CS580:
Ray Tracing

Sung-Eui Yoon
(윤성의)

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

Class Objectives (Ch. 10)

- **Understand a basic ray tracing**
- **Know its acceleration data structure and how to use it**
- **Rendering book https://sgvr.kaist.ac.kr/~sungeui/render/**

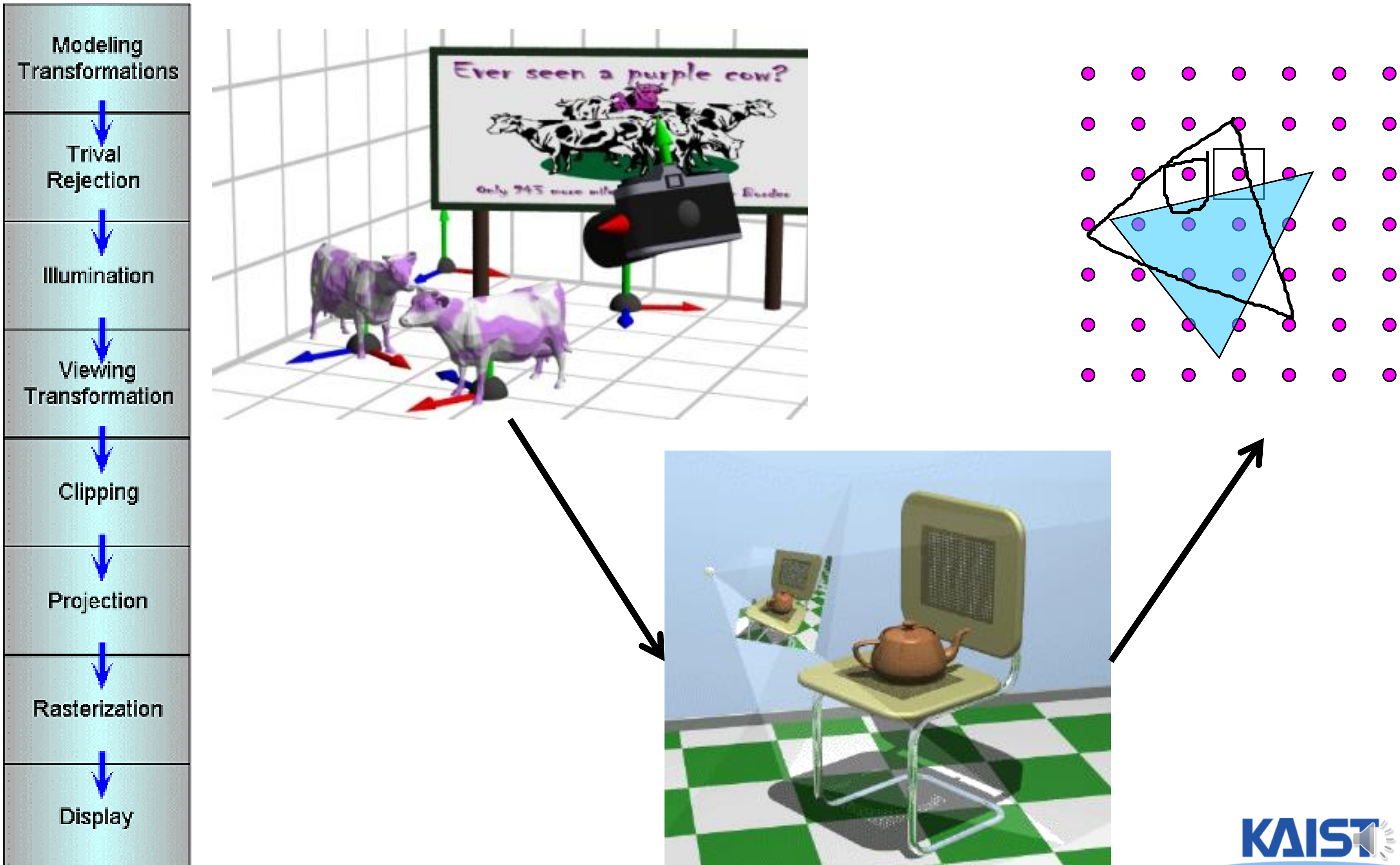
Honor Code and Etiquette

- **Collaboration encouraged, but *assignments must be your own work***
- **Cite any other's work if you use their codes**
  - **Otherwise, you will get F**
- **Classroom etiquette**
  - **Help you and your peer to focus on the class**
  - **Turn off cell phones**
  - **Arrive to the class on time**
  - **Avoid private conversations**
- **School of Computing Student Honor Code (전산학부 학생 명예규정)**
  - **https://forms.gle/TiS9LF9aT2Ymznf9A**

KAIST

Refer to short summary of under. CG
https://sgvr.kaist.ac.kr/~sungeui/GCG/

The Classic Rendering Pipeline

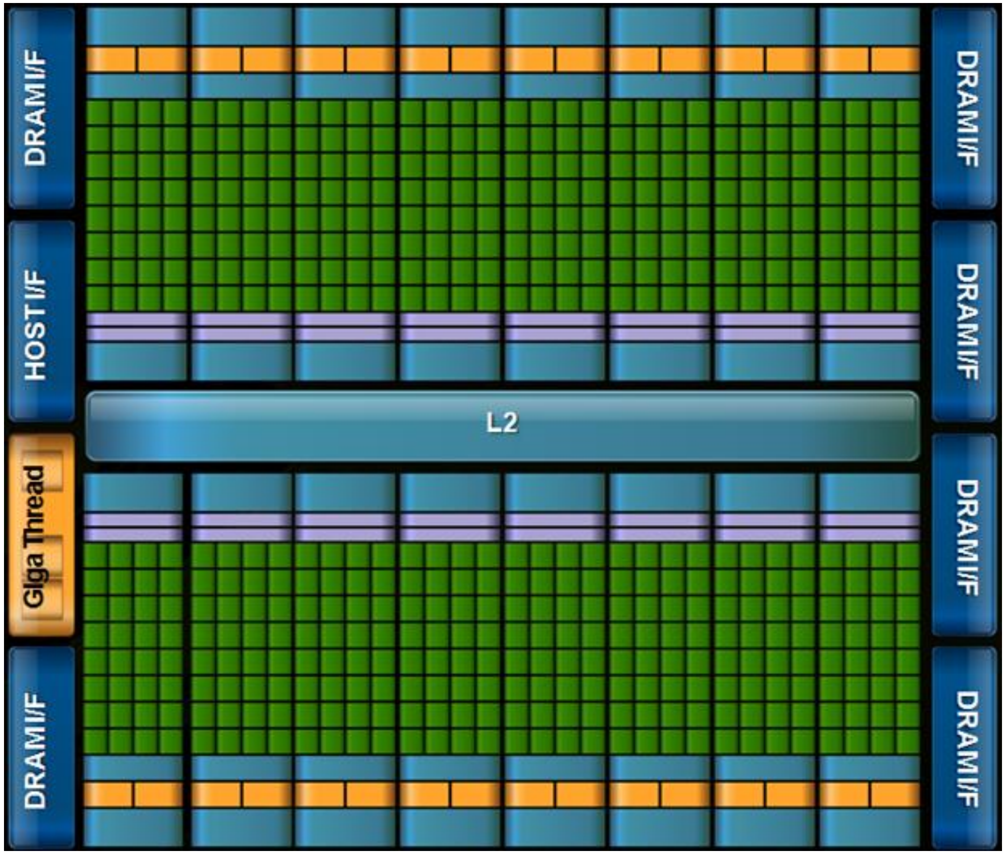


Why we are using rasterization?

- **Efficiency**
- **Reasonably quality**

Fermi GPU Architecture

16 SM (streaming processors)

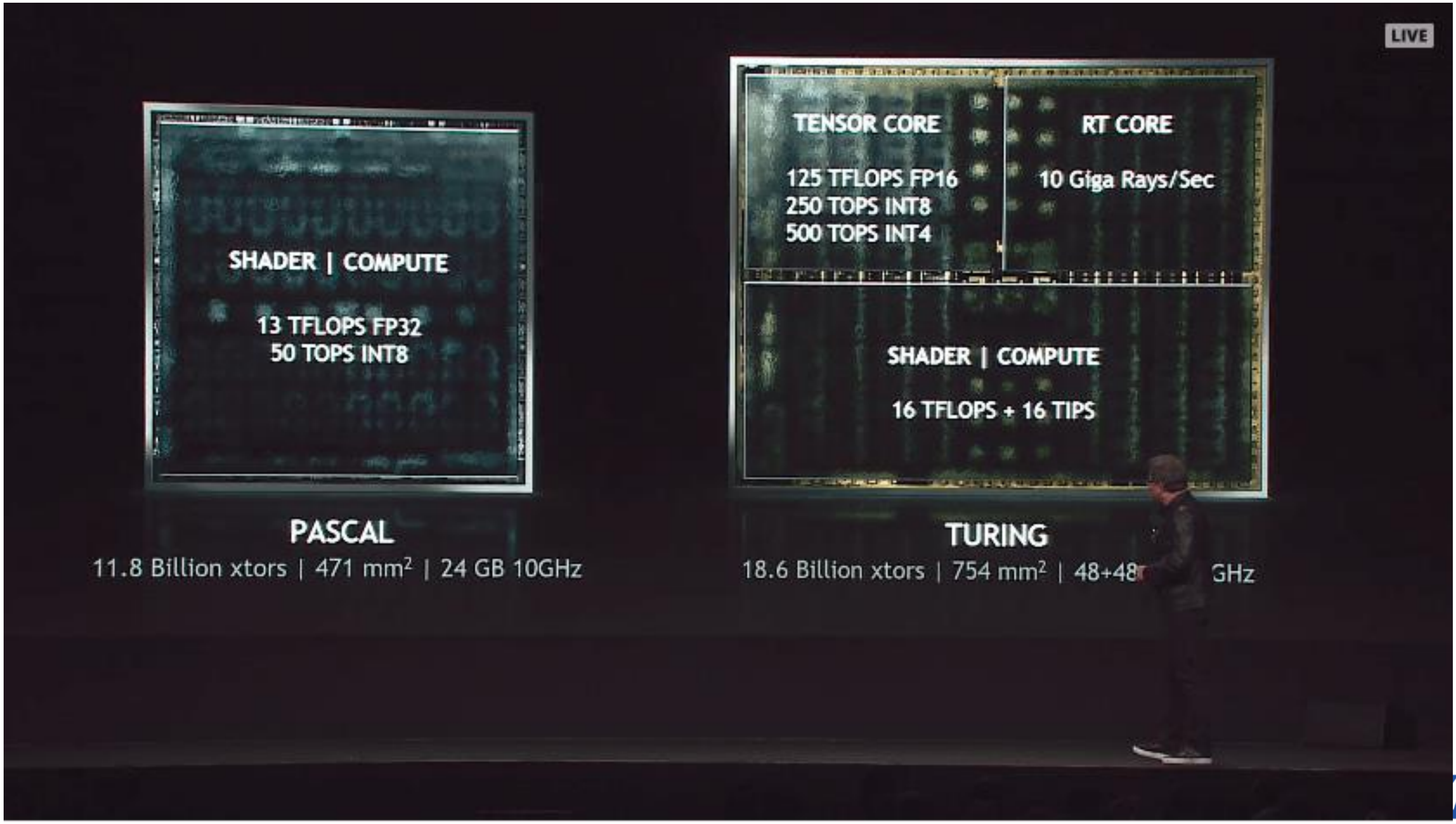


512 CUDA cores

Memory interfaces

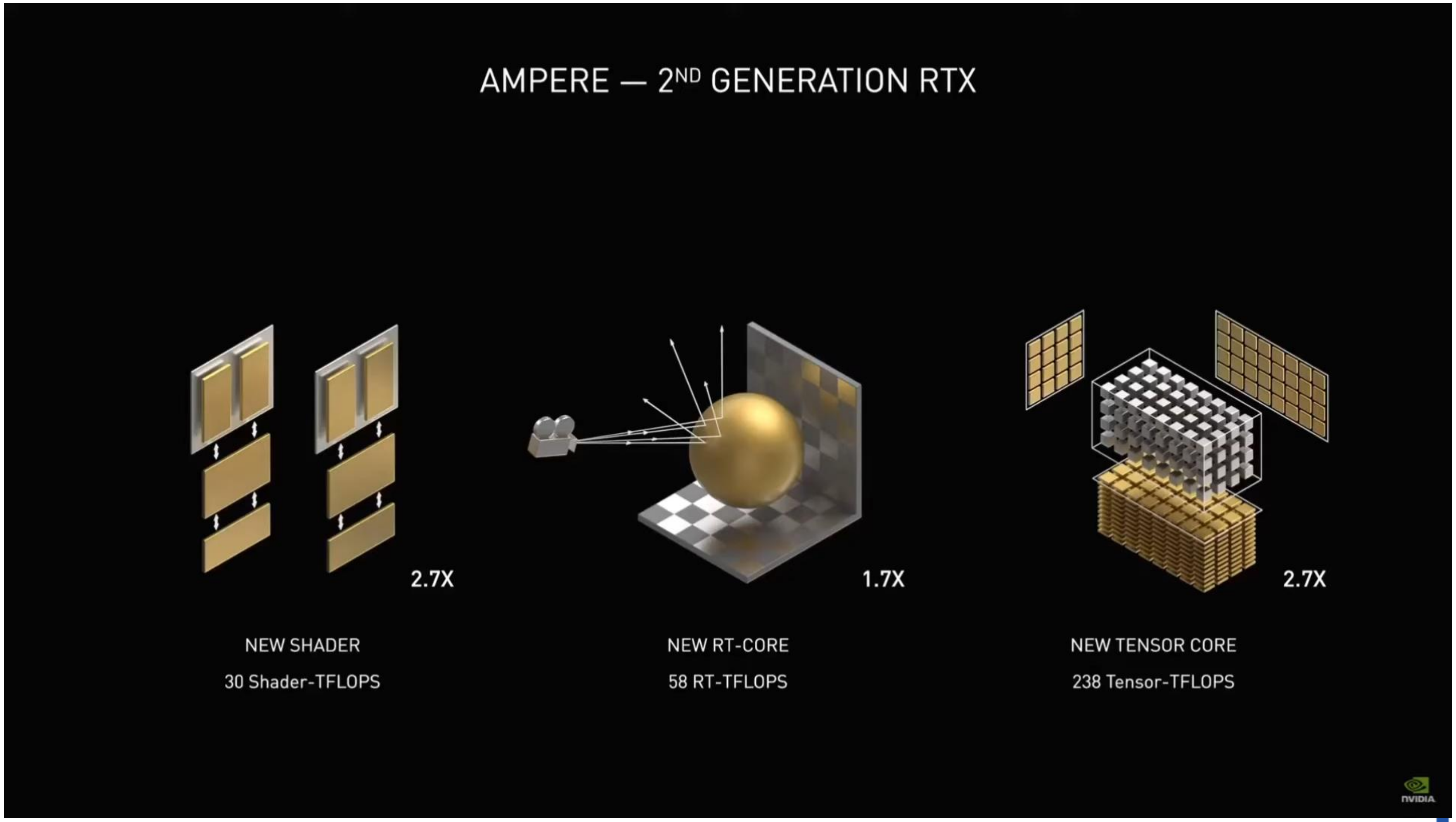
Turing Architecture, 2018

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



Ampere Architecture, 2020

- **More cores, faster computation than Turing Architecture**



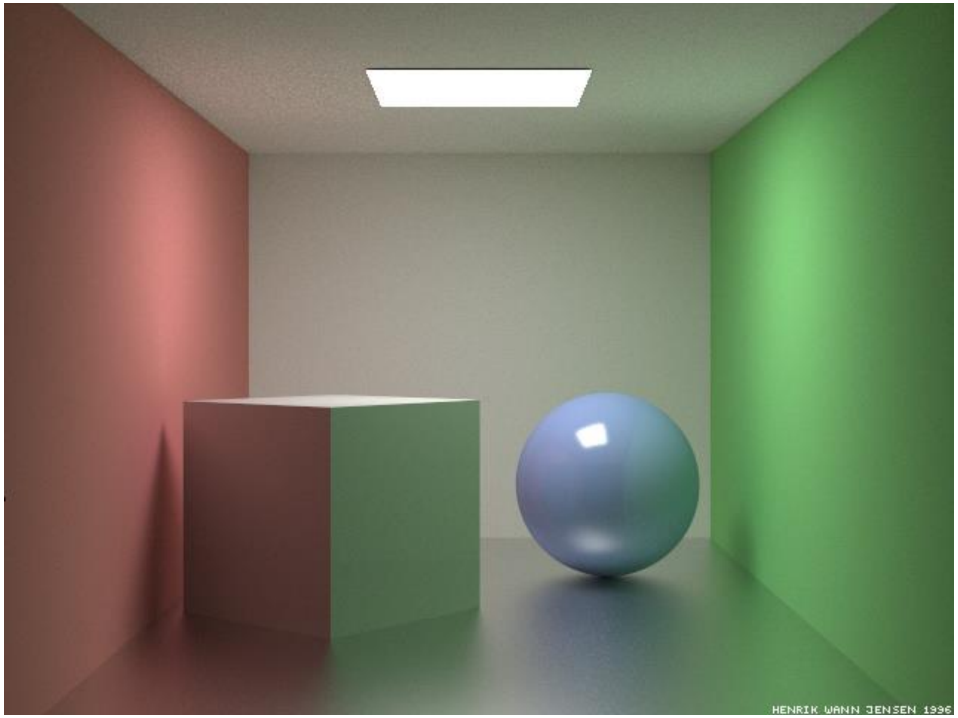
Where Rasterization Is

From Battlefield: Bad Company, EA Digital Illusions CE AB

From Eric Haines

But what about other visual cues?

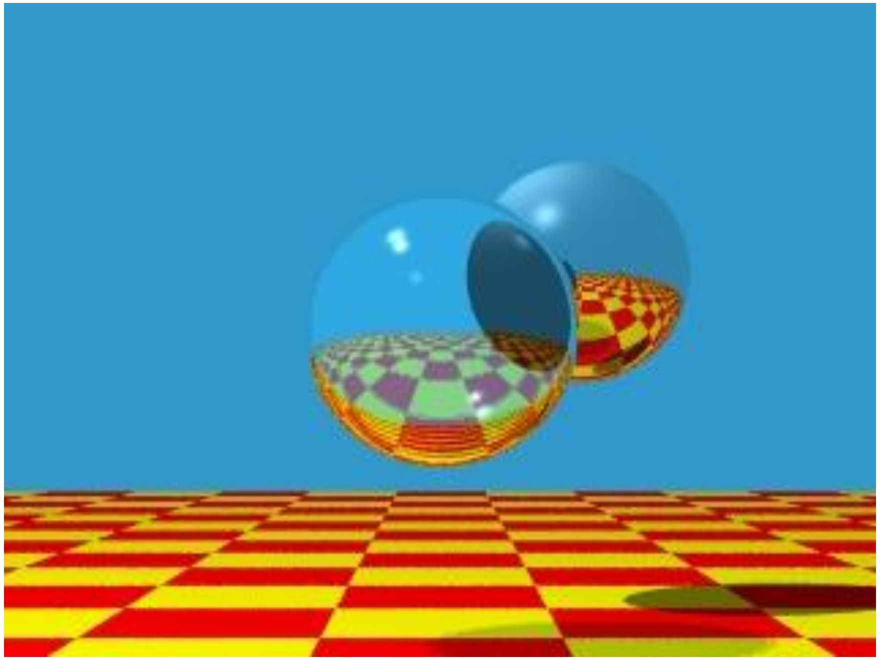
- Lighting
  - Shadows
  - Shading: glossy, transparency
- Color bleeding, etc



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**

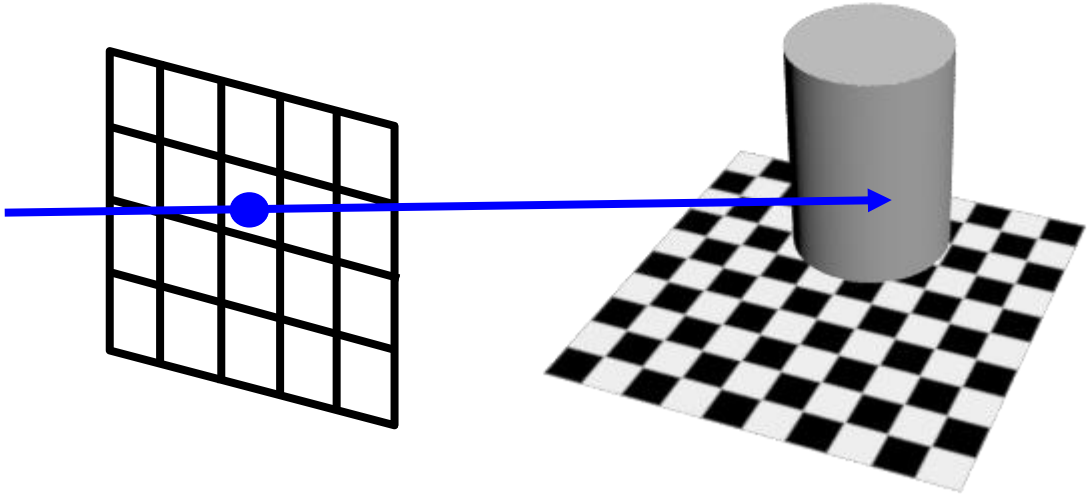


Ray Casting and Ray Tracing

- 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

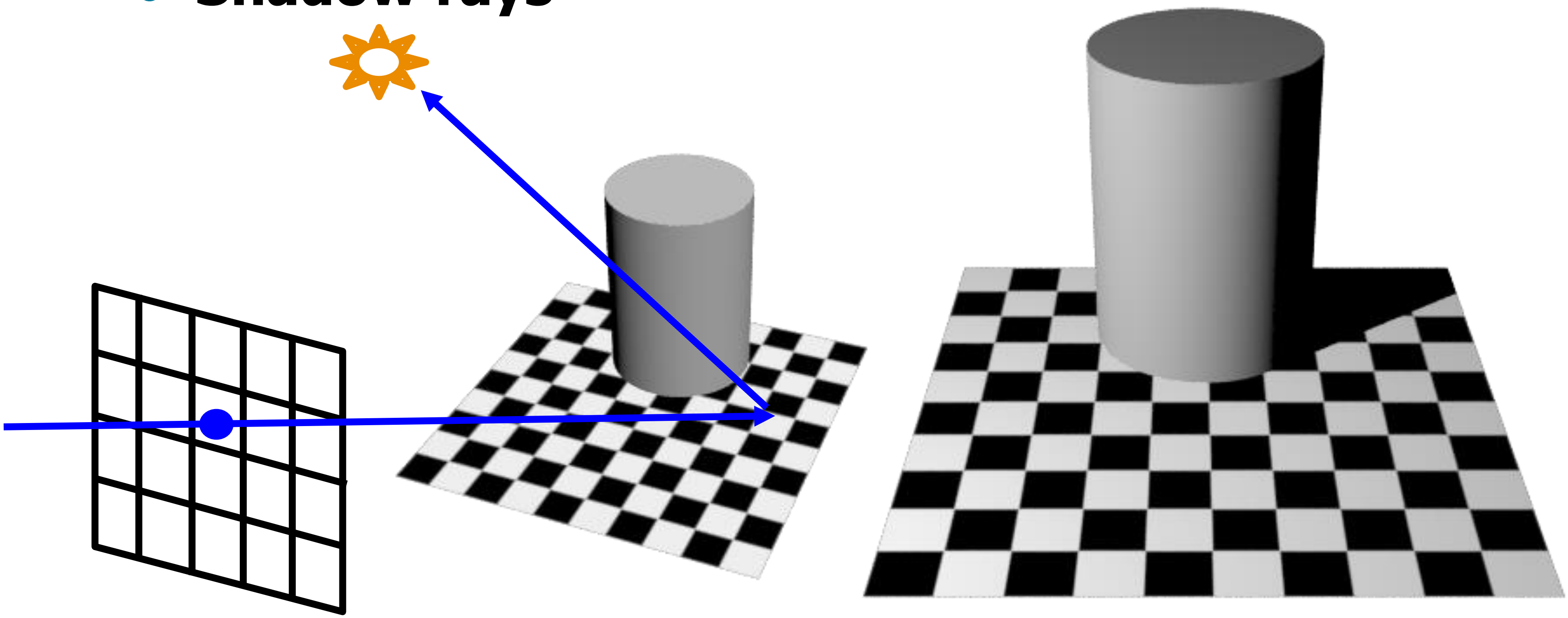
Basic Algorithms

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



Shadows

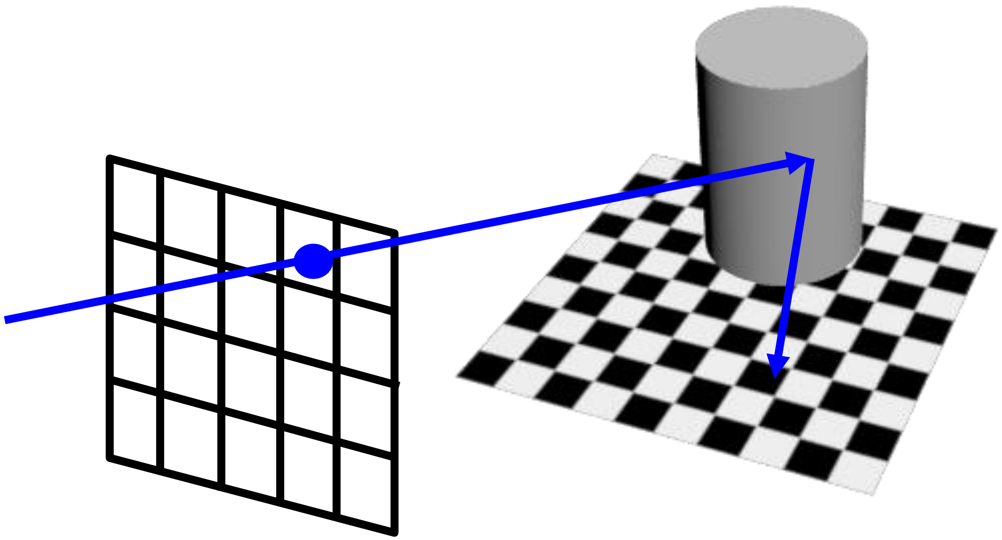
- **Cast ray from the intersection point to each light source**
  - **Shadow rays**

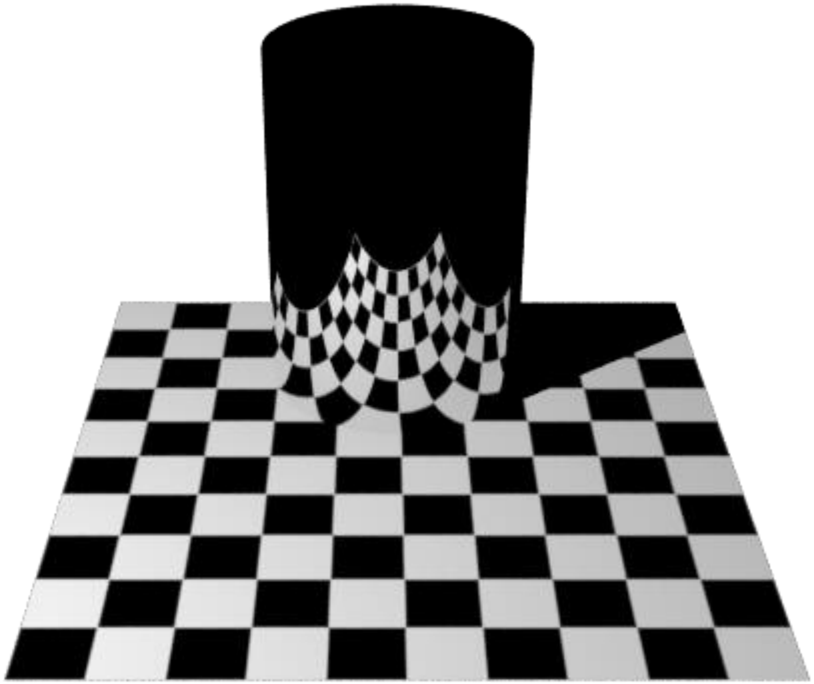


Used 3ds Max

Reflections

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

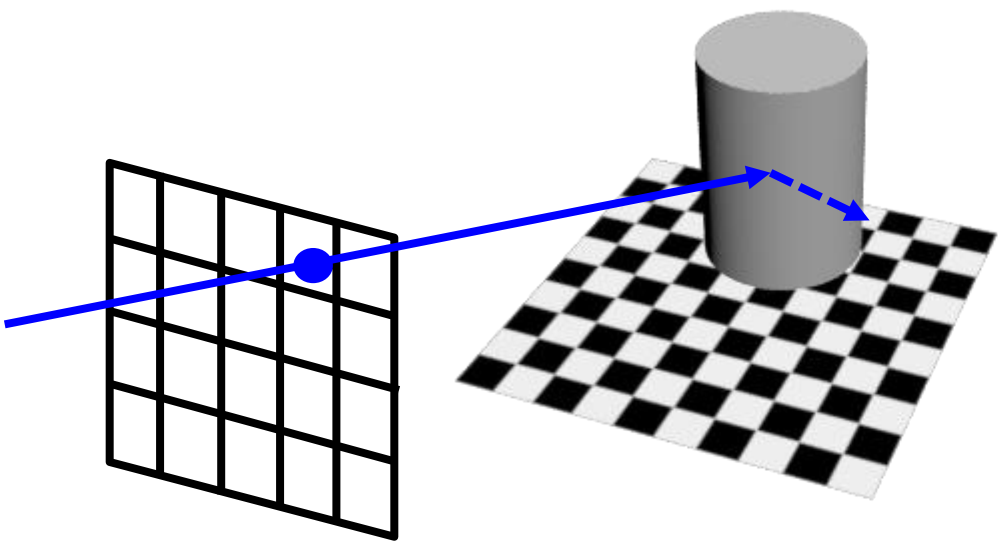


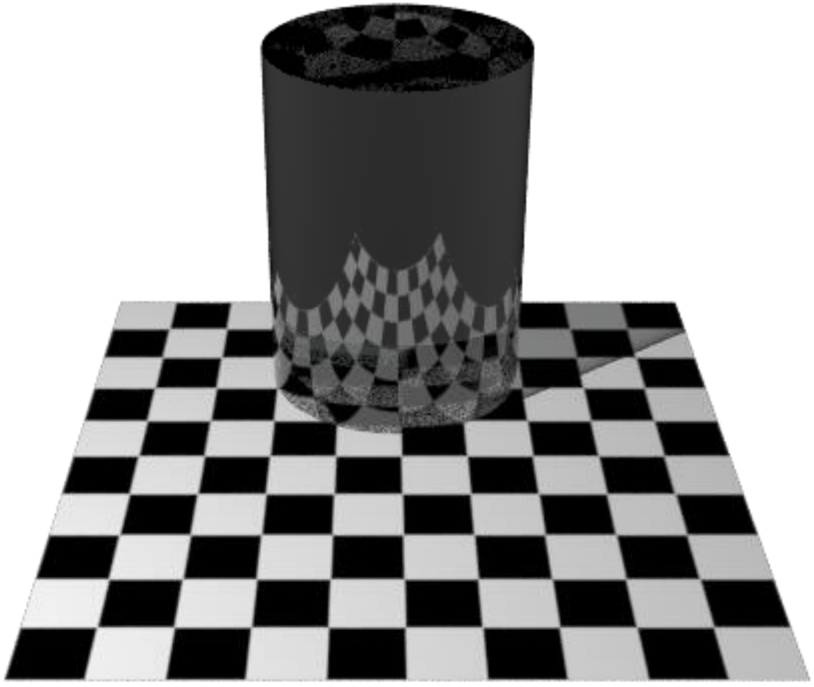


KAIST

Refractions

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





An Improved Illumination Model [Whitted 80]

- **Phong illumination model**

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

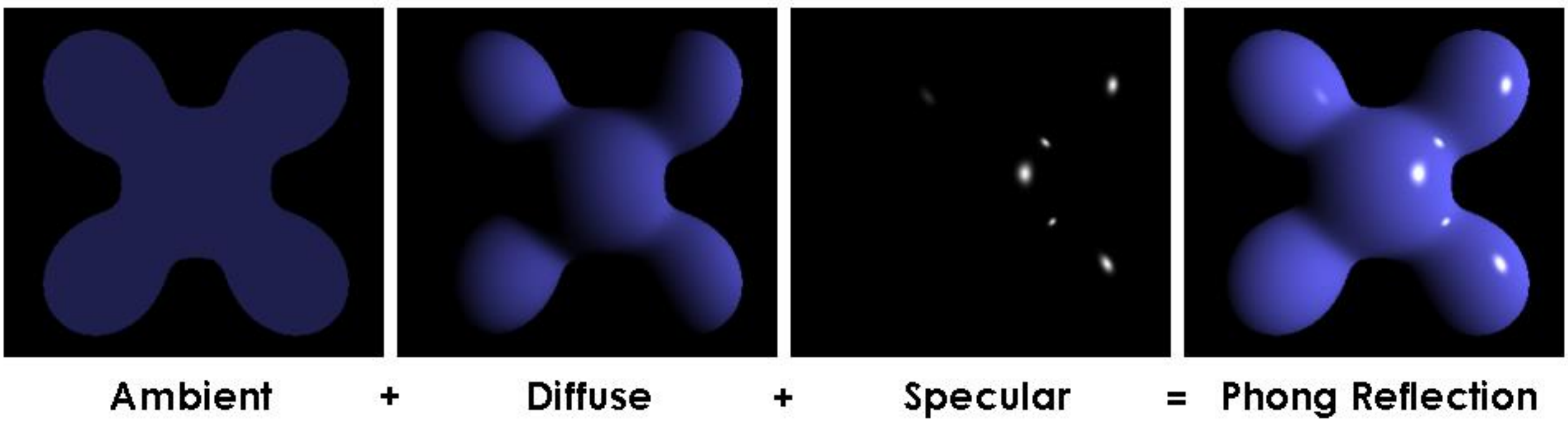
- **Whitted model**

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

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

OpenGL's Illumination Model

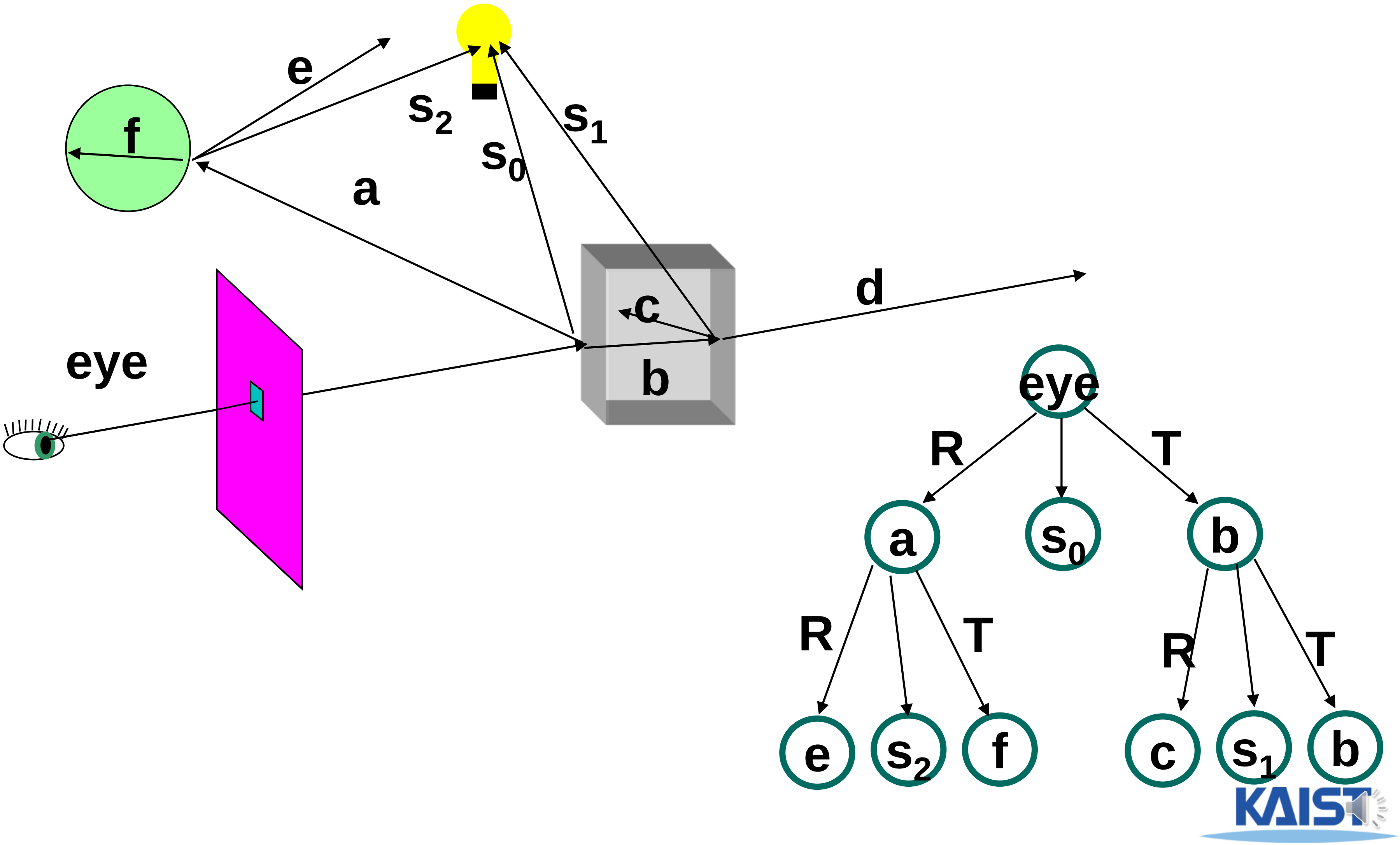
$$I_r = \sum_{j=1}^{numLights} \left(k_a^j I_a^j + k_d^j I_d^j \max((\hat{N} \bullet \hat{L}_j), 0) + k_s^j I_s^j \max((\hat{V} \bullet \hat{R})^{n_s}, 0) \right)$$



From Wikipedia

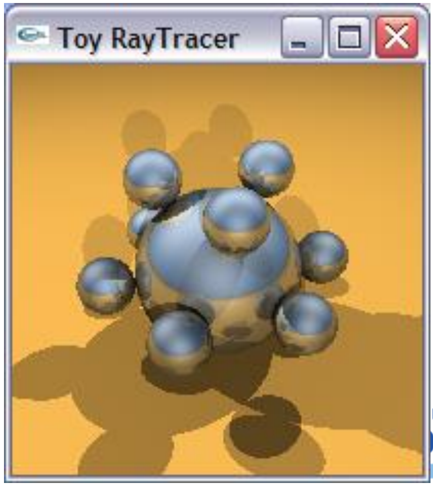
Details are available at Ch. 8 Illumination and Shading

Ray Tree



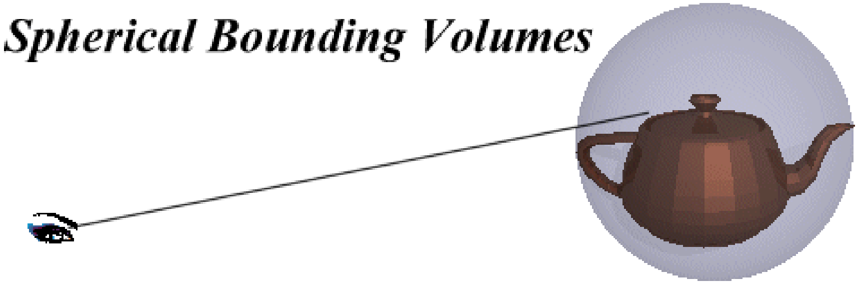
Acceleration Methods for Ray Tracing

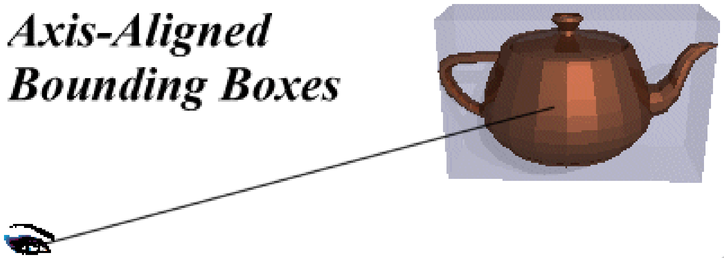
- Rendering time for a ray tracer depends on the number of ray intersection tests per pixel
  - The number of pixels X the number of primitives in the scene
- Early efforts focused on accelerating the ray-object intersection tests
  - Ray-triangle intersection tests
- More advanced methods required to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



Bounding Volumes

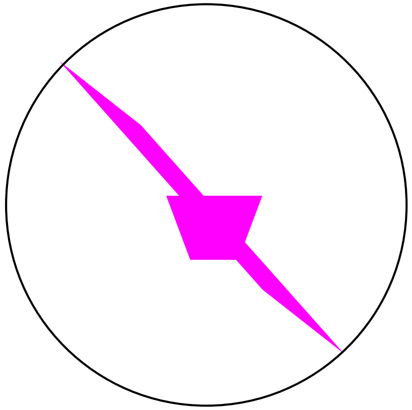
- Enclose complex objects within a simple-to-intersect objects
  - If the ray does not intersect the simple object then its contents can be ignored
  - The likelihood that it will strike the object depends on how tightly the volume surrounds the object.
- Spheres are simple, but not tight
- Axis-aligned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



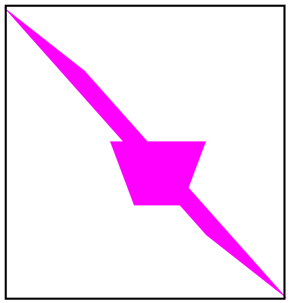


Bounding Volumes

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

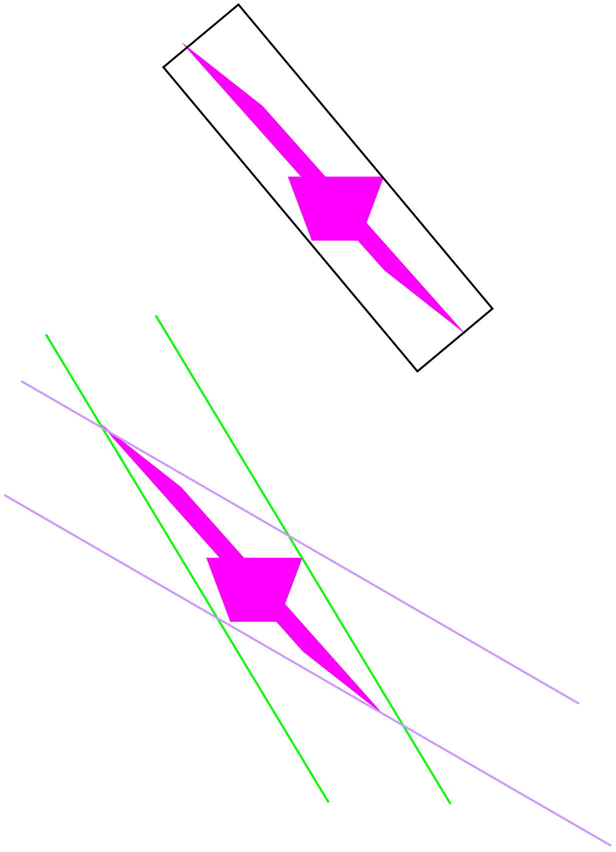


- **Axis-Aligned Bounding Box**
  - Very cheap to compute
  - Cheap test
  - Tighter than sphere



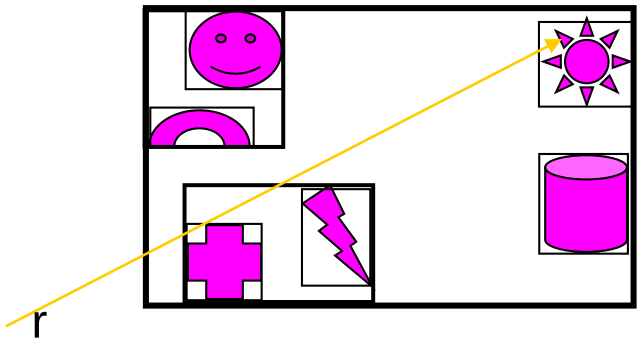
Bounding Volumes

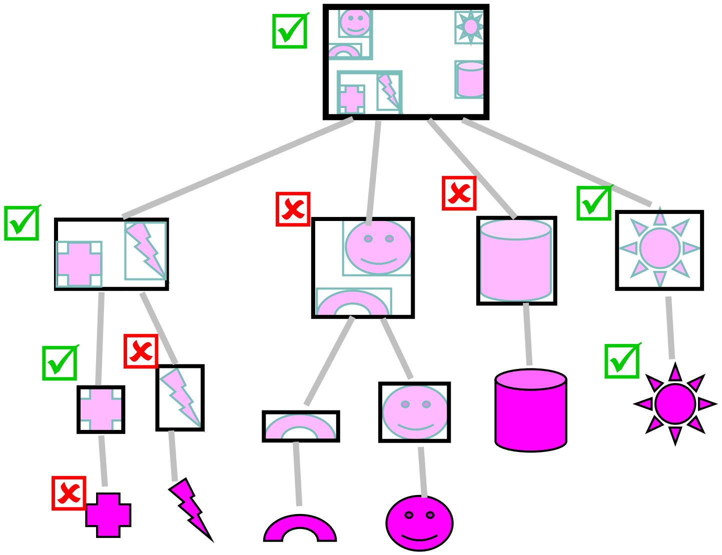
- **Oriented Bounding Box**
  - **Fairly cheap to compute**
  - **Fairly Cheap test**
  - **Generally fairly tight**
- **Slabs / K-dops**
  - **More expensive to compute**
  - **Fairly cheap test**
  - **Can be tighter than OBB**



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

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

Christian Lauterbach, Sung-eui Yoon, David Tuft, Dinesh Manocha

IEEE Interactive Ray Tracing, 2006

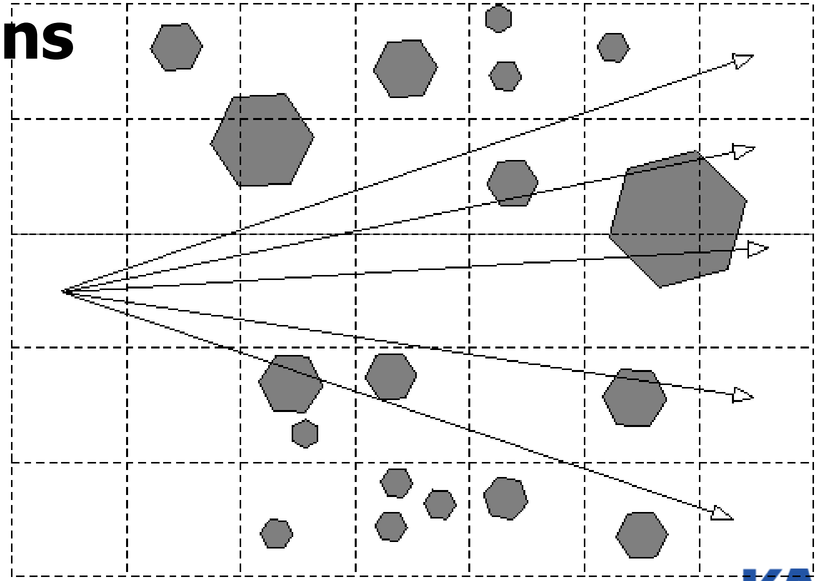
RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs

Christian Lauterbach* University of North Carolina at Chapel Hill
Sung-Eui Yoon† Lawrence Livermore National Laboratory
David Tuft‡ University of North Carolina at Chapel Hill
Dinesh Manocha§ University of North Carolina at Chapel Hill

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**



Classic Ray Tracing

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

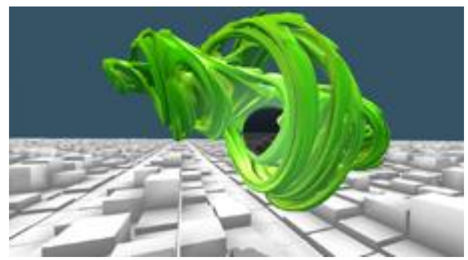
- Cons:
  - Simple light model, material, and light propagation
  - Not a complete solution
  - Hard to accelerate with special-purpose H/W

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
  - Check the KLMS
- 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 Mon. class
  - Just one paragraph for each summary

Example:

Title: XXX XXXX XXXX, Conf: XXX, Year: XXX

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.

Class Objectives were:

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

Next Time

- **Radiosity**