CS380: Computer Graphics
Ray Tracing

Sung-Eui Yoon
(윤성의)

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

KAIST

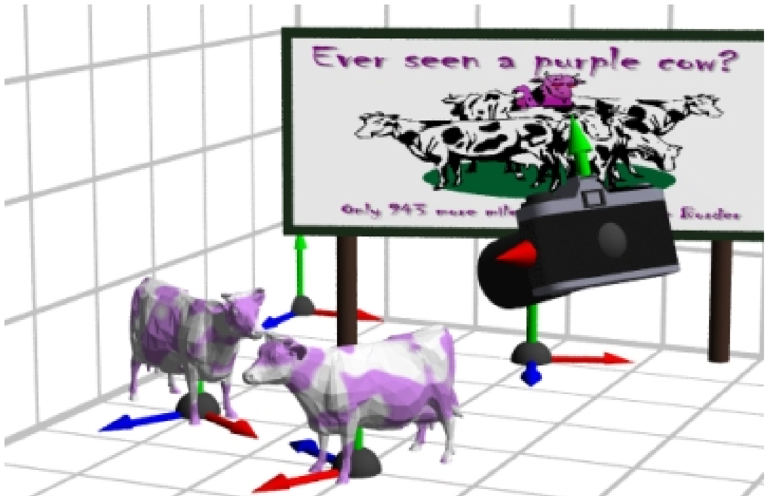
Class Objectives (Ch. 10)

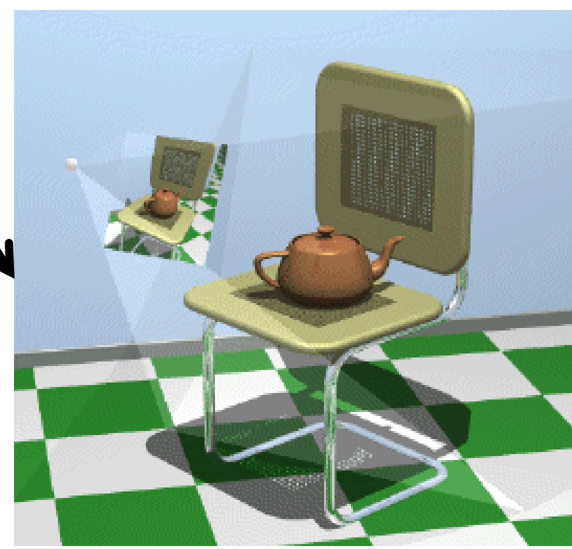
- **Understand a basic ray tracing**
- **Know its acceleration data structure and how to use it**
- **Related chapter**
  - **Part II, Ray Tracing**
  - **https://sgvr.kaist.ac.kr/~sungeui/render/**

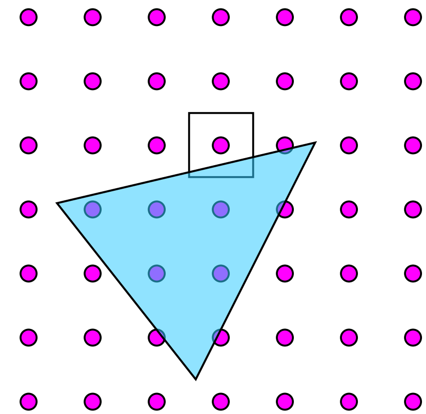
- **At the last class:**
  - **Many different part of rasterization process**
  - **Texture mapping and filtering methods**

The Classic Rendering Pipeline

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





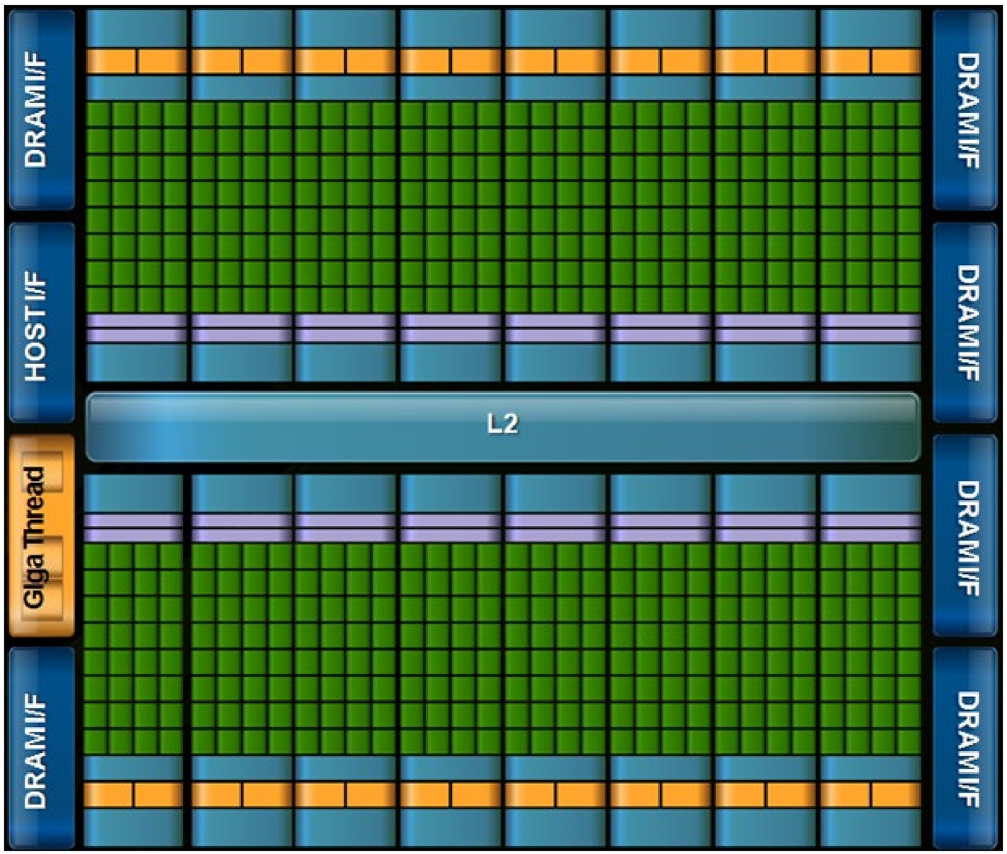


Why we are using rasterization?

- Efficiency
- Reasonably quality

Fermi GPU Architecture

16 SM (streaming processors)



512 CUDA cores

Memory interfaces

Nvidia Hopper Architecture (18K FP32 cores)

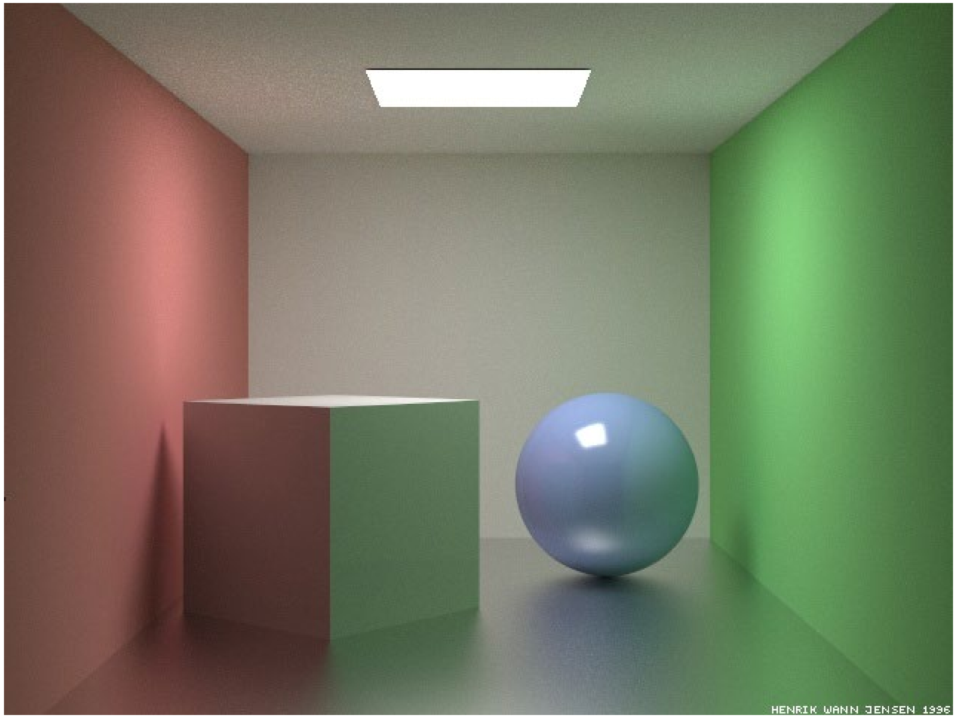
Where Rasterization Is

From Battlefield: Bad Company, EA Digital Illusions CE AB

From Eric Haines

But what about other visual cues?

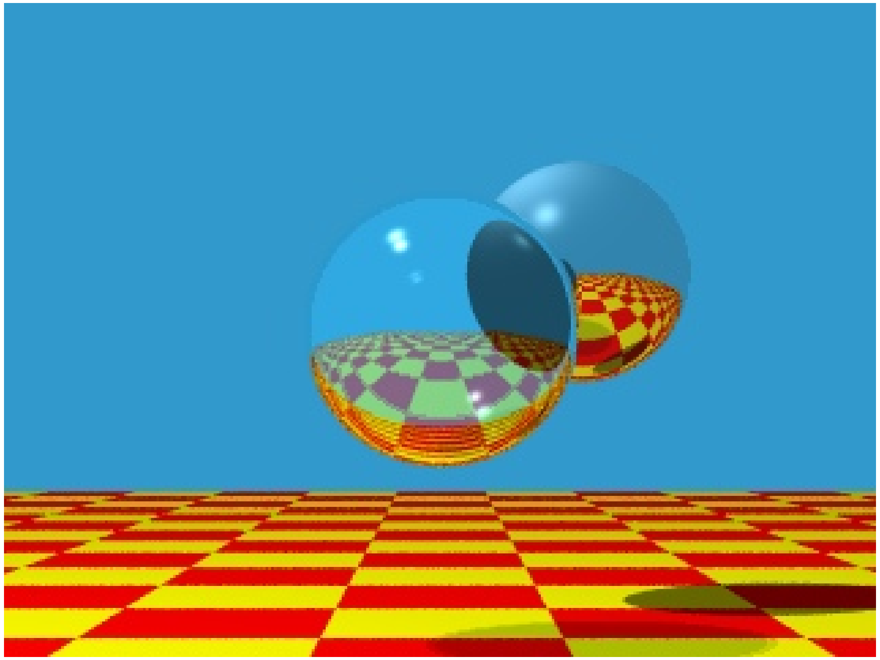
- Lighting
  - Shadows
  - Shading: glossy, transparency
- Color bleeding, etc



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**

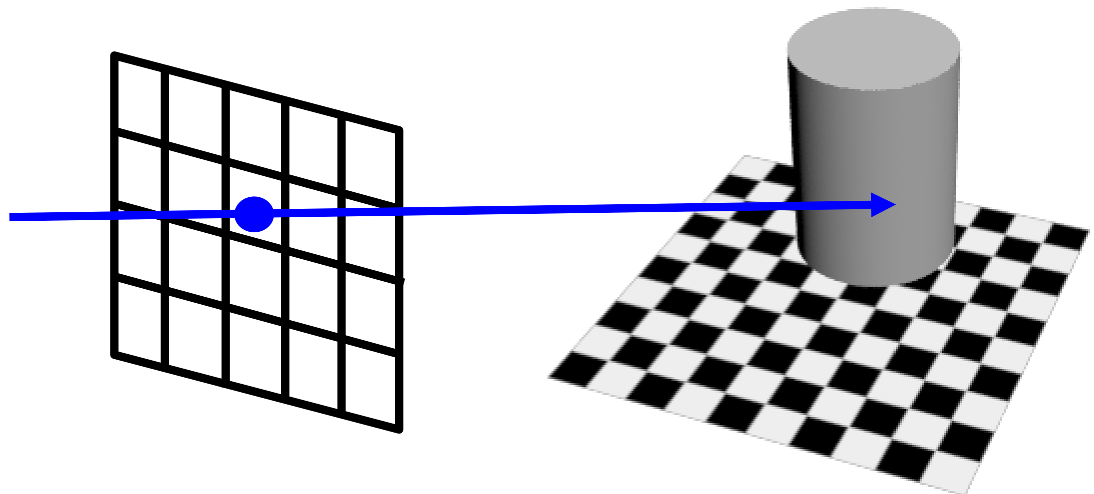


Ray Casting and Ray Tracing

- 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

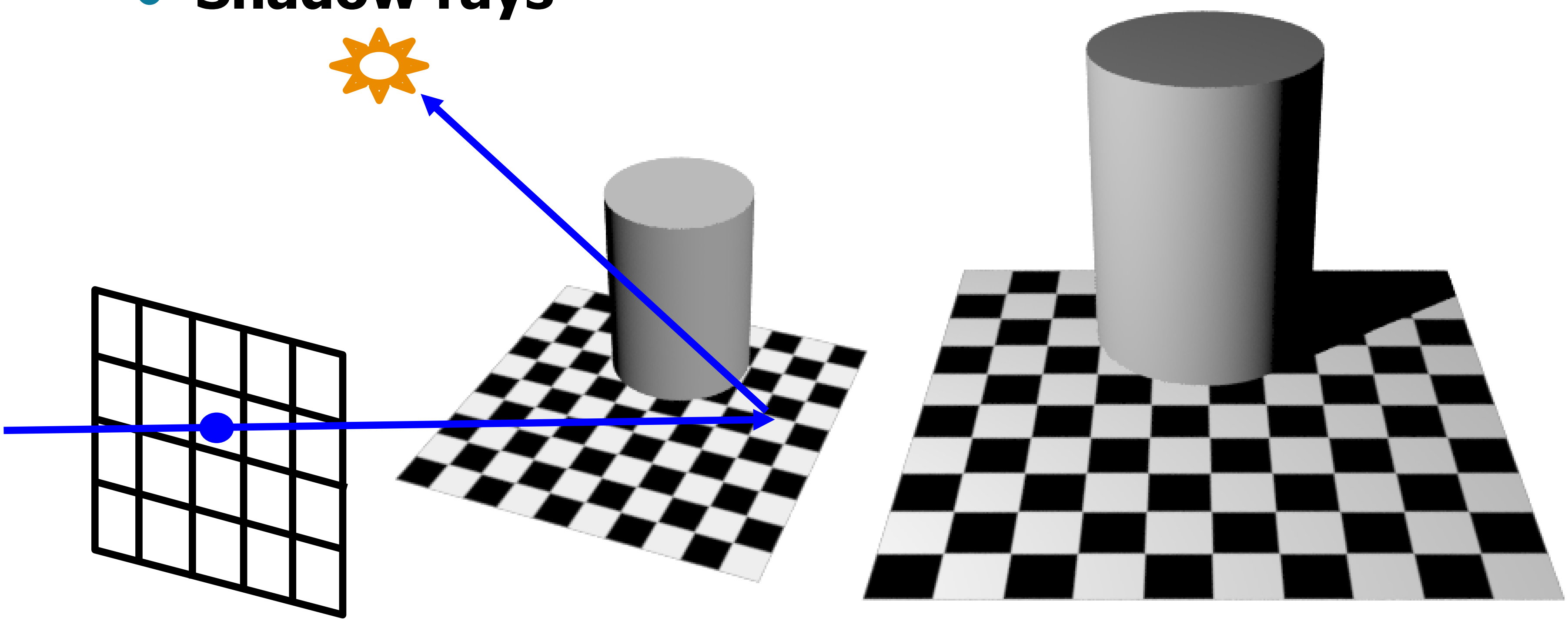
Basic Algorithms

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



Shadows

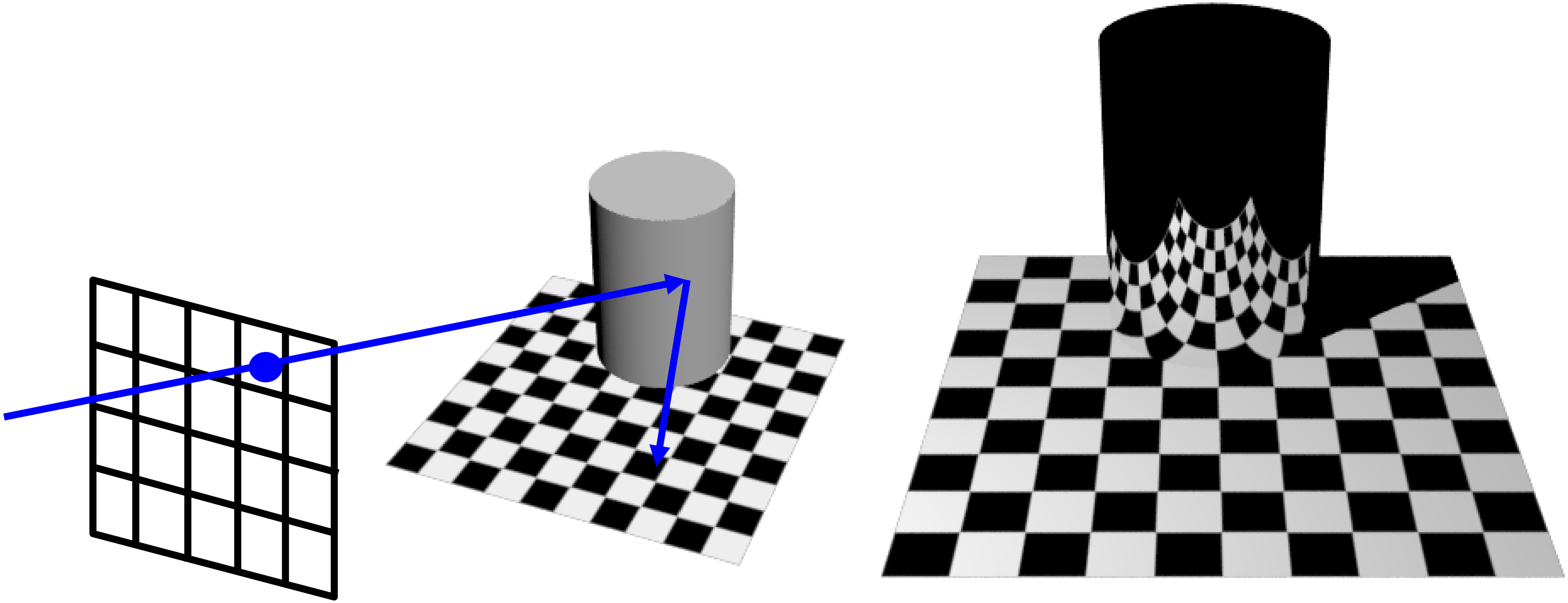
- **Cast ray from the intersection point to each light source**
  - **Shadow rays**



Used 3ds Max

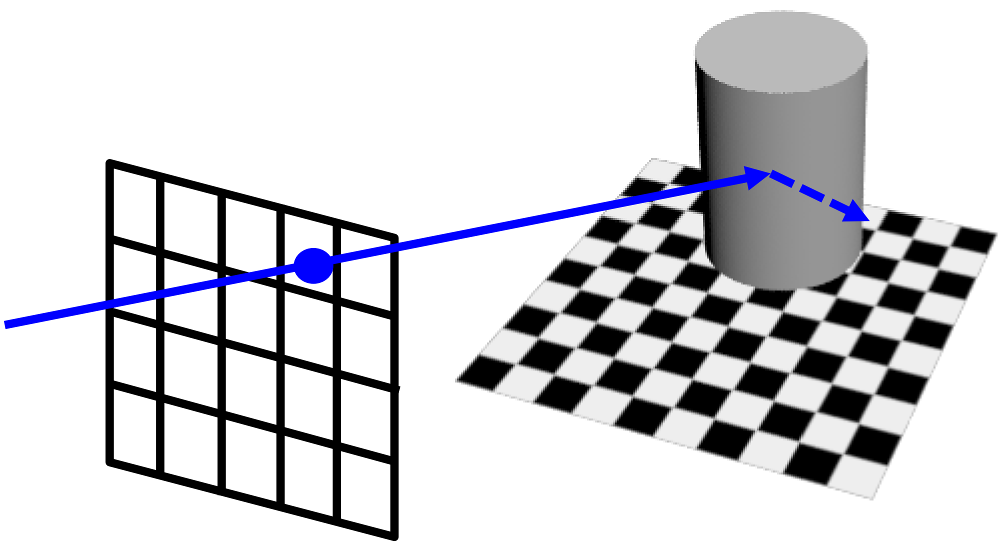
Reflections

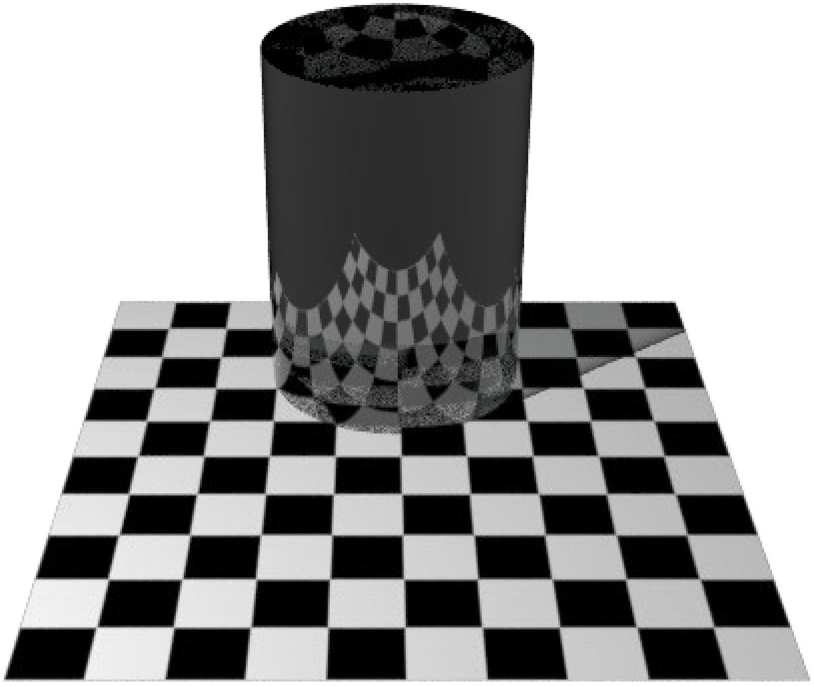
- **If object specular, cast secondary reflected rays**



Refractions

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





An Improved Illumination Model [Whitted 80]

- **Phong illumination model**

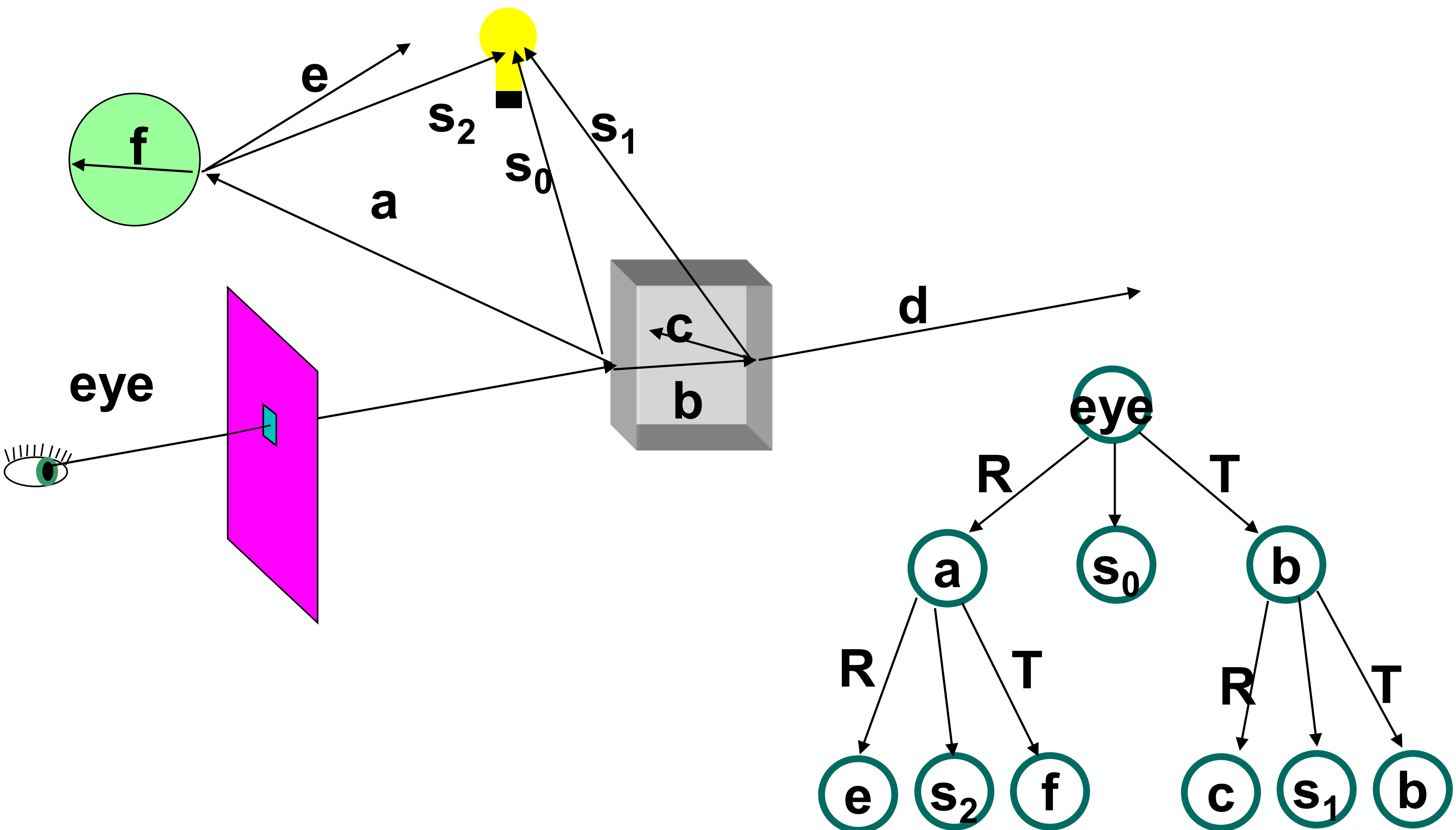
$$I_r = \sum_{j=1}^{numLights} (k_a^j I_a^j + k_d^j I_d^j (\hat{N} \bullet \hat{L}_j) + k_s^j I_s^j (\hat{V} \bullet \hat{R})^{n_s})$$

- **Whitted model**

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

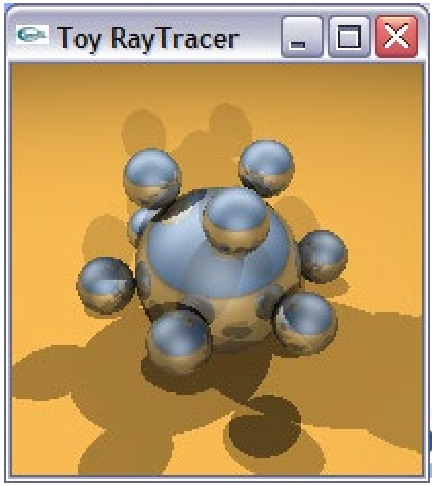
  - **S and T are intensity of light from reflection and transmission rays**
  - **Ks and Kt are specular and transmission coefficient**

Ray Tree



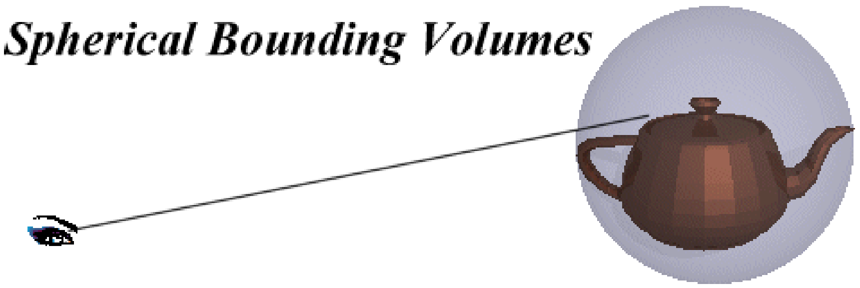
Acceleration Methods for Ray Tracing

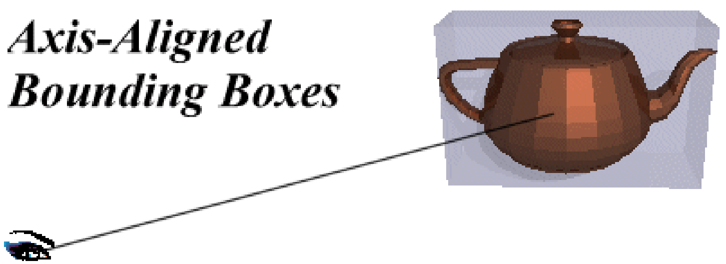
- Rendering time for a ray tracer depends on the number of ray intersection tests per pixel
  - The number of pixels X the number of primitives in the scene
- Early efforts focused on accelerating the ray-object intersection tests
  - Ray-triangle intersection tests
- More advanced methods required to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



Bounding Volumes

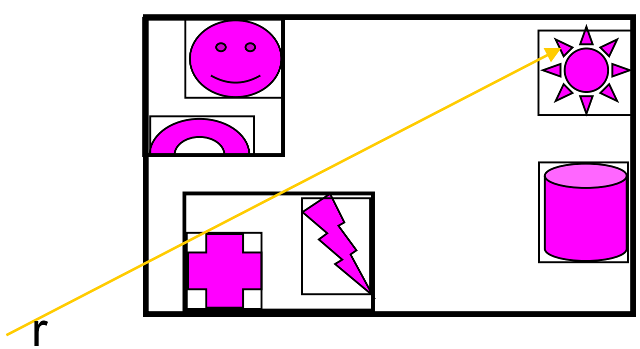
- **Enclose complex objects within a simple-to-intersect objects**
  - **If the ray does not intersect the simple object then its contents can be ignored**
  - **The likelihood that it will strike the object depends on how tightly the volume surrounds the object.**
- **Spheres are simple, but not tight**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

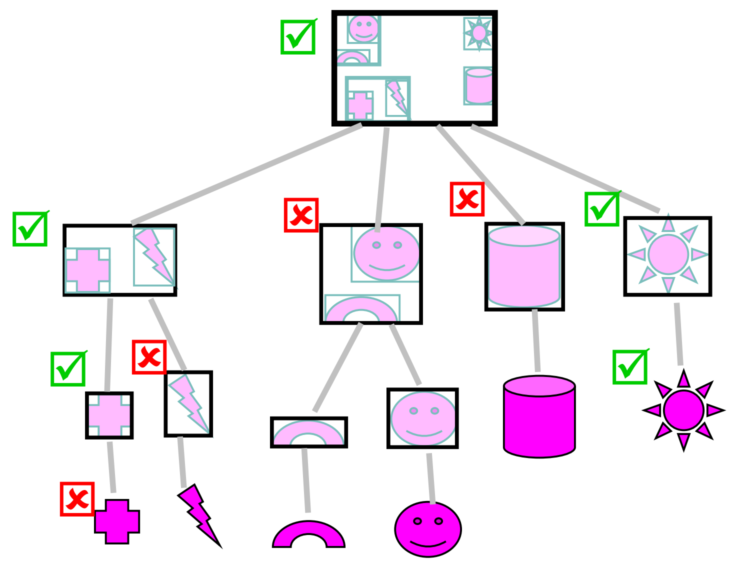




Bounding Volume Hierarchy (BVH)

- **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**





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

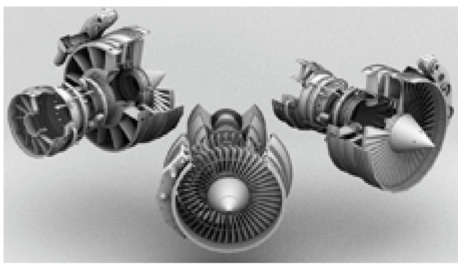
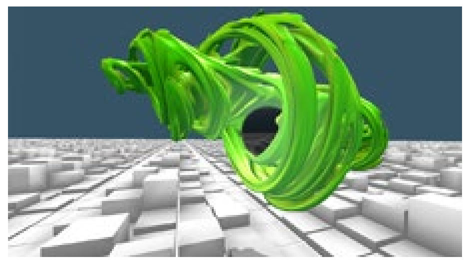
- **Problems with classic ray tracing**
  - **Not realistic**
  - **View-dependent**

- **Radiosity (1984)**
  - **Global illumination in diffuse scenes**

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

Interactive Ray Tracing Kernels

- **OptiX, Nvidia**
  - **Utilize GPU computing architectures and CUDA**

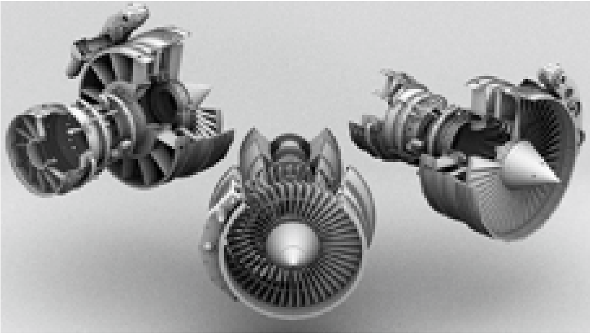




- **Embree, Intel**
  - **Utilize CPUs (multi-threaded and SIMD)**



PA

- Get to know OptiX or Embree
  - Download, and compile either one of those two methods
  - Or just use precompiled ones
  - Try out a few scenes
  - Upload images of those scenes in KLMS
- Deadline
  - Check the KLMS
- Note
  - Easy one, but start early



Class Objectives were:

- Understand a basic ray tracing
- Know its acceleration data structure and how to use it

Homework

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

Next Time

- **Rendering equation**