tracer depends on the number of ray intersection tests per pixel
  - The number of pixels X the number of primitives in the scene
- Early efforts focused on accelerating the ray-object intersection tests
  - Ray-triangle intersection tests
- More advanced methods required to make ray tracing practical
  - Bounding volume hierarchies
  - Spatial subdivision (e.g., kd-trees)



Bounding Volumes

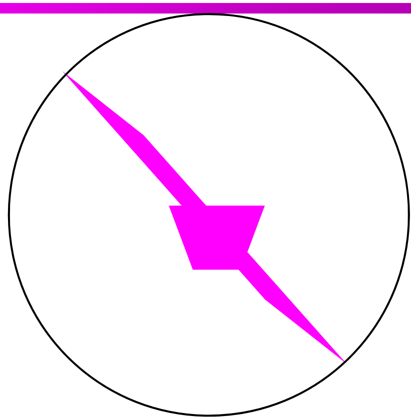
- Enclose complex objects within a simple-to-intersect objects
  - If the ray does not intersect the simple object then its contents can be ignored
  - The likelihood that it will strike the object depends on how tightly the volume surrounds the object.
- Spheres are simple, but not tight
- Axis-aligned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



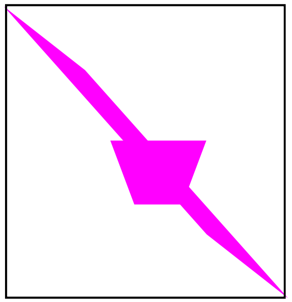


Bounding Volumes

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

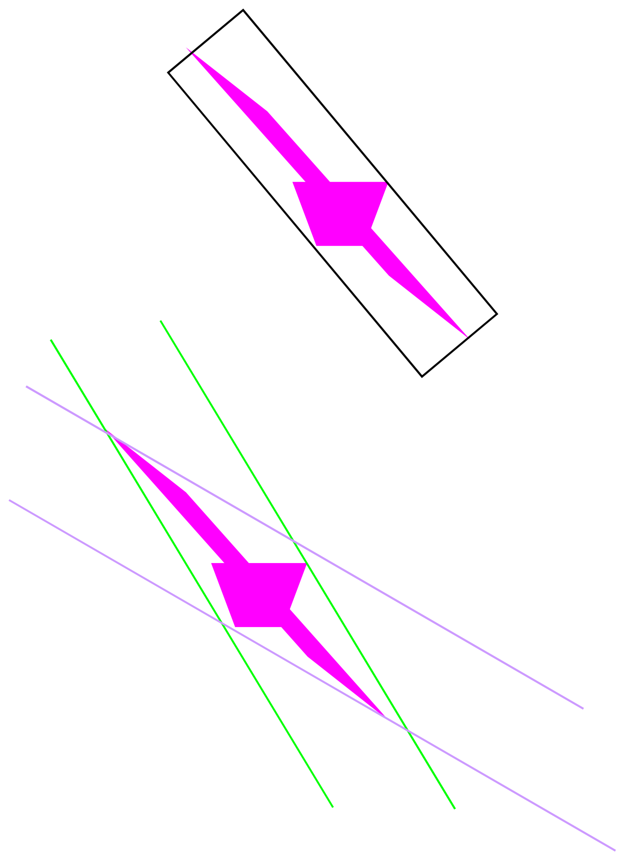


- **Axis-Aligned Bounding Box**
  - **Very cheap to compute**
  - **Cheap test**
  - **Tighter than sphere**



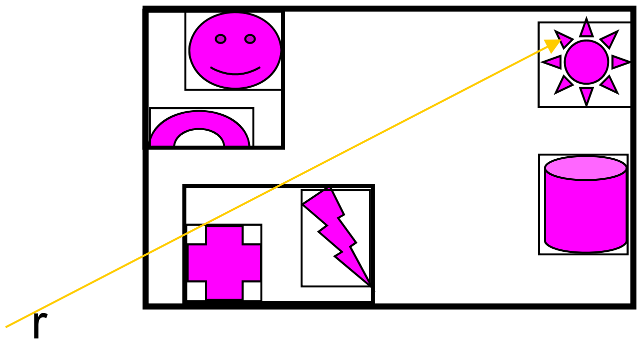
Bounding Volumes

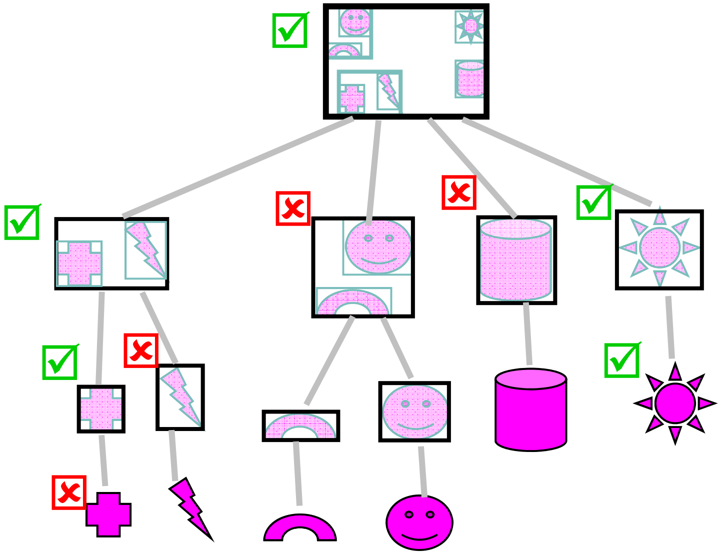
- Oriented Bounding Box
  - Fairly cheap to compute
  - Fairly Cheap test
  - Generally fairly tight
- Slabs / K-dops
  - More expensive to compute
  - Fairly cheap test
  - Can be tighter than OBB



Hierarchical Bounding Volumes

- **Organize bounding volumes as a tree**
  - **Choose a partitioning plane and distribute triangles into left and right nodes**
- **Each ray starts with the scene BV and traverses down through the hierarchy**

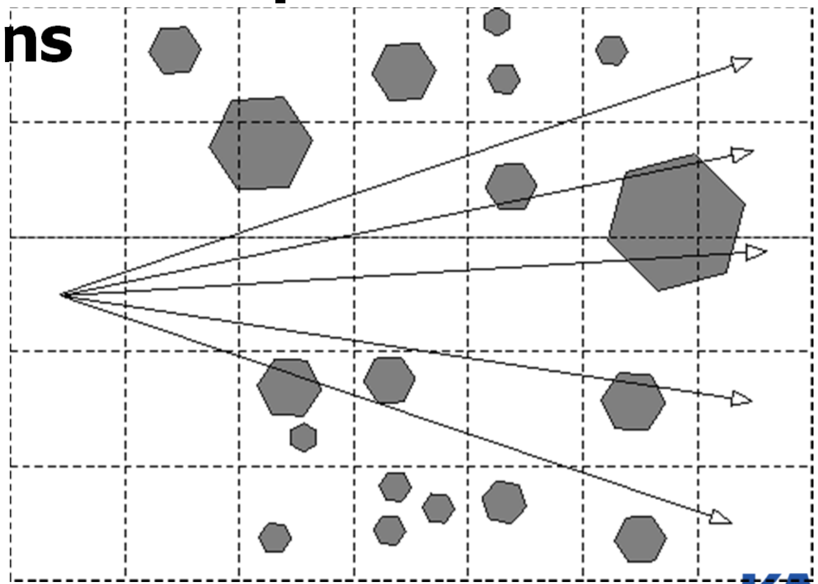




Spatial Subdivision

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**

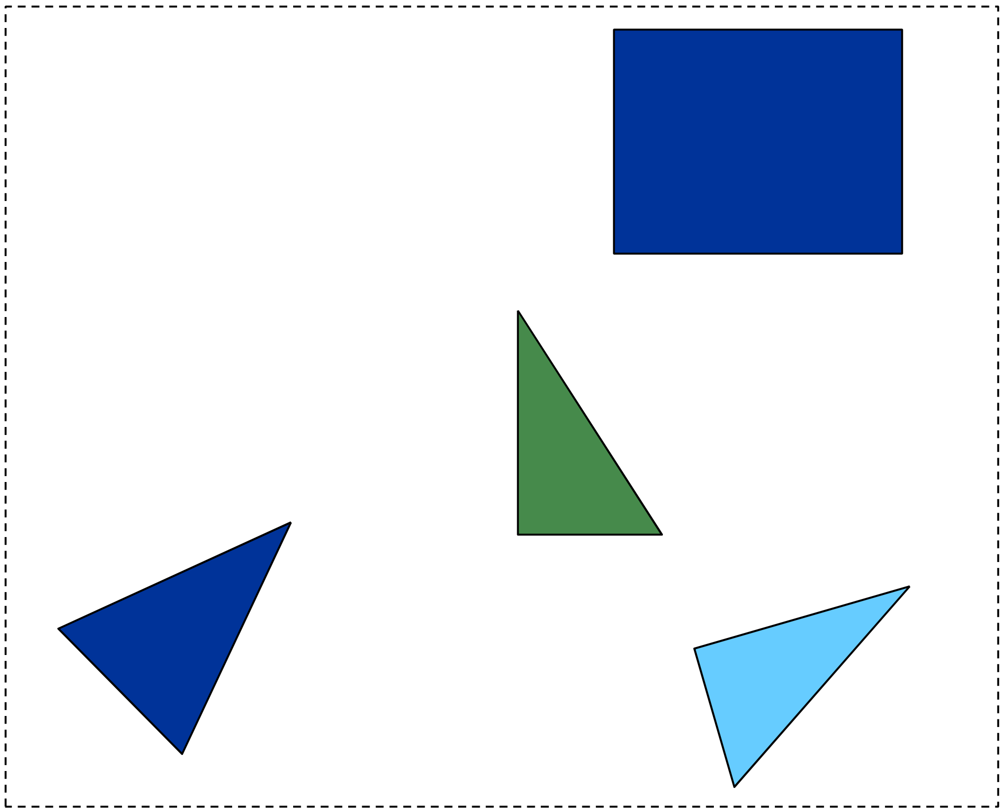


KAIST

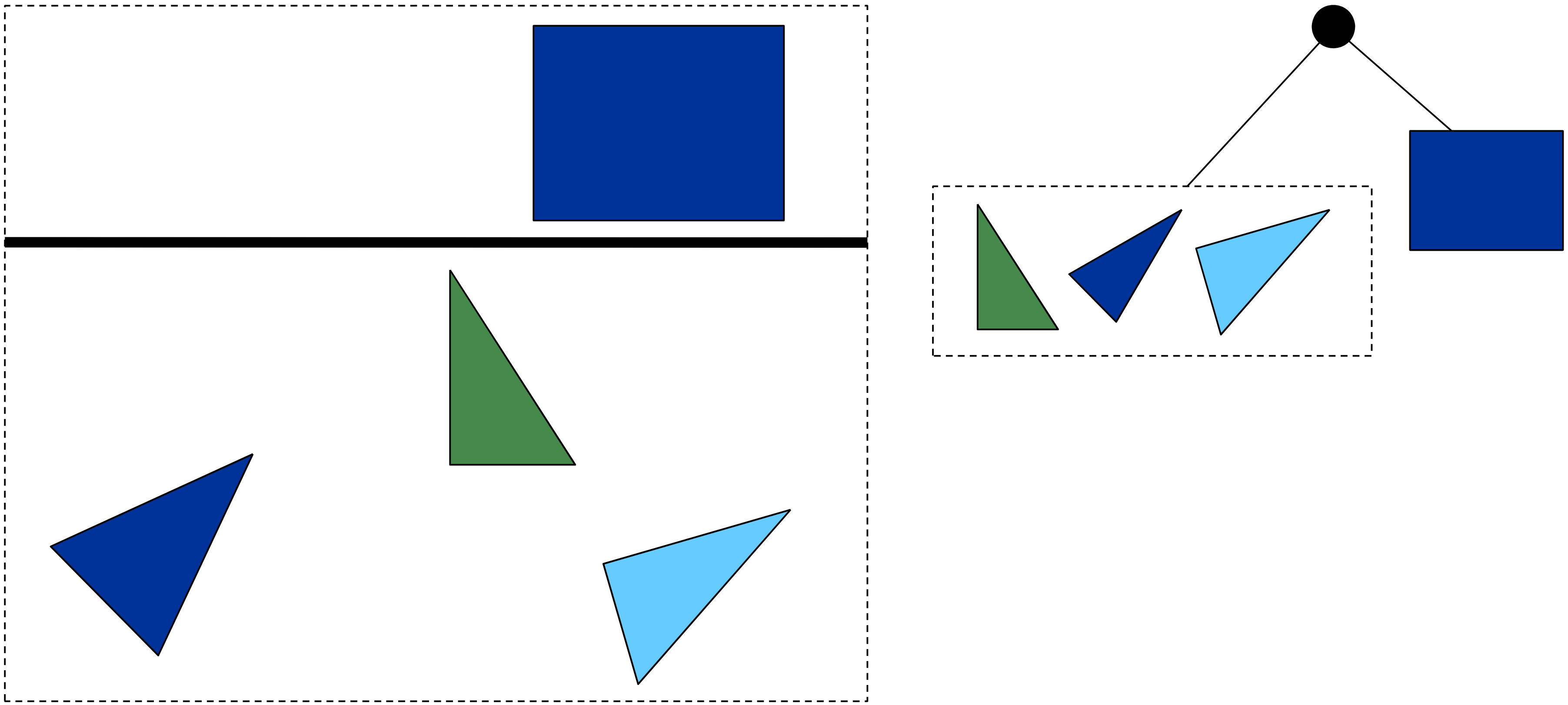
Overview of kd-Trees

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

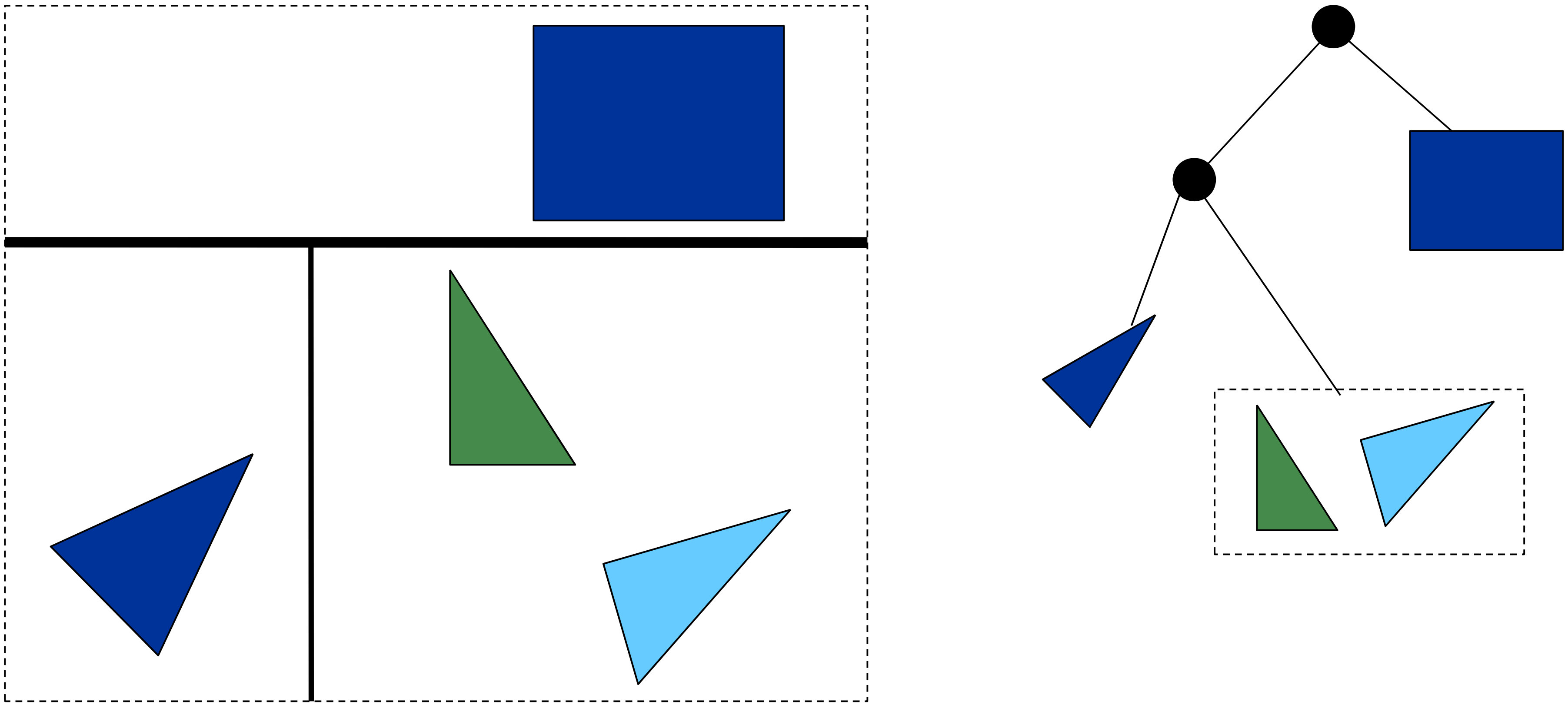
Example



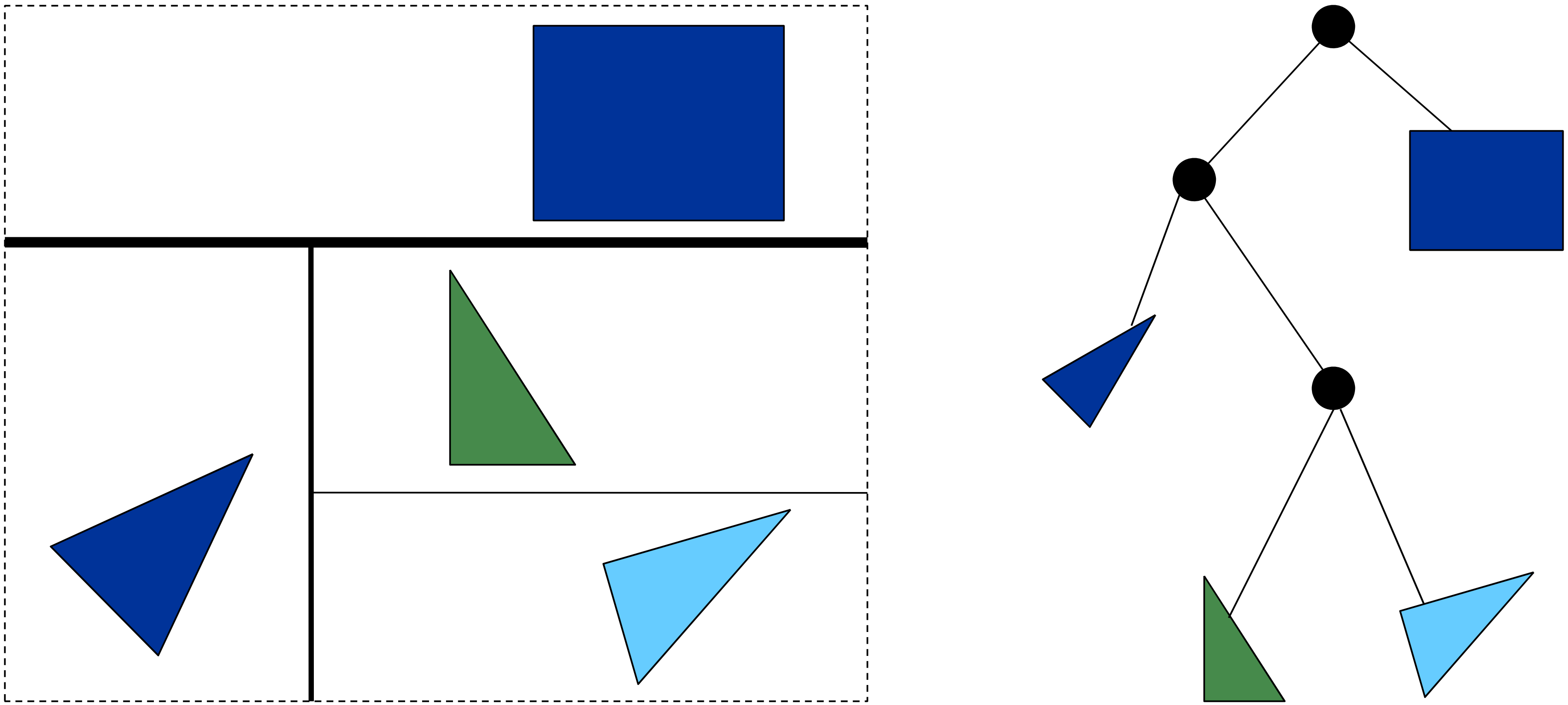
Example



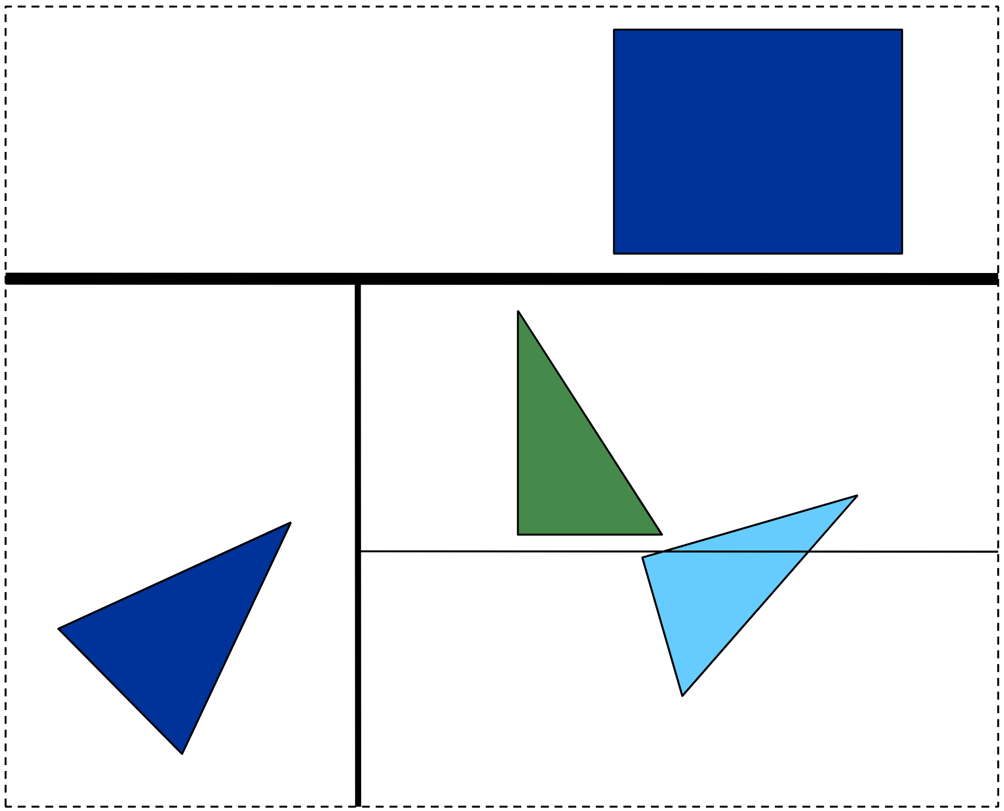
Example



Example

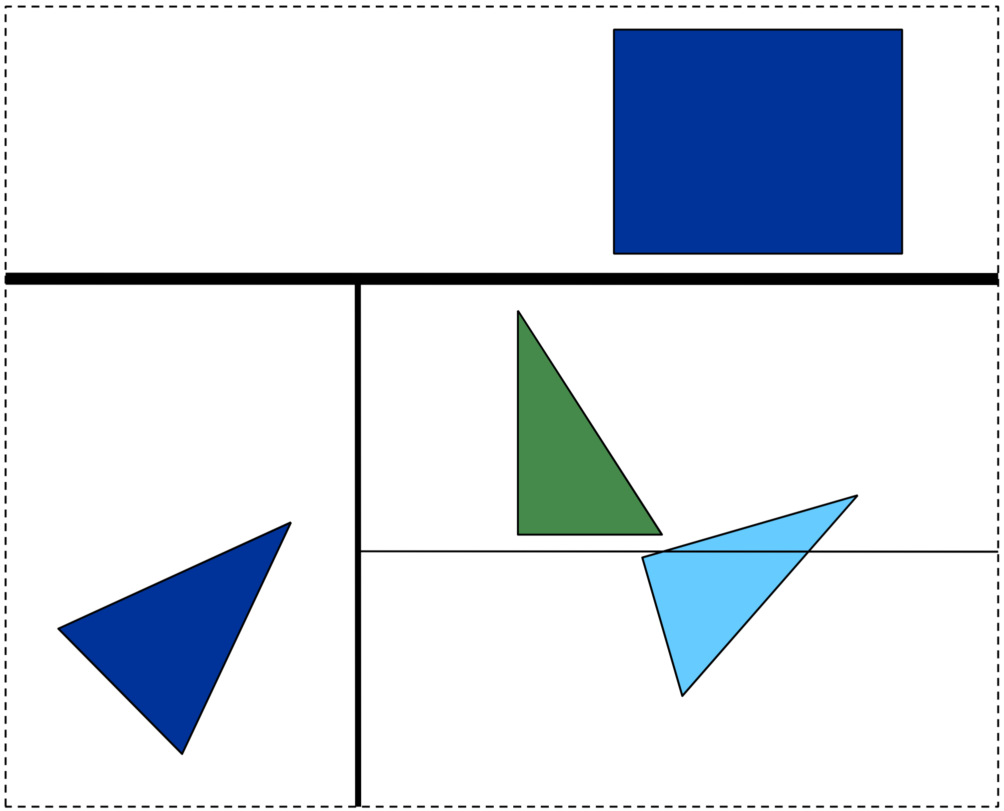


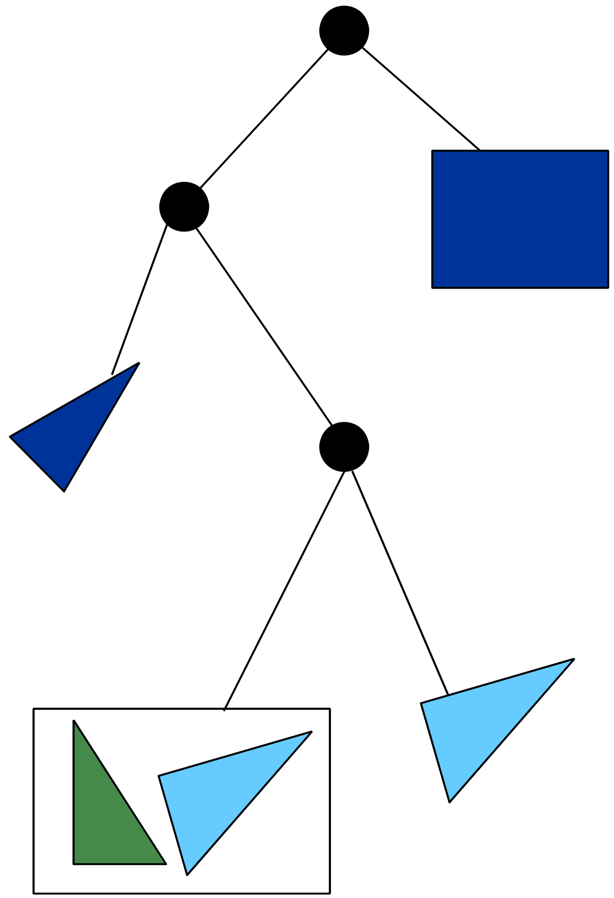
Example



What about triangles overlapping the split?

Example

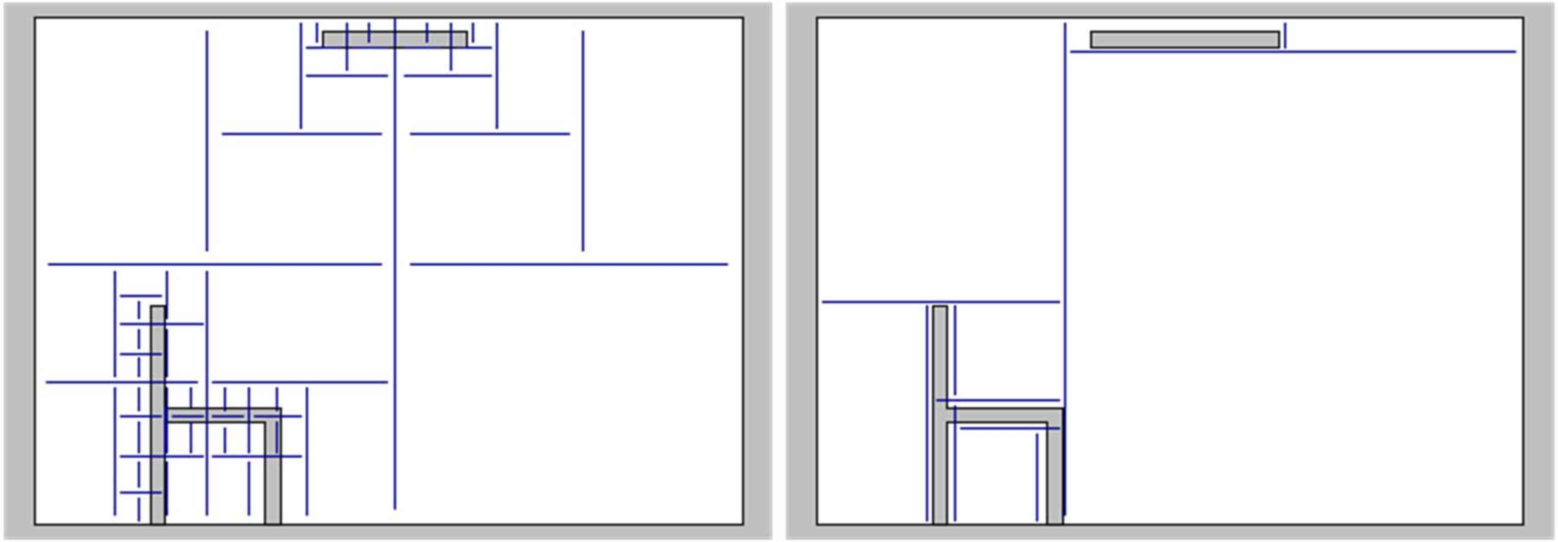




Split Planes

- 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



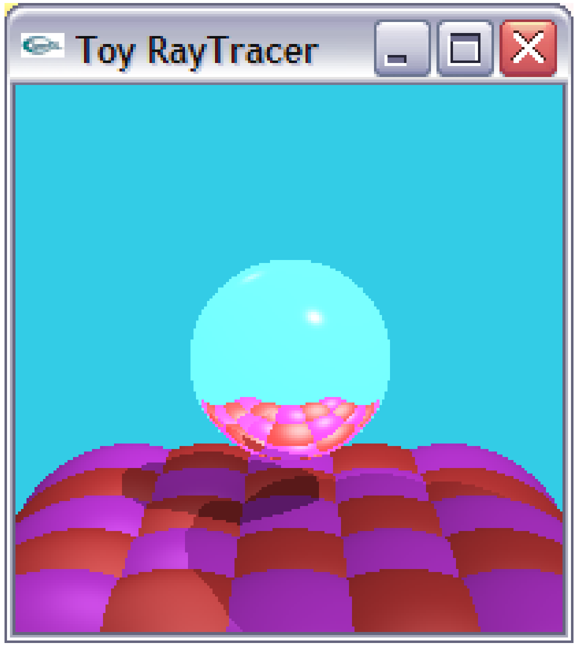
(from [Wald04])

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**

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**

Class Objectives were:

- **Understand a basic ray tracing**
- **Know the Phong illumination model**
- **Implement its acceleration data structure and know how to use it**

Any Questions?

- **Come up with one question on what we have discussed in the class and submit at the end of the class**
  - **1 for already answered questions**
  - **2 for typical questions**
  - **3 for questions with thoughts**
  - **4 for questions that surprised me**

Homework

- **Go over the next lecture slides before the class**
- **Watch 2 SIGGRAPH videos and submit your summaries every Tue. class**
  - **Just one paragraph for each summary**

Example:

Title: XXX XXXX XXXX

Abstract: this video is about accelerating the performance of ray tracing. To achieve its goal, they design a new technique for reordering rays, since by doing so, they can improve the ray coherence and thus improve the overall performance.

Next Time

- **Radiosity**