

---

**CS482:**  
**Ray Tracing**

---

**Sung-Eui Yoon**  
(윤성익)

**Course URL:**  
**<http://sglab.kaist.ac.kr/~sungeui/ICG/>**

**KAIST**

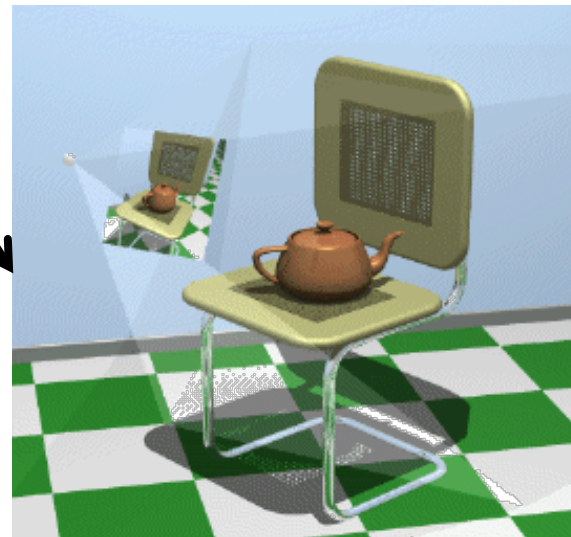
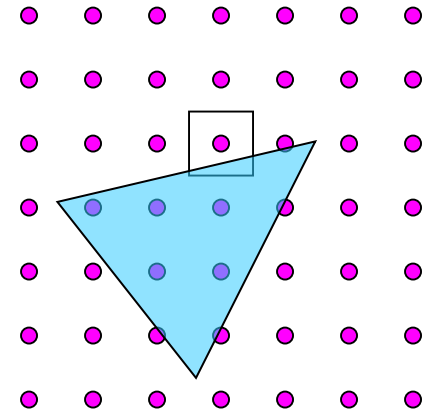


# Class Objectives (Ch. 10)

---

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

# The Classic Rendering Pipeline



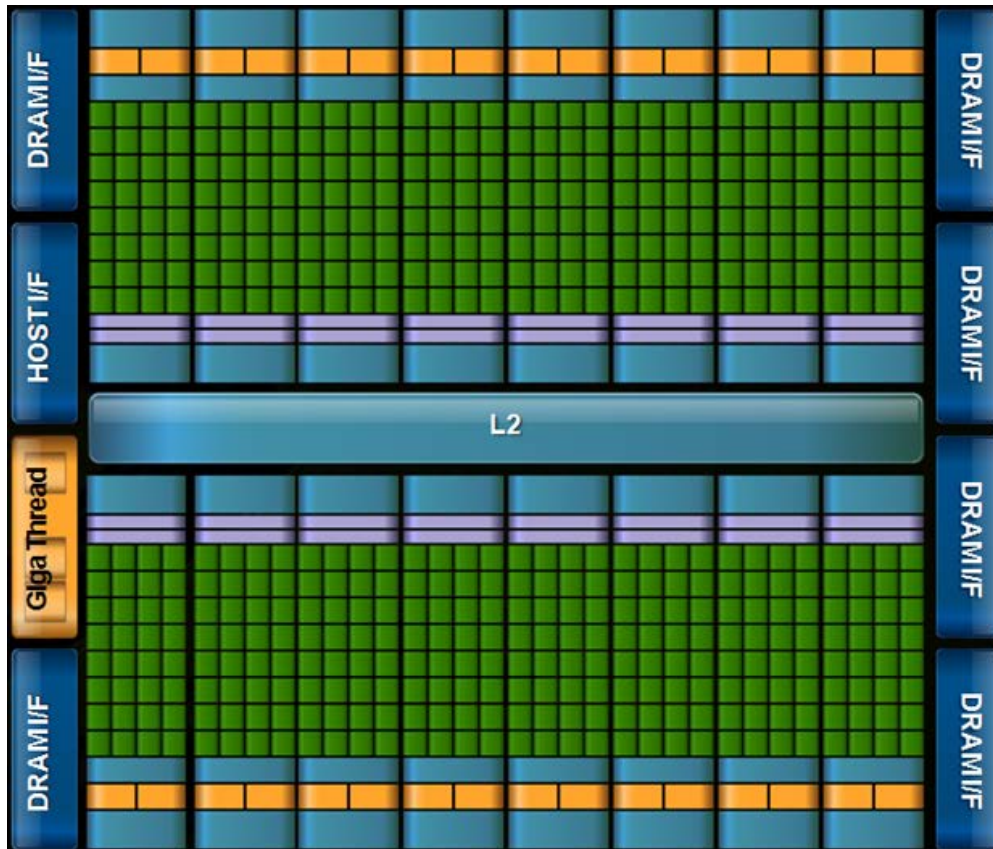
# Why we are using rasterization?

---

- Efficiency
- Reasonably quality

# Fermi GPU Architecture

16 SM (streaming processors)



512 CUDA cores

Memory interfaces

# Turing Architecture, 2018

- Aims to combine shade, compute, ray tracing, and AI

**LIVE**

Architecture	Shader   Compute	Tensor Core	RT Core	Specs
PASCAL	13 TFLOPS FP32 50 TOPS INT8	-	-	11.8 Billion xtors   471 mm <sup>2</sup>   24 GB 10GHz
TURING	16 TFLOPS + 16 TIPS	125 TFLOPS FP16 250 TOPS INT8 500 TOPS INT4	10 Giga Rays/Sec	18.6 Billion xtors   754 mm <sup>2</sup>   48+48 GHz

**AI**

# Where Rasterization Is

---

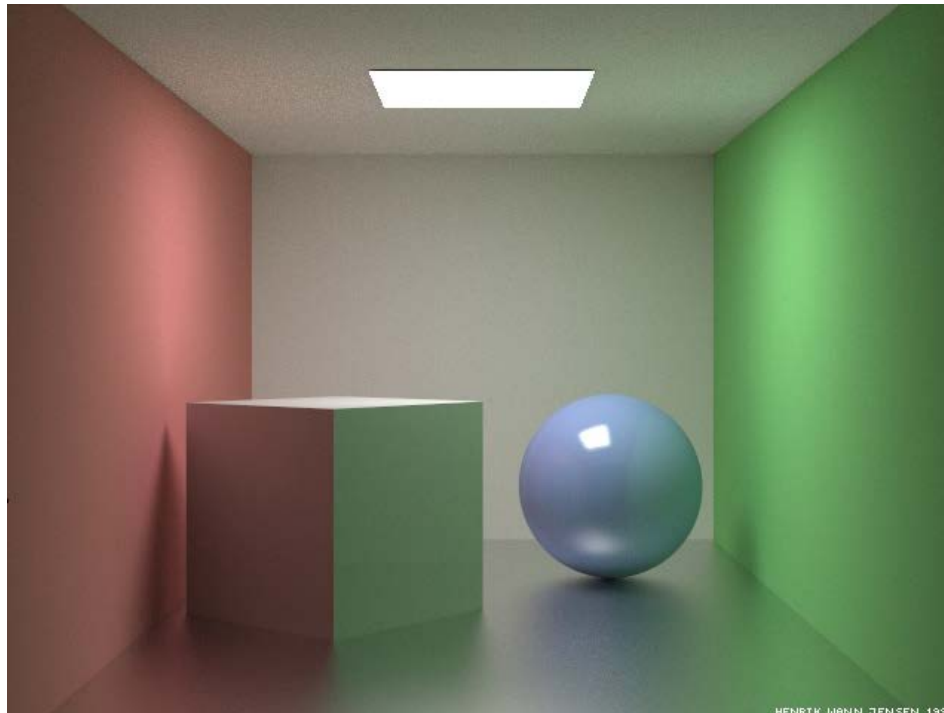


From Battlefield: Bad Company, EA Digital Illusions  
CE AB

# But what about other visual cues?

---

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

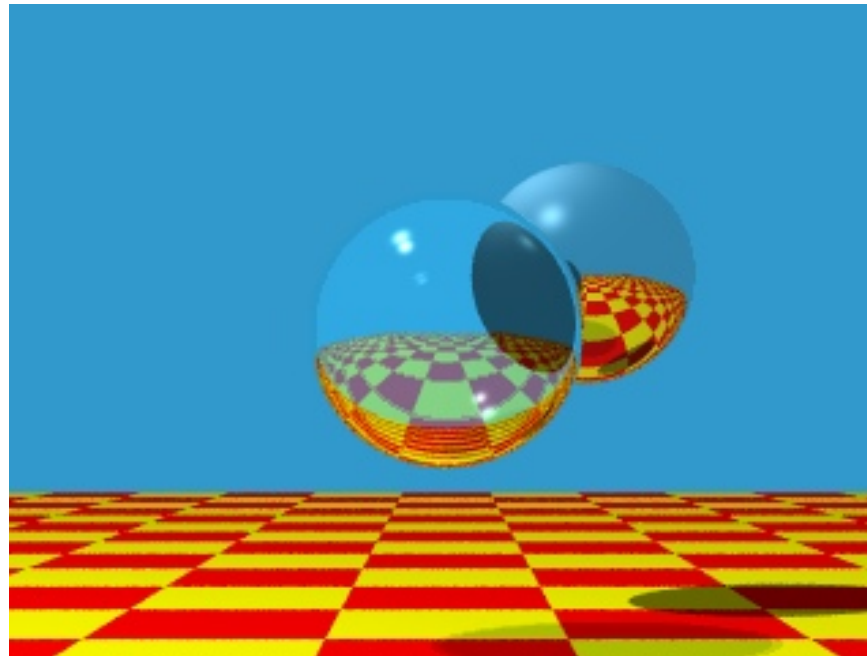




# Recursive Ray Casting

---

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



# Ray Casting and Ray Tracing

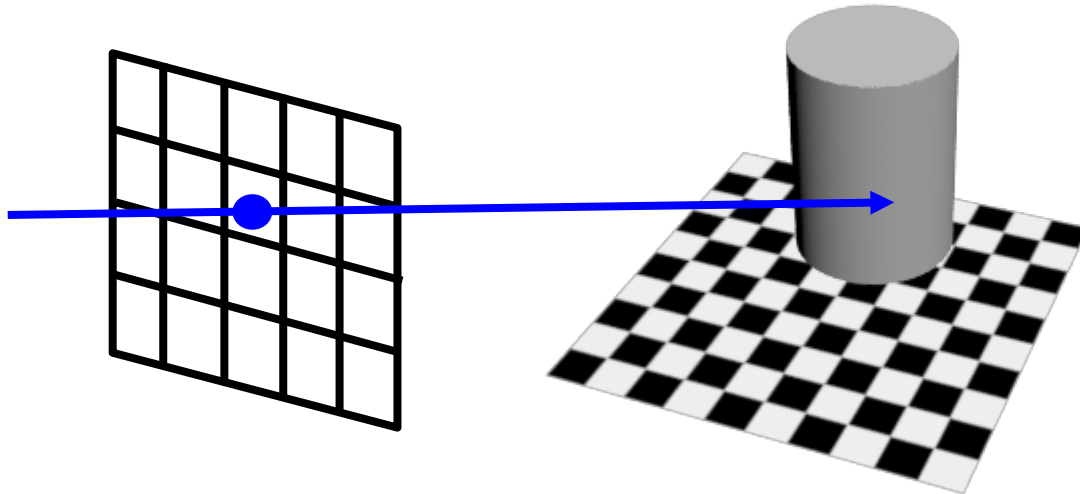
---

- 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

---

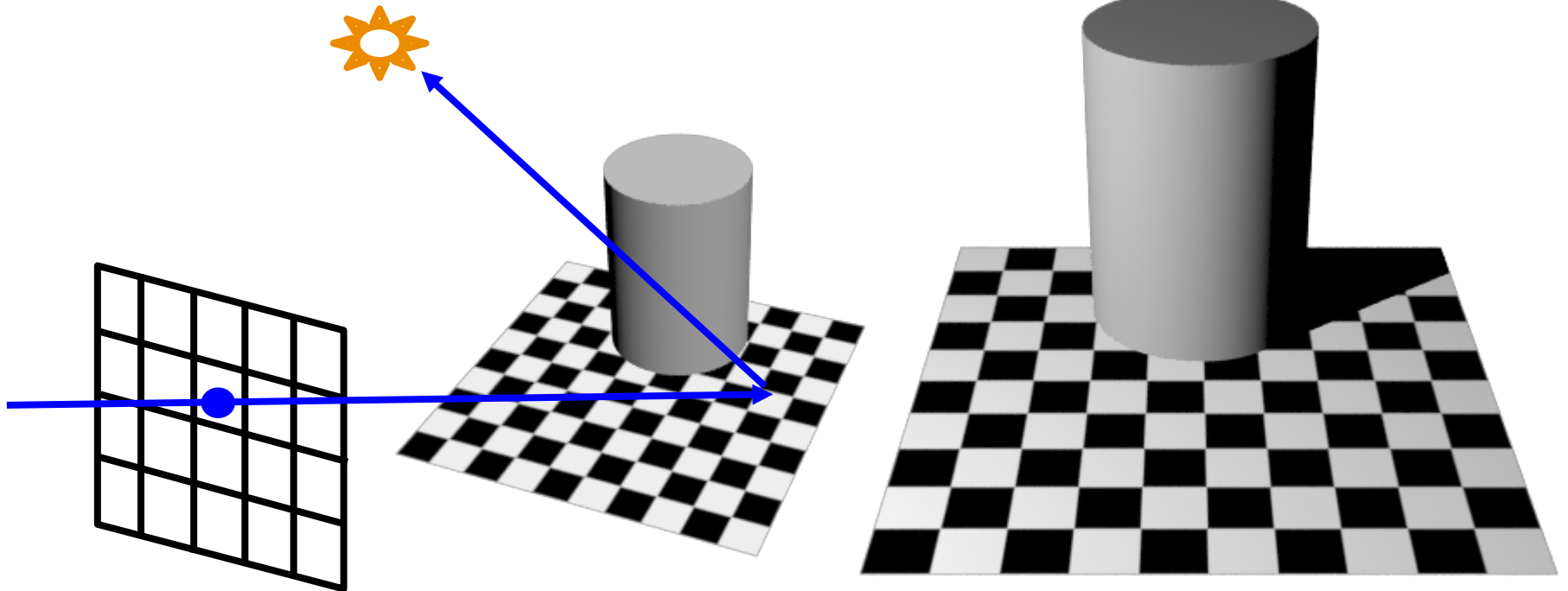
- Rays are cast from the eye point through each pixel in the image



# Shadows

---

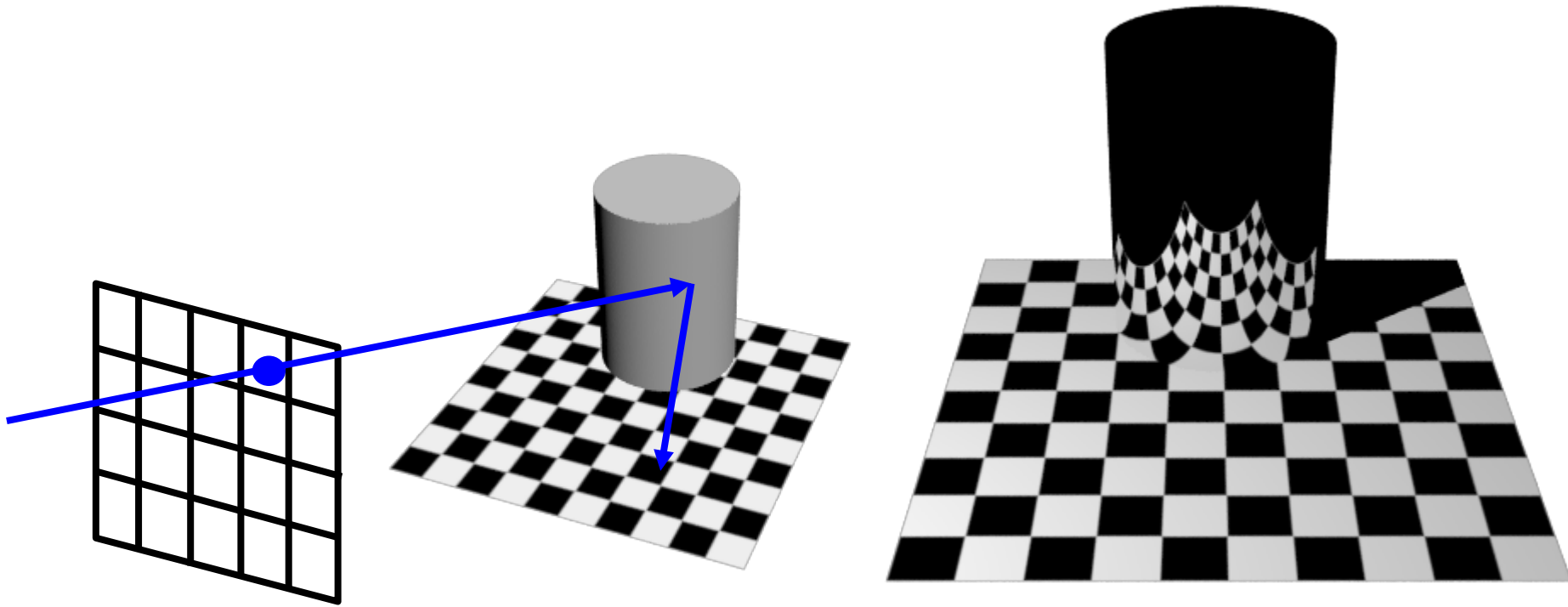
- Cast ray from the intersection point to each light source
  - Shadow rays



# Reflections

---

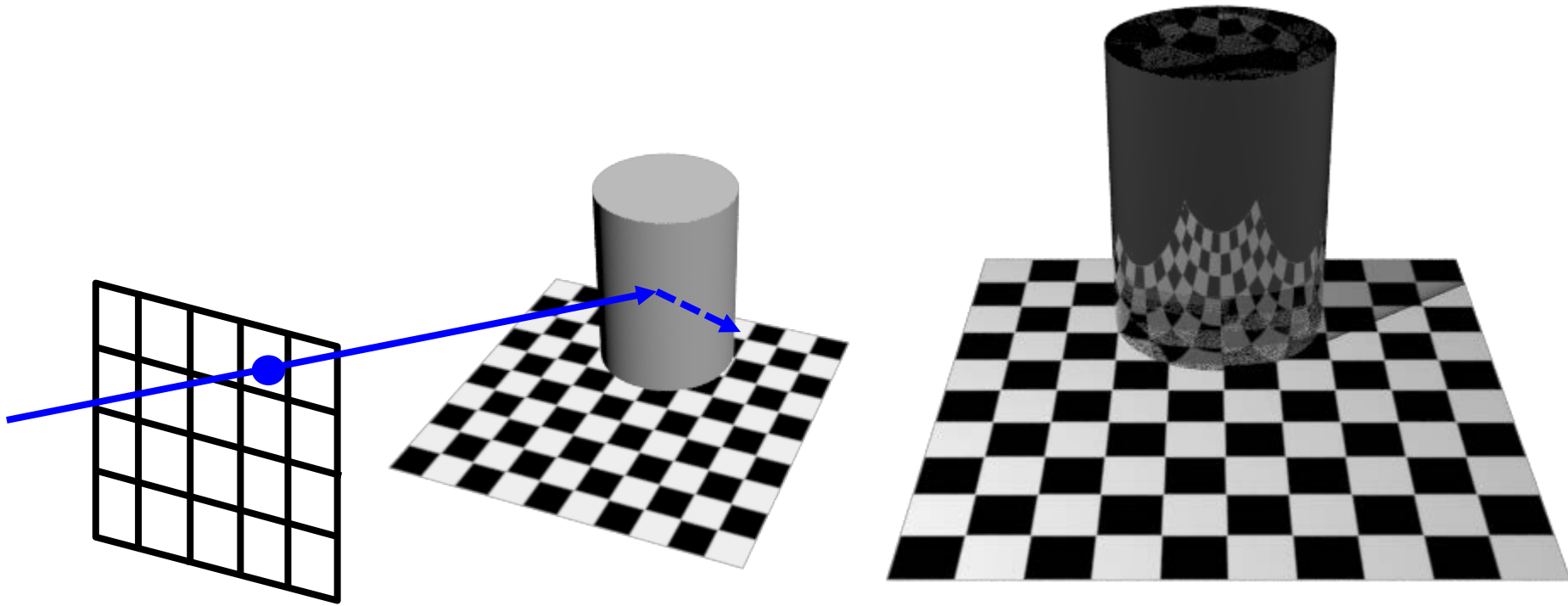
- If object specular, cast secondary reflected rays



# Refractions

---

- If object transparent, cast secondary refracted rays



# An Improved Illumination Model [Whitted 80]

- Phong illumination model

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

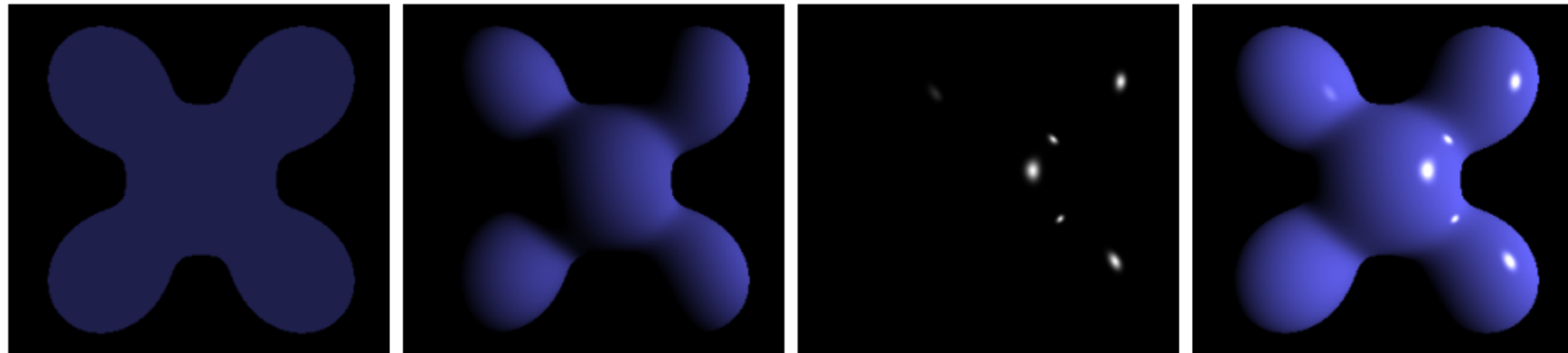
- Whitted model

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

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

# OpenGL's Illumination Model

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



Ambient

+

Diffuse

+

Specular

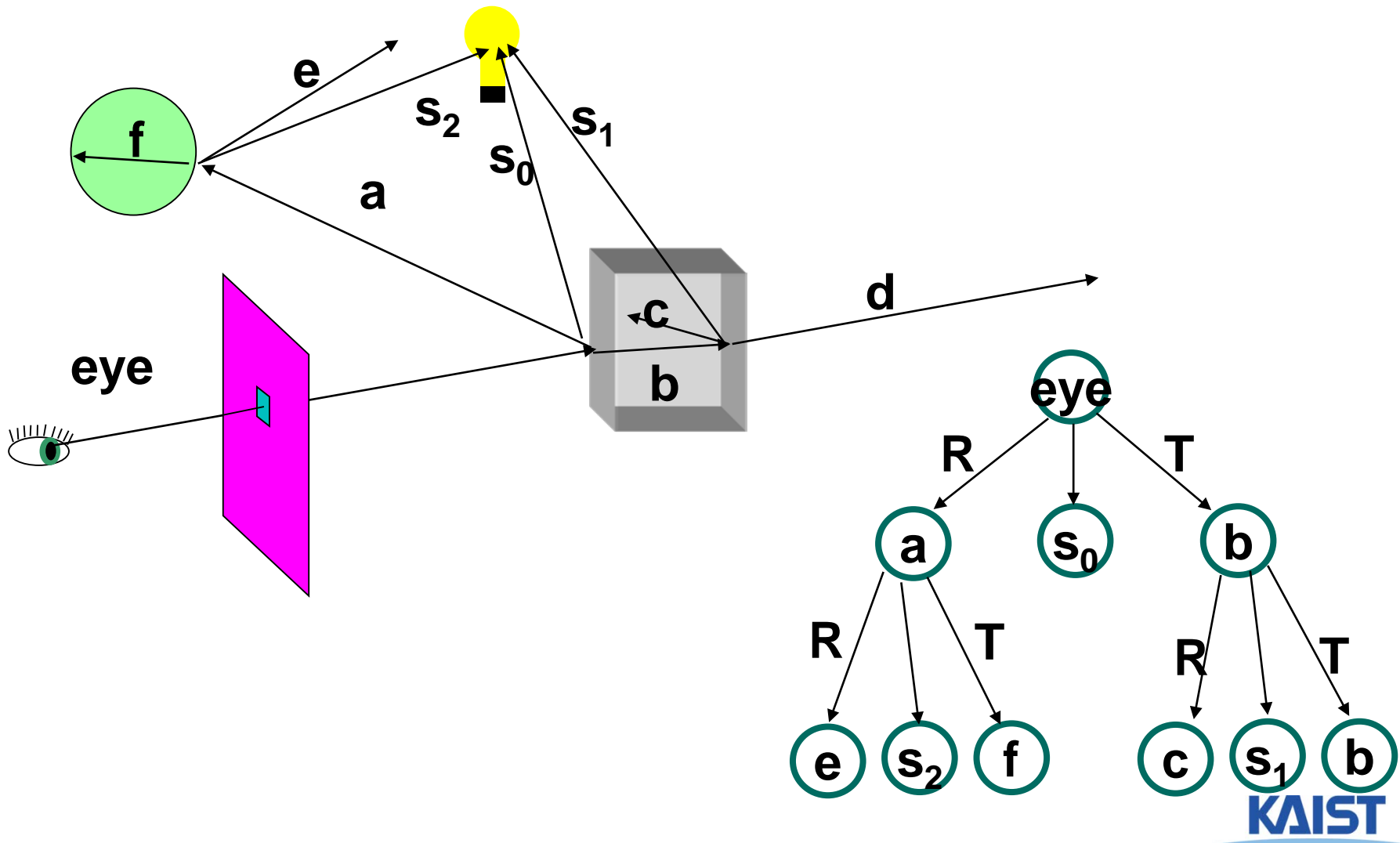
=

Phong Reflection

From Wikipedia



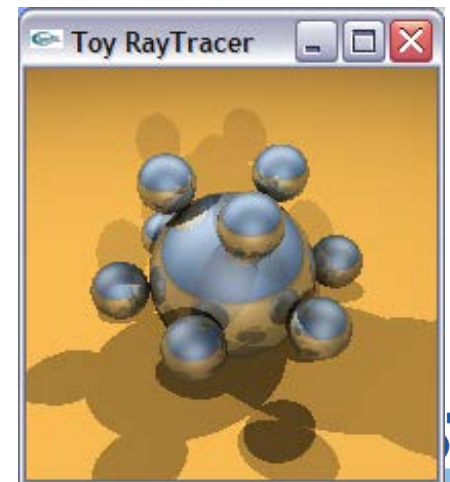
# Ray Tree



# 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)

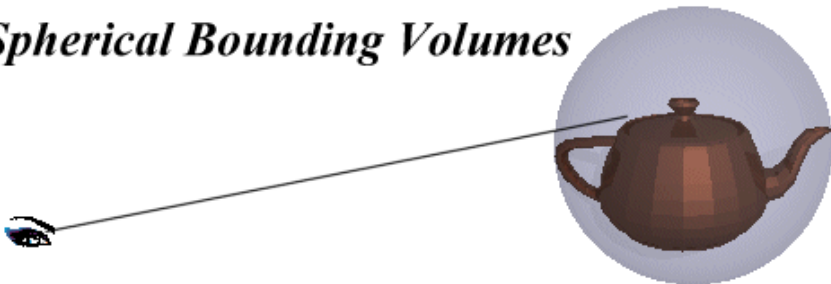


# Bounding Volumes

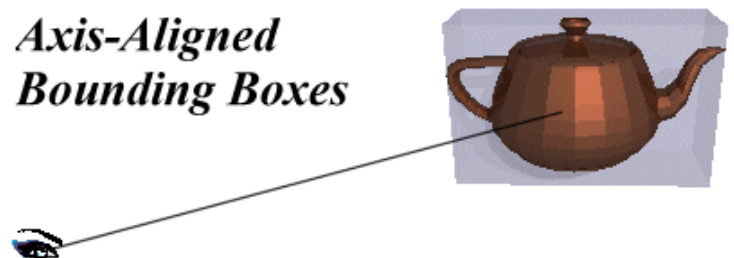
---

- **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

*Spherical Bounding Volumes*



*Axis-Aligned Bounding Boxes*

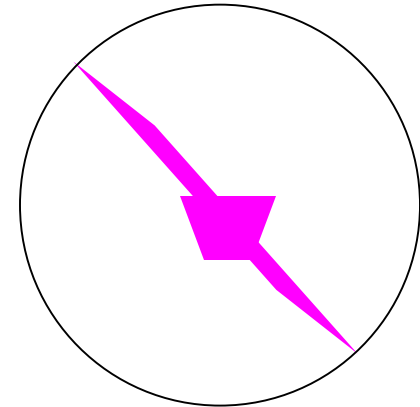


# Bounding Volumes

---

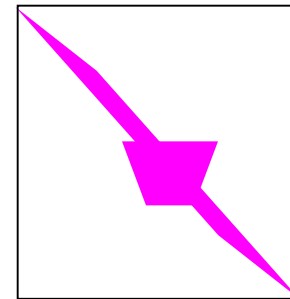
- **Sphere [Whitted80]**

- Cheap to compute
- Cheap test
- Potentially very bad fit



- **Axis-Aligned Bounding Box**

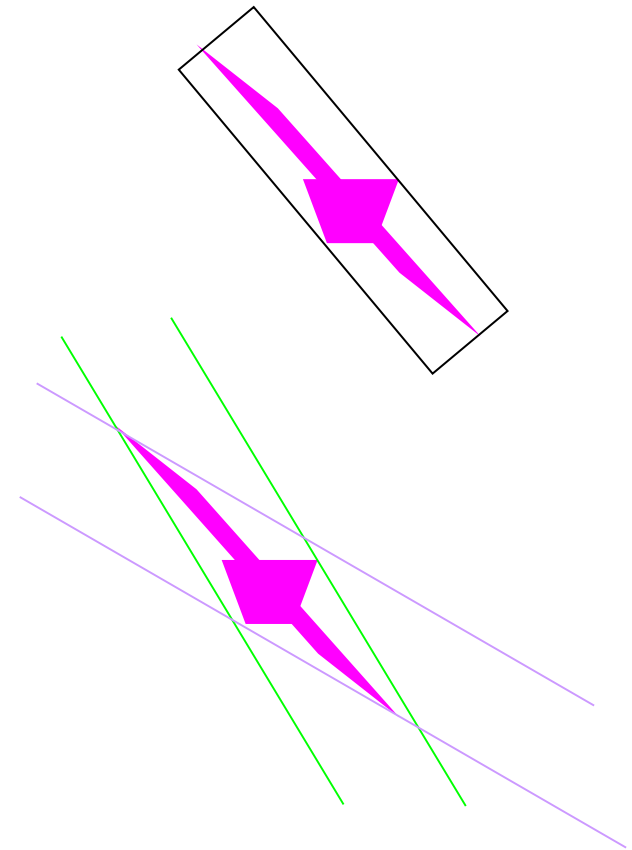
- Very cheap to compute
- Cheap test
- Tighter than sphere



# Bounding Volumes

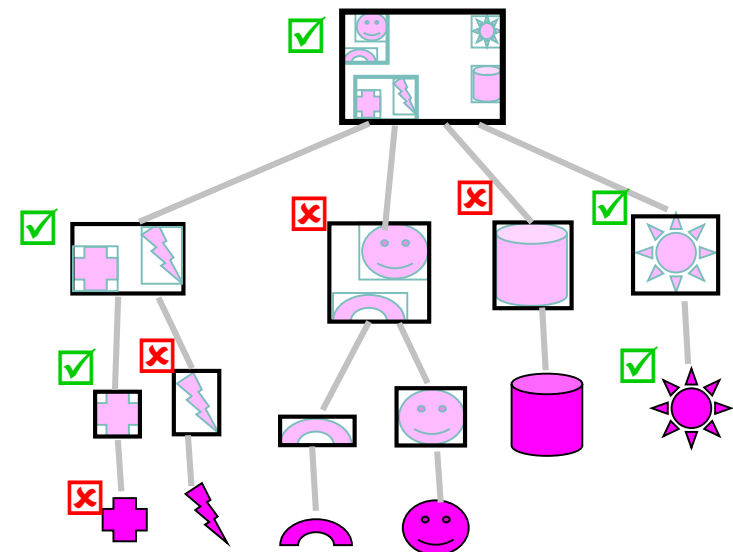
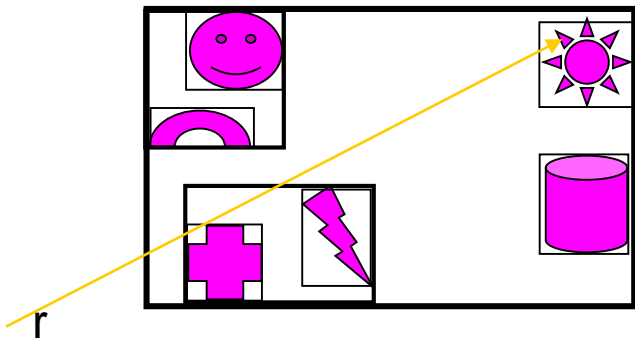
---

- **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



# 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



# Test-Of-Time 2006 Award

## High-Performance Graphics 2015

Los Angeles, August 7-9, 2015

Home

Full Program

CFP

Registration

Accommodations

Venue

Submissions

Organization



## RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs

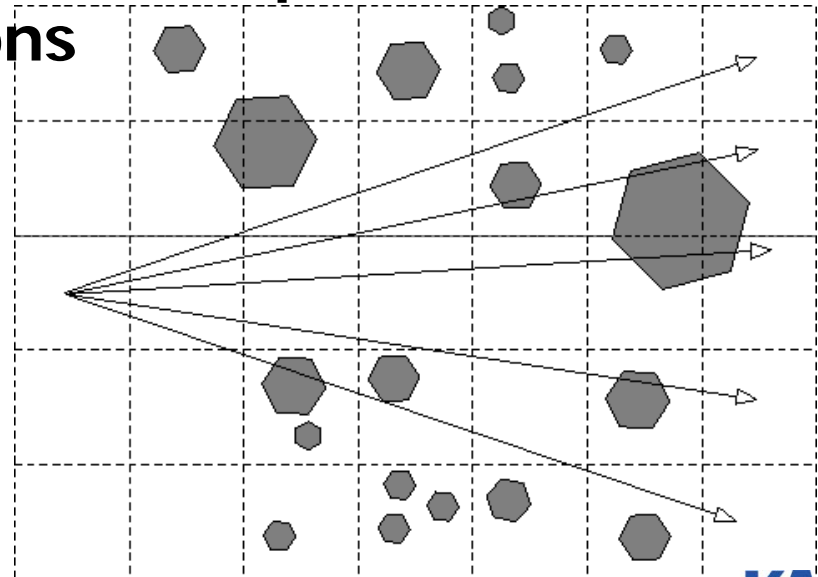
Christian Lauterbach, [Sung-eui Yoon](#),  
David Tuft, Dinesh Manocha  
IEEE Interactive Ray Tracing, 2006



# 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





# Ray Tracing with kd-tree or BVHs

---

- **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

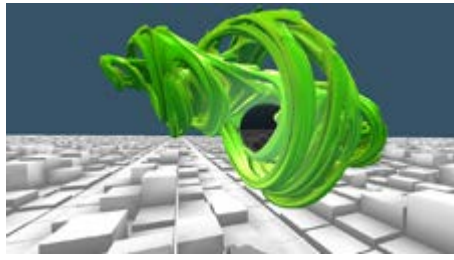
---

- **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

# Interactive Ray Tracing Kernels

- **OptiX, Nvidia**

- Utilize GPU computing architectures and CUDA



- **Embree, Intel**

- Utilize CPUs (multi-threaded and SIMD)



# PA1

---

- **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**
  - 11:59pm on 18<sup>th</sup> of Sep.
- **Note**
  - Easy one, but start early



# Homework

---

- **Go over the next lecture slides before the class**
- **Watch 2 paper (or 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.**

# 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 typical questions
  - 2 for questions that have some thoughts or surprise me
- **Write a question more than 4 times on Sep./Oct.**
  - Online submission is available at the course webpage

# Class Objectives were:

---

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



# Next Time

---

- Radiosity