CS580:
Ray Tracing

Sung-Eui Yoon
(윤성의)

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

Class Objectives (Ch. 10)

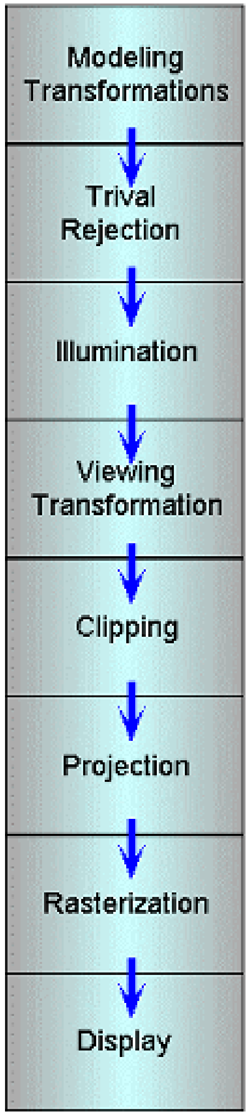
- Understand a basic ray tracing
- Know its acceleration data structure and how to use it
- Rendering book https://sgvr.kaist.ac.kr/~sungeui/render/

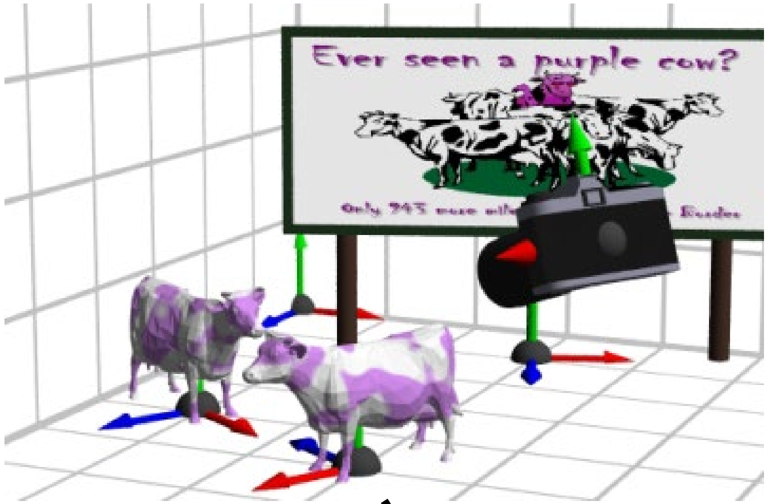
Honor Code and Etiquette

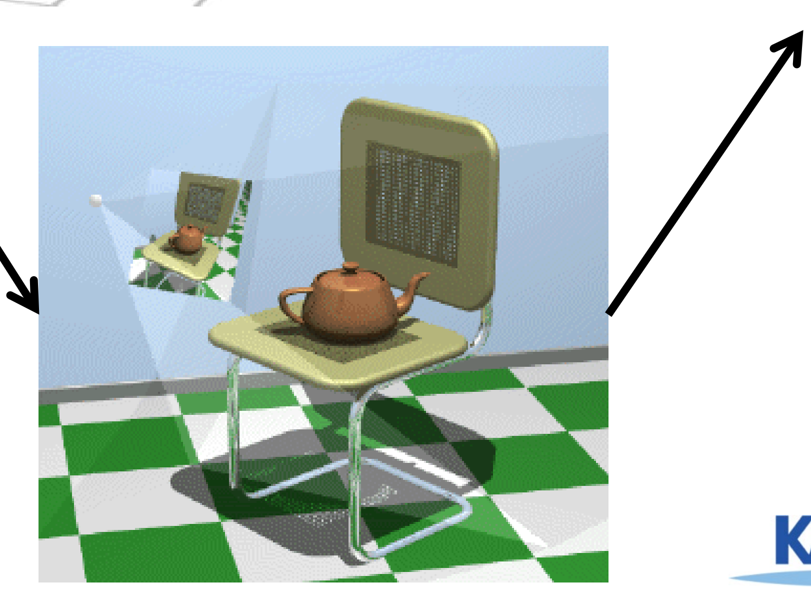
- **Collaboration encouraged, but *assignments must be your own work***
- **Cite any other's work if you use their codes**
  - **Otherwise, you will get F**
- **Classroom etiquette**
  - **Help you and your peer to focus on the class**
  - **Turn off cell phones**
  - **Arrive to the class on time**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

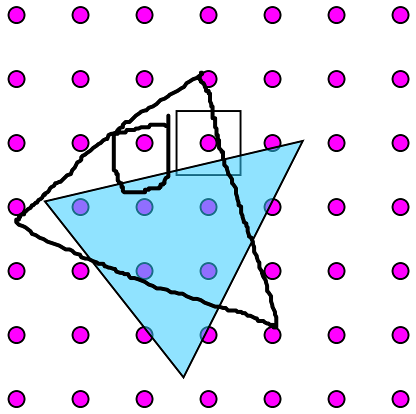
Refer to Part I of my book
https://sgvr.kaist.ac.kr/~sungeui/GCG/

The Classic Rendering Pipeline







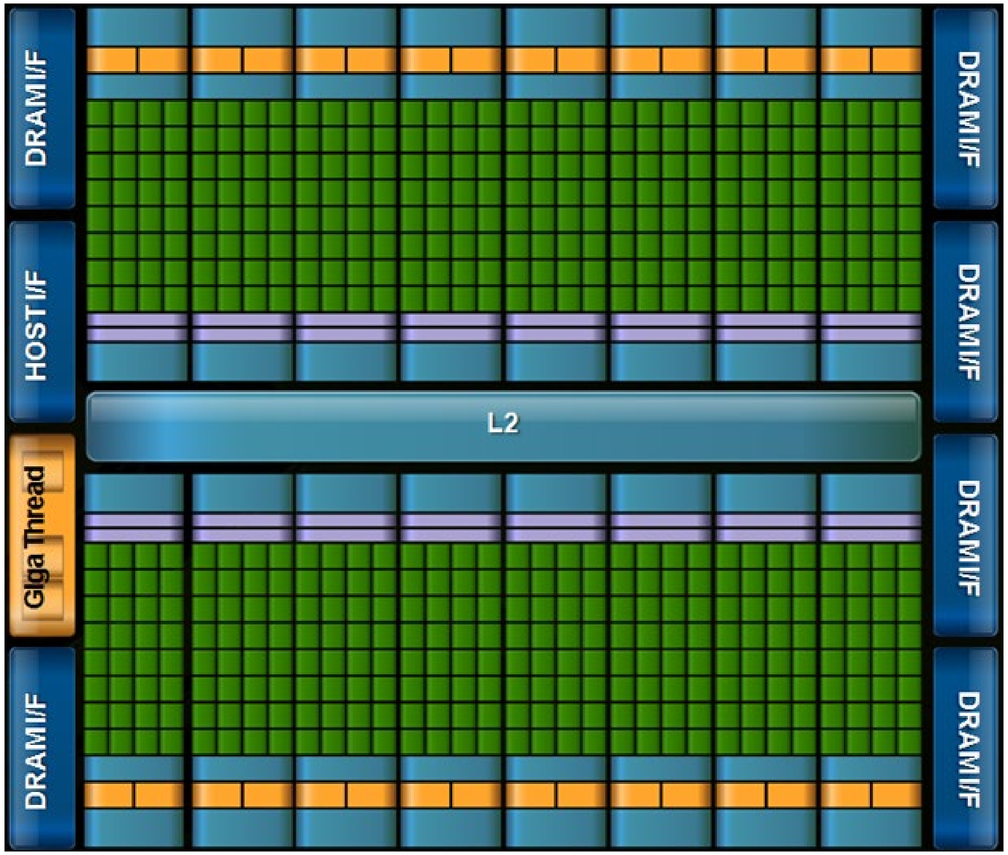


Why we are using rasterization?

- Efficiency
- Reasonably quality

Fermi GPU Architecture

16 SM (streaming processors)

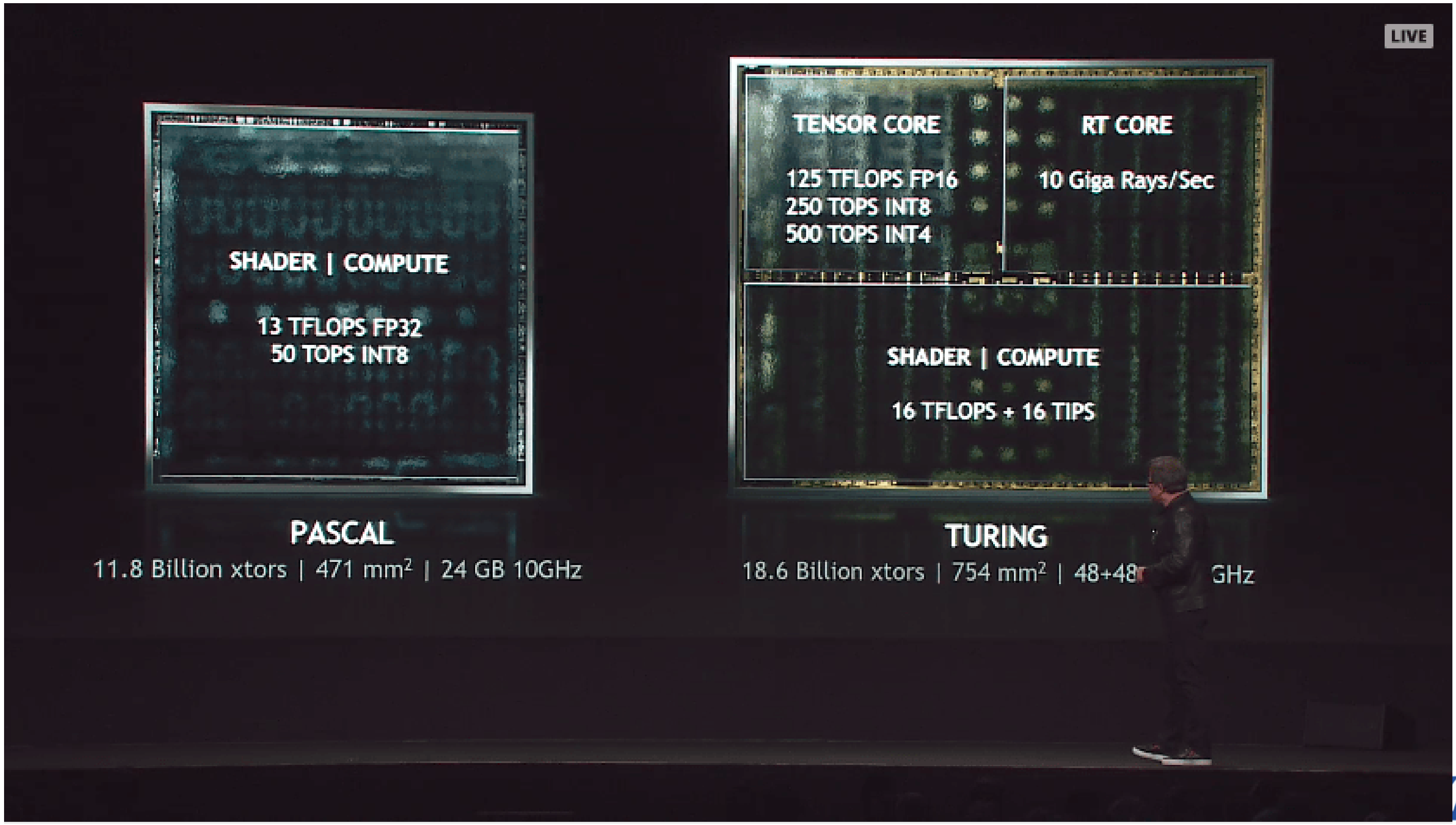


512 CUDA cores

Memory interfaces

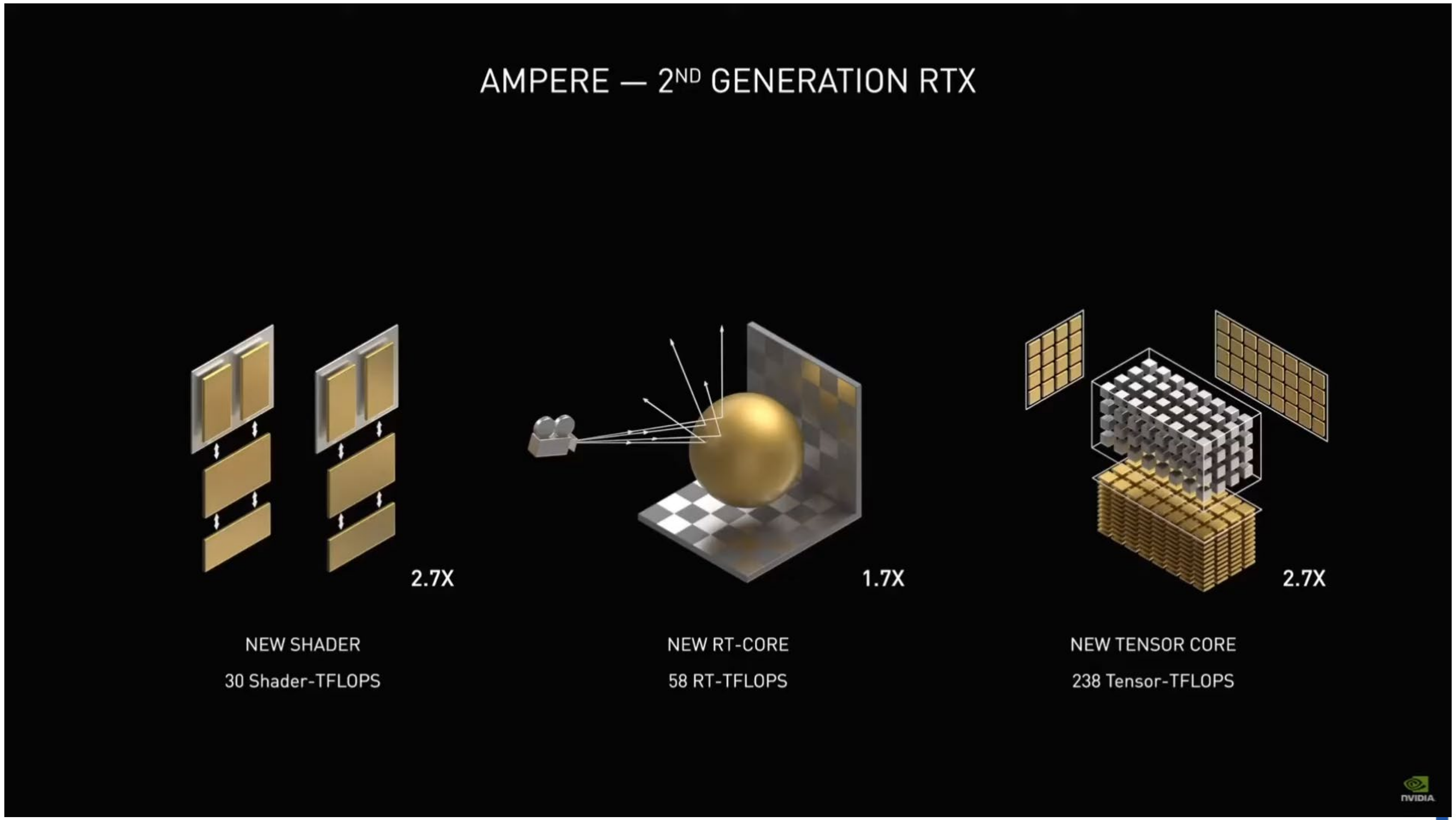
Turing Architecture, 2018

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

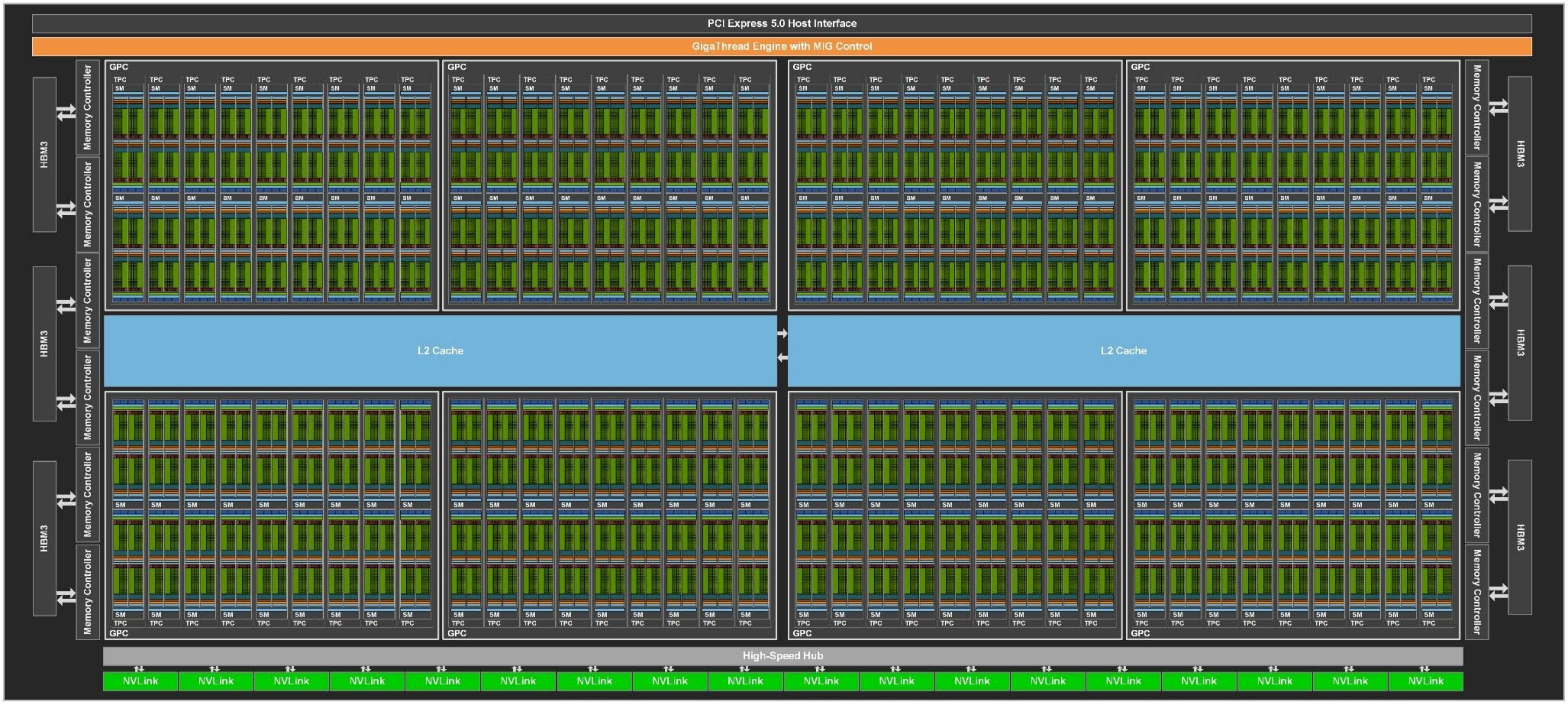


A100
Ampere Architecture, 2020

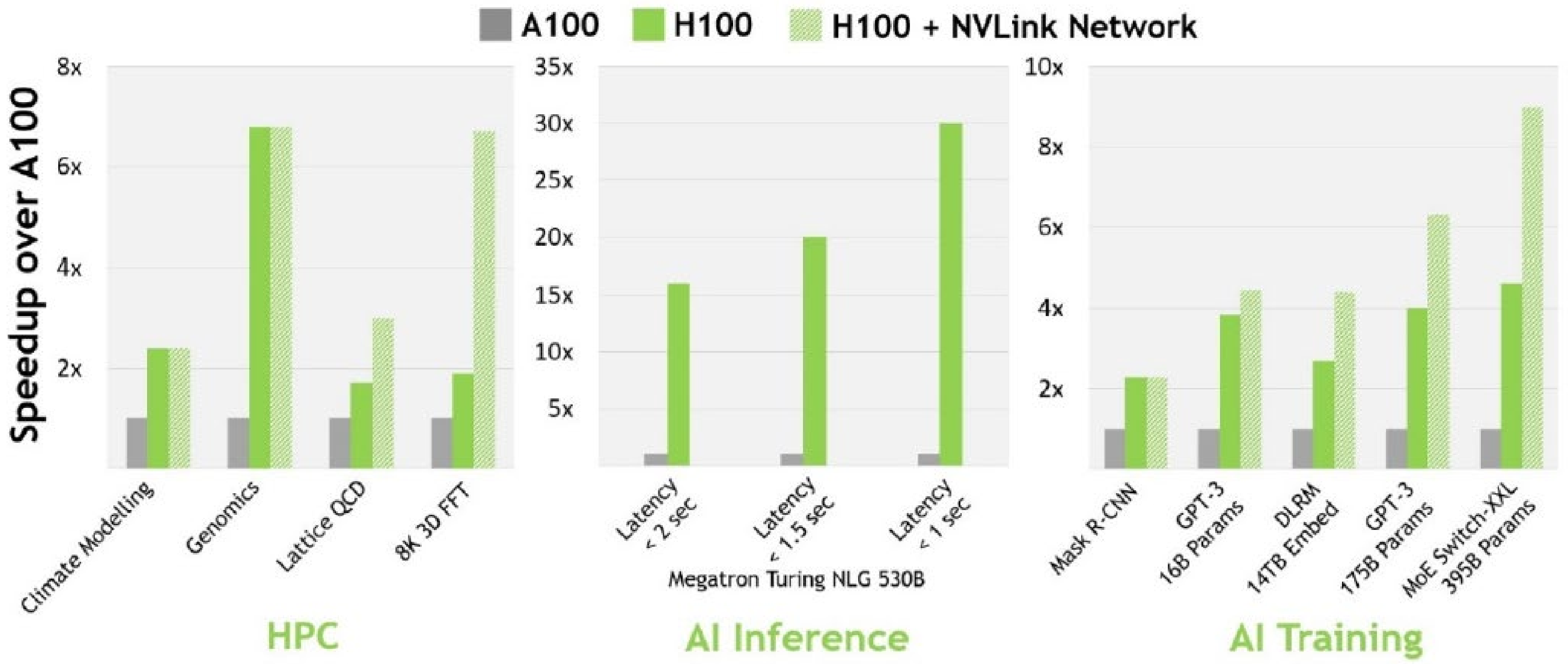
- **More cores, faster computation than Turing Architecture**



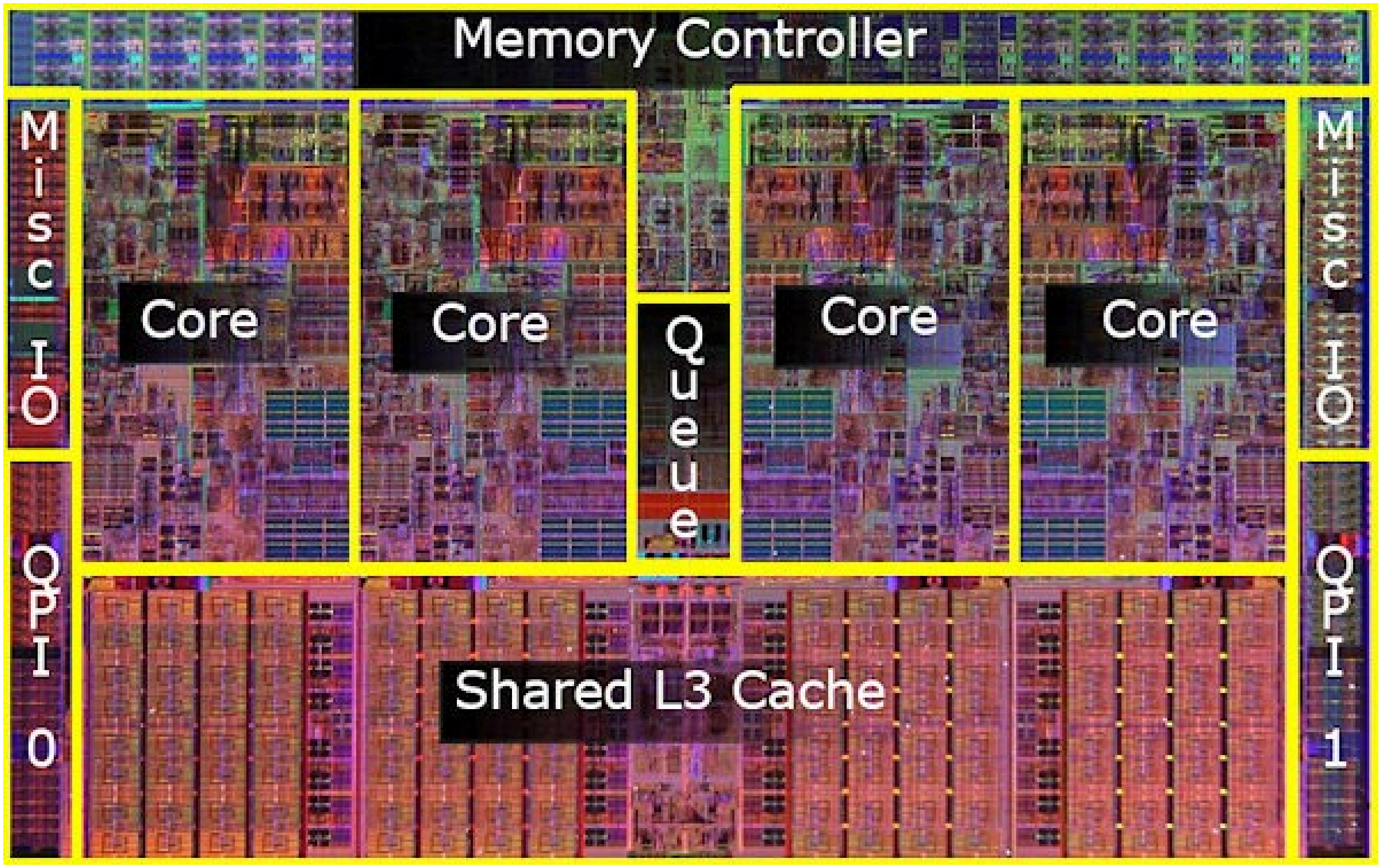
Nvidia Hopper Architecture (18K FP32 cores)


PCI Express 5.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
Memory Controller
HBM3
High-Speed Hub
NVLink

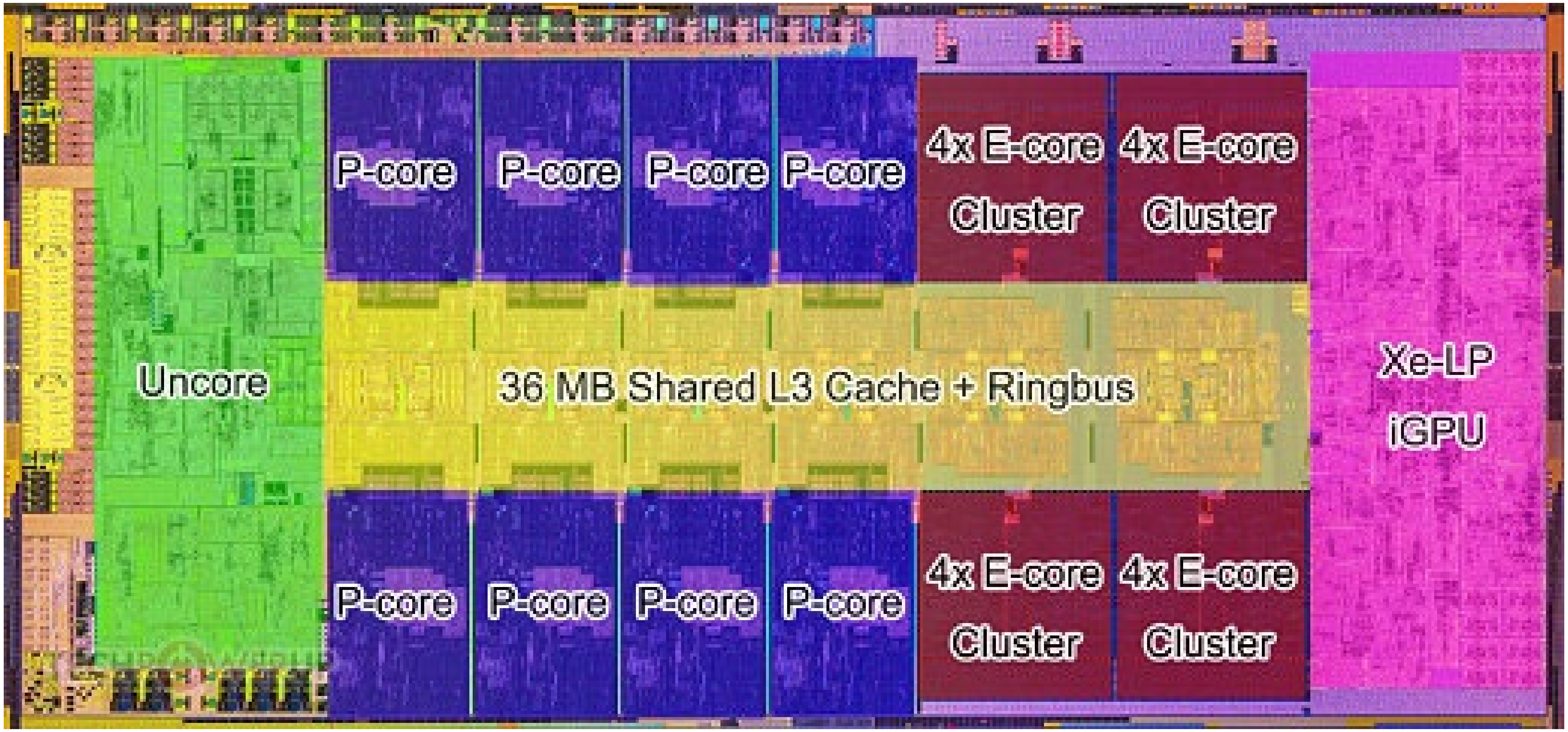
H100
Hopper (or Ada Lovelace), 2022



Recent CPU Chips (Intel's Core i7 processors) around 2020



Intel Core i9-13900K (USD 650 == 800 Korean won)



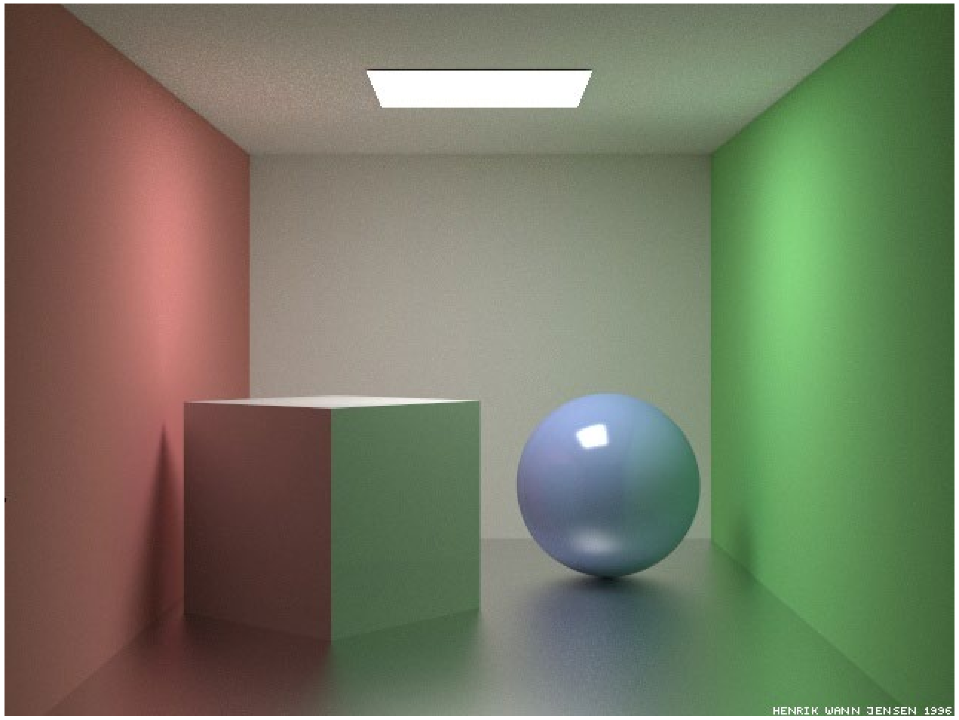
Where Rasterization Is

From Battlefield: Bad Company, EA Digital Illusions CE AB

From Eric Haines

But what about other visual cues?

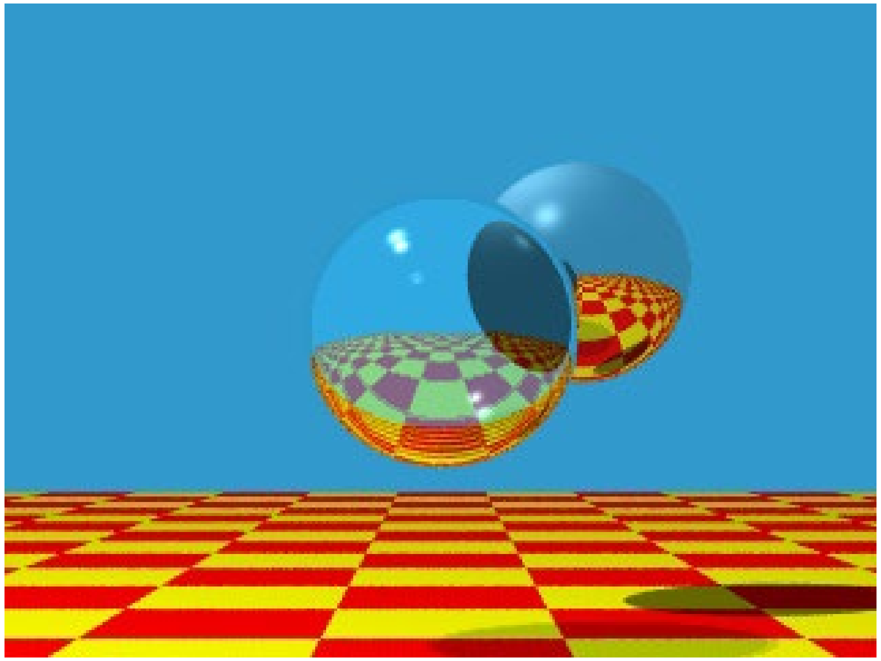
- Lighting
  - Shadows
  - Shading: glossy, transparency
- Color bleeding, etc



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**

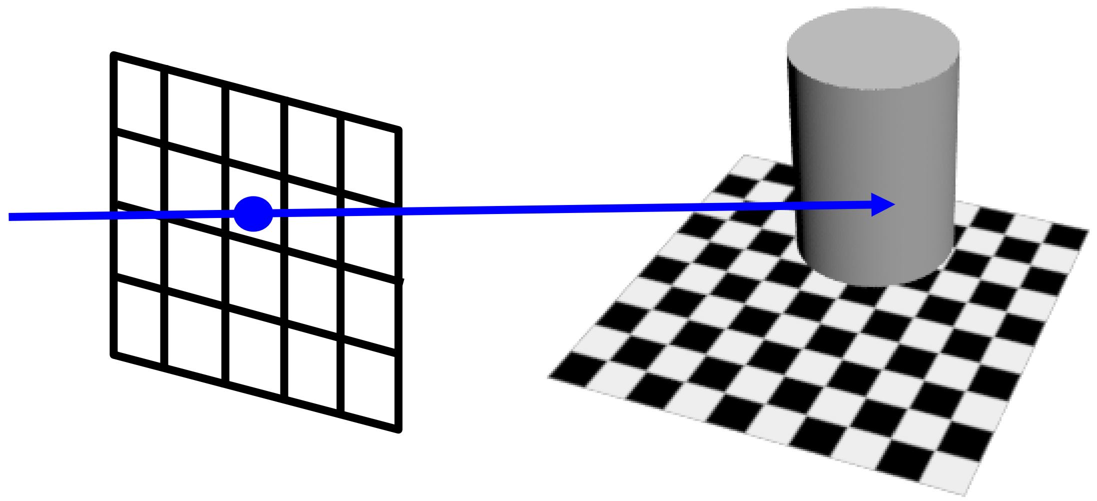


Ray Casting and Ray Tracing

- 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

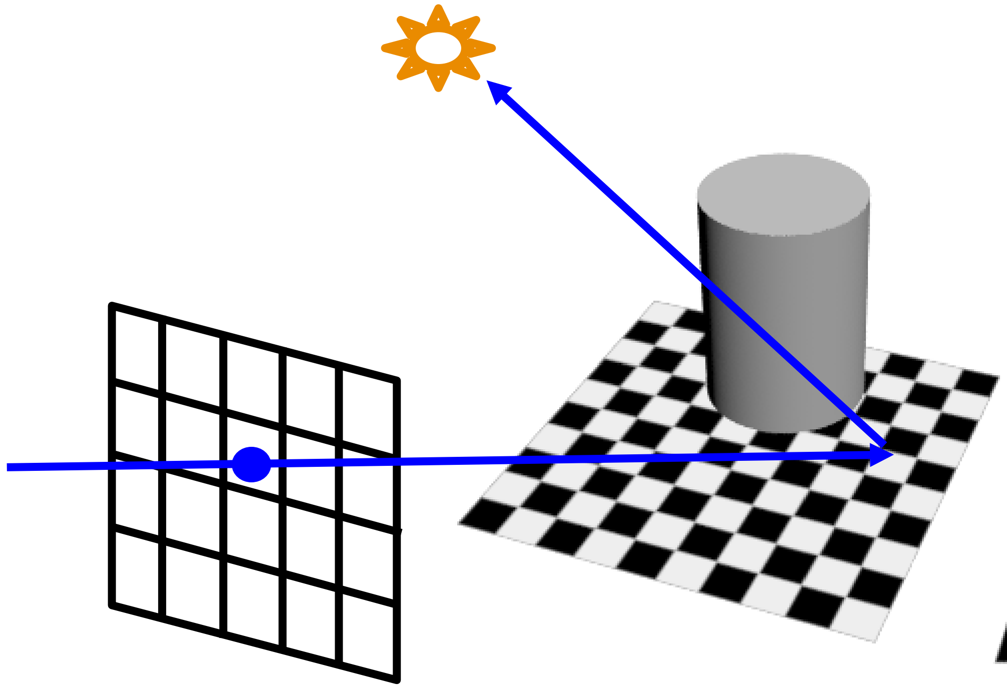
Basic Algorithms

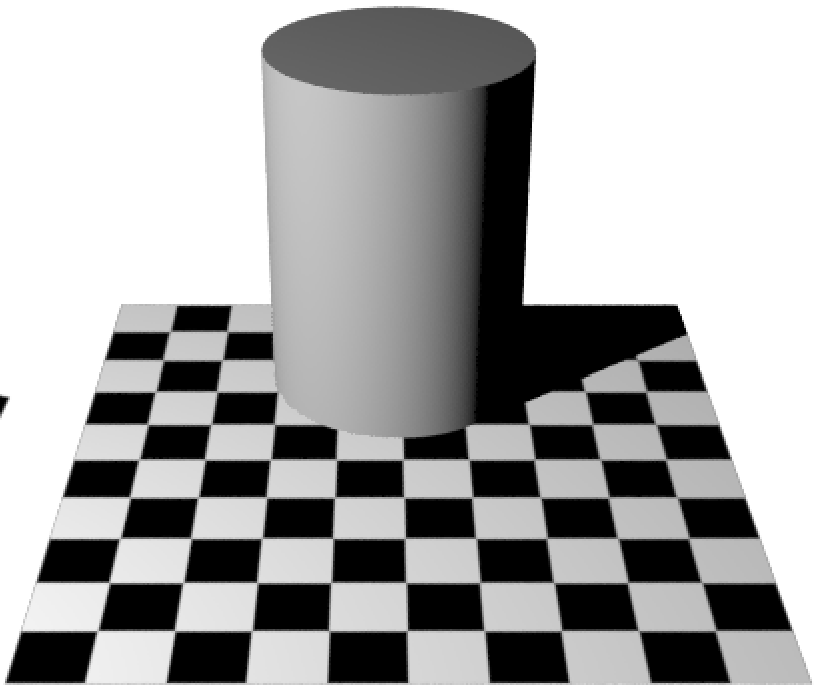
- **Rays are cast from the eye point through each pixel in the image**



Shadows

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

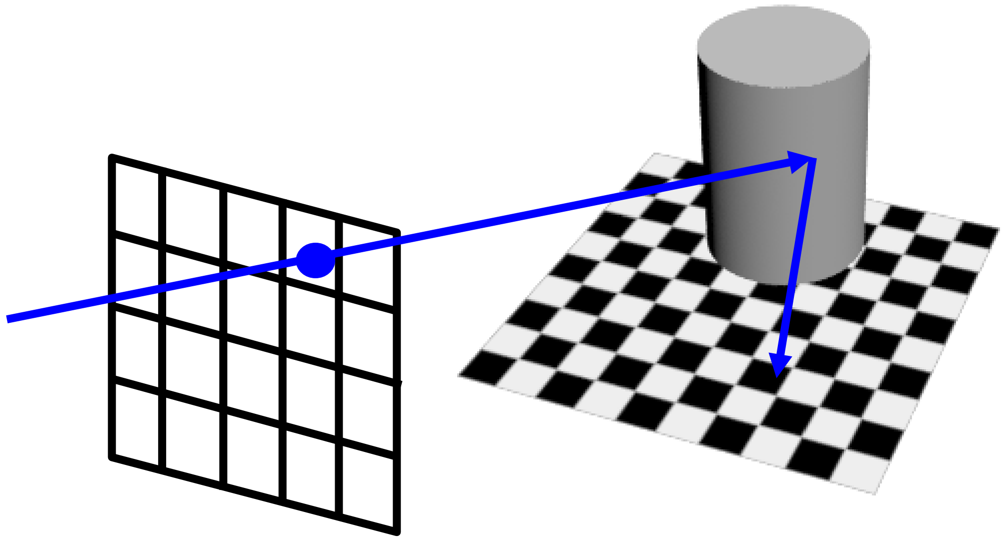


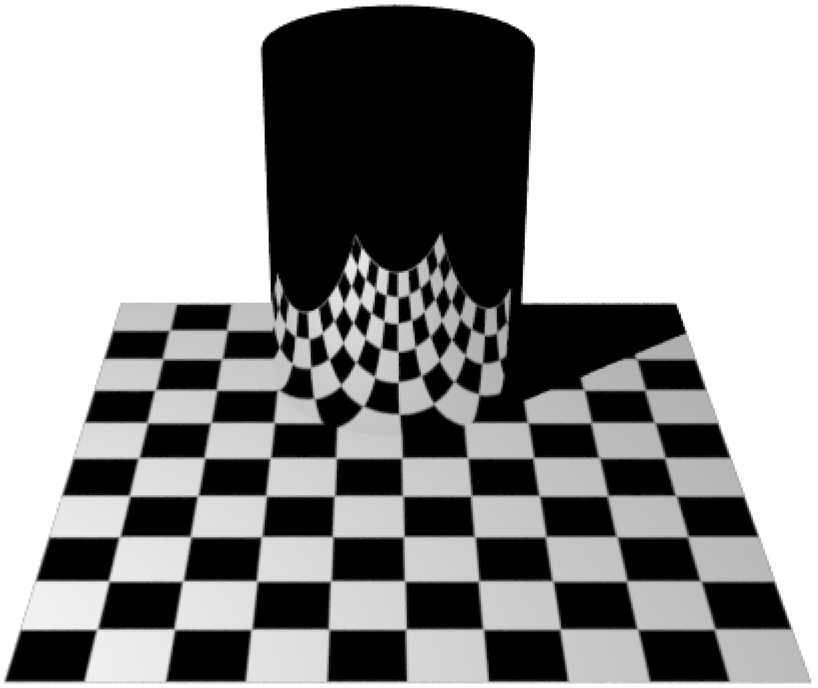


Used 3ds Max

Reflections

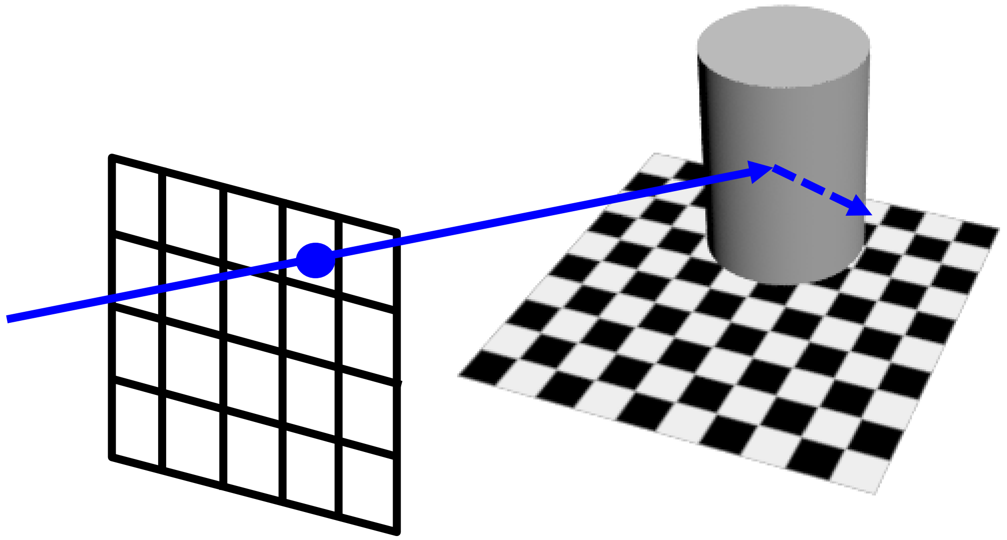
- **If object specular, cast secondary reflected rays**

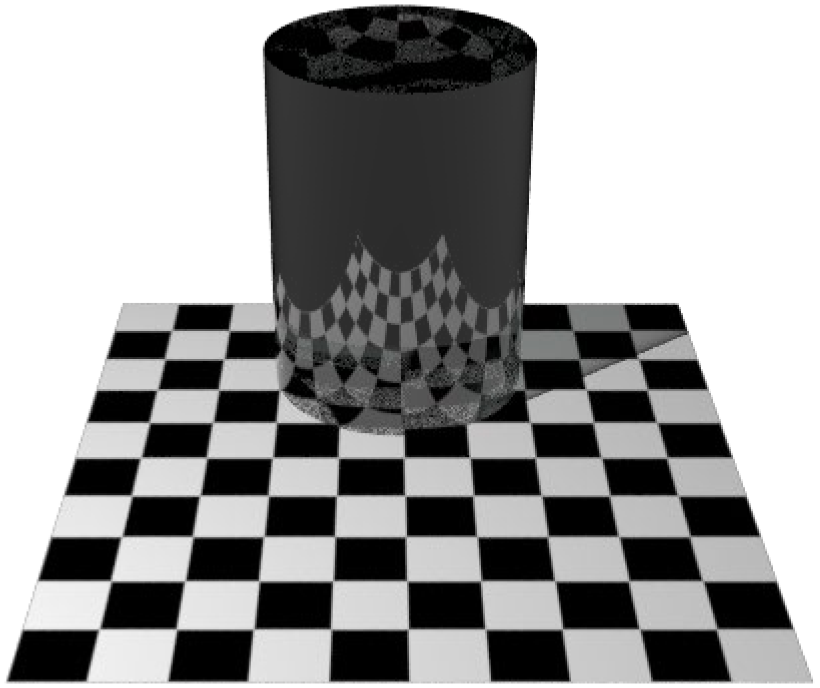




Refractions

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





An Improved Illumination Model [Whitted 80]

- **Phong illumination model**

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

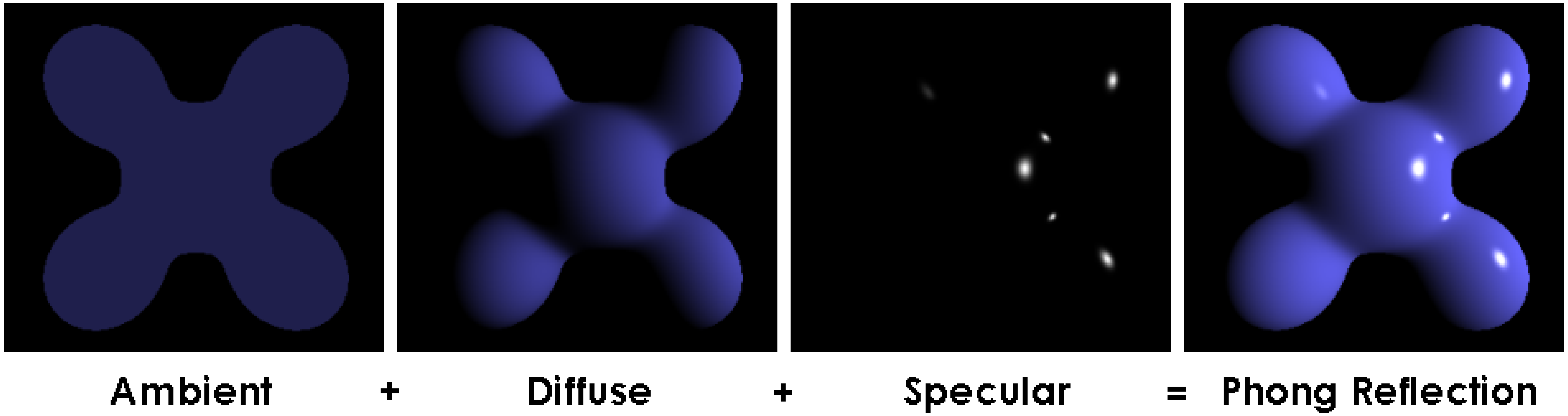
- **Whitted model**

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

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

OpenGL's Illumination Model

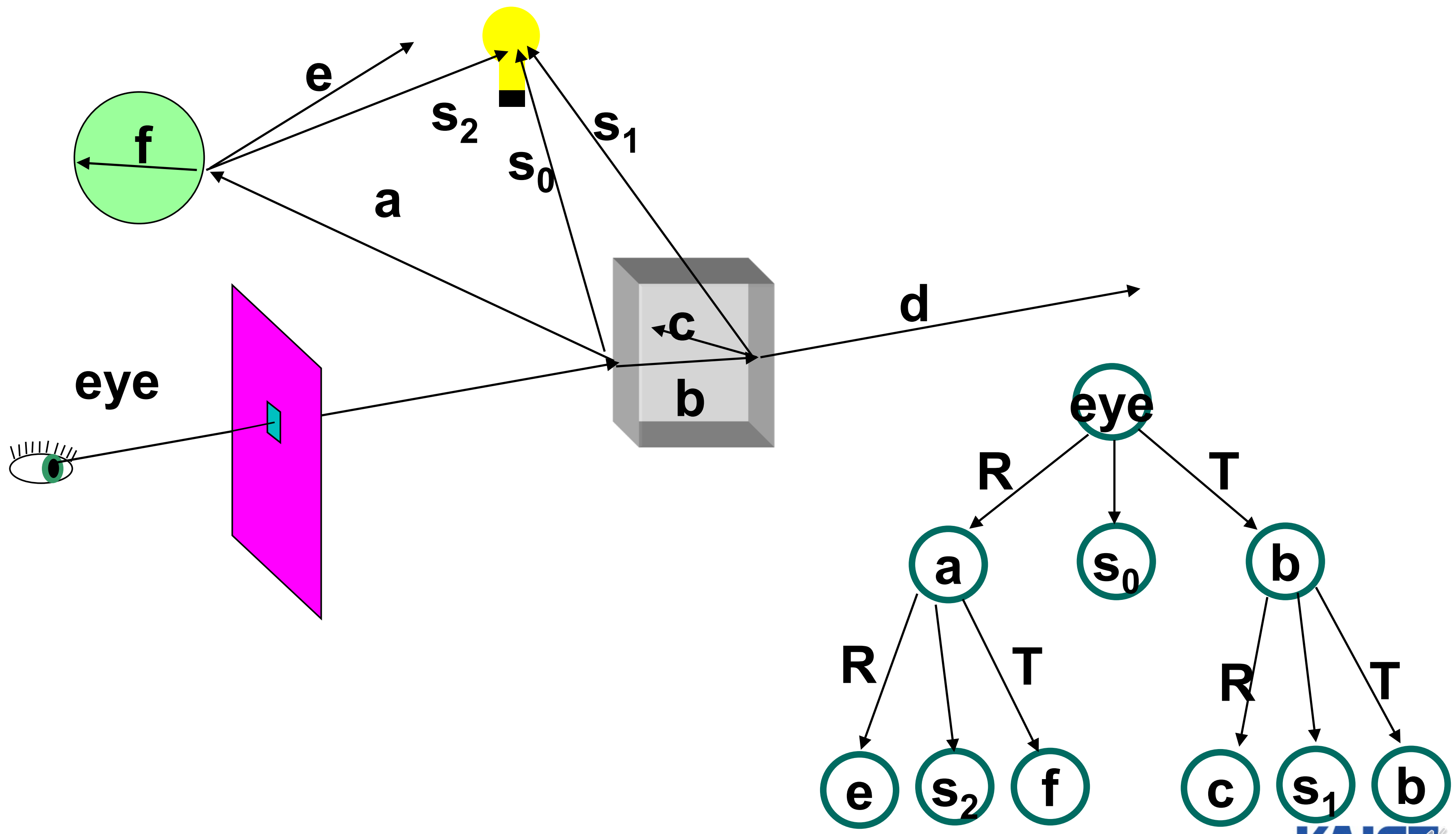
$$I_r = \sum_{j=1}^{numLights} (k_a^j I_a^j + k_d^j I_d^j \max((\hat{N} \bullet \hat{L}_j), 0) + k_s^j I_s^j \max((\hat{V} \bullet \hat{R})^{n_s}, 0))$$



From Wikipedia

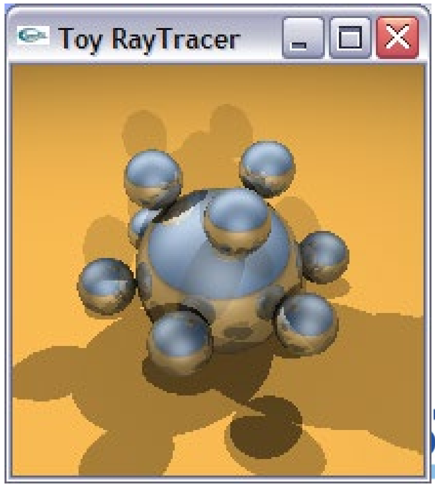
Details are available at Ch. 8 Illumination and Shading

Ray Tree



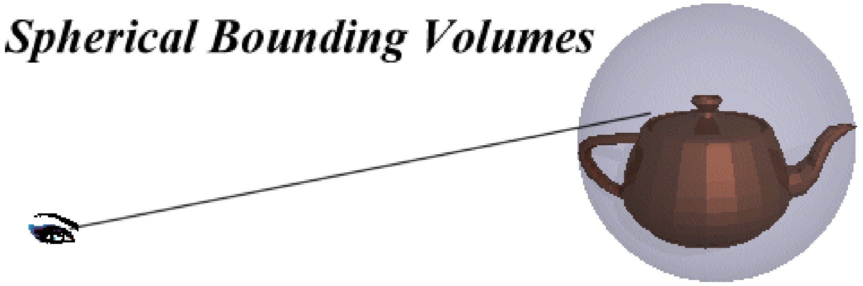
Acceleration Methods for Ray Tracing

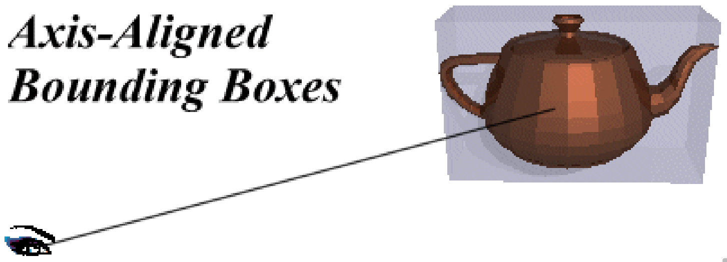
- **Rendering time for a ray tracer depends on the number of ray intersection tests per pixel**
  - **The number of pixels X the number of primitives in the scene**
- **Early efforts focused on accelerating the ray-object intersection tests**
  - **Ray-triangle intersection tests**
- **More advanced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



Bounding Volumes

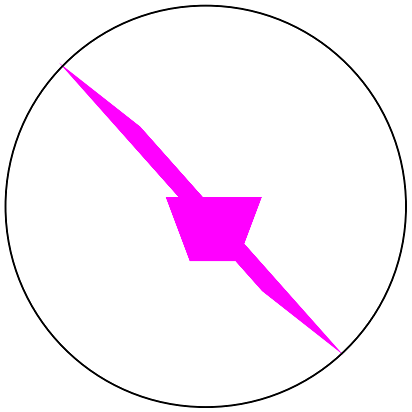
- Enclose complex objects within a simple-to-intersect objects
  - If the ray does not intersect the simple object then its contents can be ignored
  - The likelihood that it will strike the object depends on how tightly the volume surrounds the object.
- Spheres are simple, but not tight
- Axis-aligned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

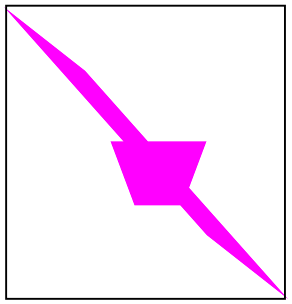




Bounding Volumes

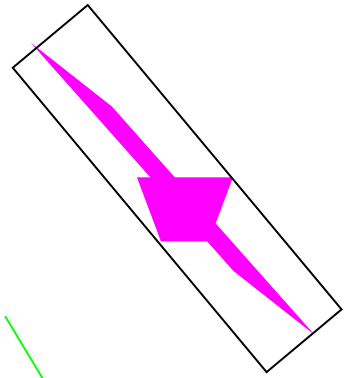
- Sphere [Whitted80]
  - Cheap to compute
  - Cheap test
  - Potentially very bad fit
- Axis-Aligned Bounding Box
  - Very cheap to compute
  - Cheap test
  - Tighter than sphere

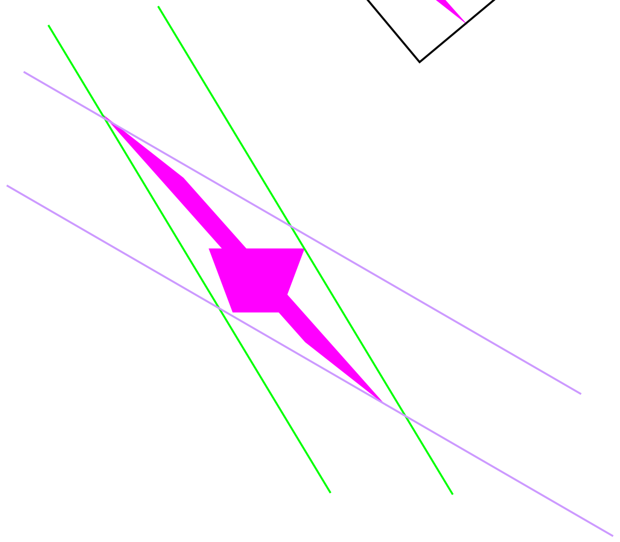




Bounding Volumes

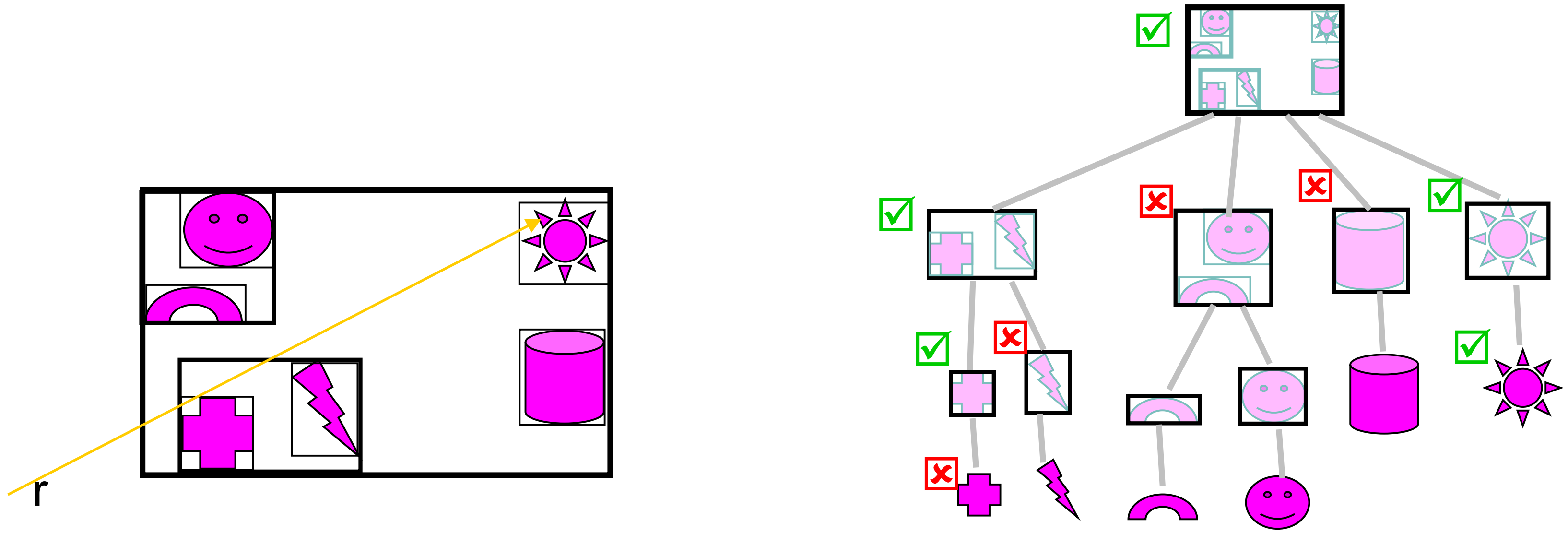
- Oriented Bounding Box
  - Fairly cheap to compute
  - Fairly Cheap test
  - Generally fairly tight
- Slabs / K-dops
  - More expensive to compute
  - Fairly cheap test
  - Can be tighter than OBB



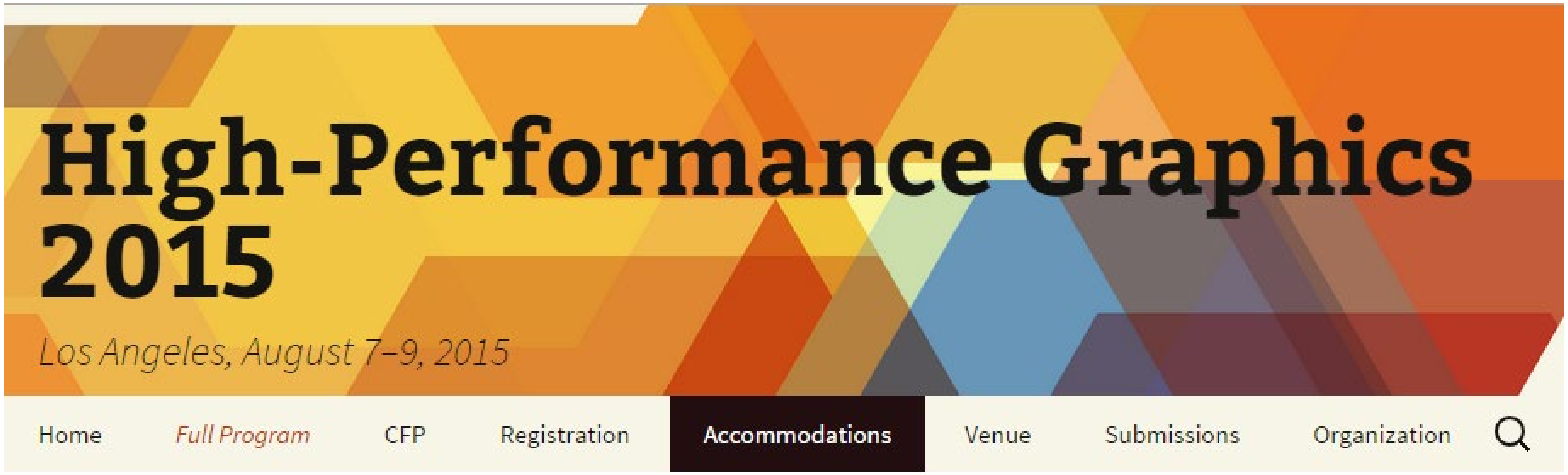


Bounding Volume Hierarchy (BVH)

- 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

Figure 1: Dress simulation: *Four different images of a 210 step sequence taken from a dynamic cloth simulation and consisting of 40K triangles. By updating in real-time instead of rebuilding the BVH of the deforming model according to our heuristic, we are able to render the animation at 13 frames per second with 512^2 screen resolution using a dual-core P4 processor at 2.8 GHz.*

ABSTRACT

We present an efficient approach for interactive ray tracing of deformable or animated models. Unlike many of the recent approaches for ray tracing static scenes, we use bounding volume hierarchies (BVHs) instead of kd-trees as the underlying acceleration structure. Our algorithm makes no assumptions about the simulation or the motion of objects in the scene and dynamically updates or recomputes the BVHs. We also describe a method to detect BVH quality degradation during the simulation in order to determine when the hierarchy needs to be rebuilt. Furthermore, we show that the ray coherence techniques introduced for kd-trees can be naturally extended to BVHs and yield similar improvements. Finally, we compare BVHs to spatial kd-trees, which have been used recently as a replacement for AABB hierarchies. Our algorithm has been applied to different scenarios arising in animation and simulation and consisting of tens of thousands to a million triangles. In practice, our system can ray trace these models at 3-13 frames a second on a desktop PC including secondary rays.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing;

Keywords: ray tracing, bounding volume hierarchies, deformable models, animation

1 INTRODUCTION

Ray tracing is a classic problem in computer graphics and has been studied in the literature for more than three decades. Most of the earlier ray tracing algorithms were used to generate high quality images for offline rendering, but in the last few years, there has been renewed interest in real-time ray tracing. At a broad level, most of the work in real-time ray tracing algorithms can be classified into three main categories: improved techniques to compute acceleration structures, exploiting ray coherence, and parallel algorithms on shared memory or distributed memory systems.

Most current interactive ray tracing algorithms use kd-trees as an acceleration data structure [22, 31]. In practice, kd-trees are simple to implement, can be stored in a compact manner, and are used for efficient tree traversal during ray intersections. However, one of the the main disadvantages of kd-trees is the high construction time; current algorithms can take seconds even on models composed of tens of thousands of triangles [9, 30]. Although fast build methods exist (e.g. [37]), the relative performance penalty for using a faster kd-tree build algorithm is severe compared to doing the same with object hierarchies. Furthermore, no simple and fast algorithms are known for incrementally updating the kd-tree hierarchy, even when the primitives undergo a simple deformation. As a result, current algorithms for interactive ray tracing are mainly limited to static scenes.

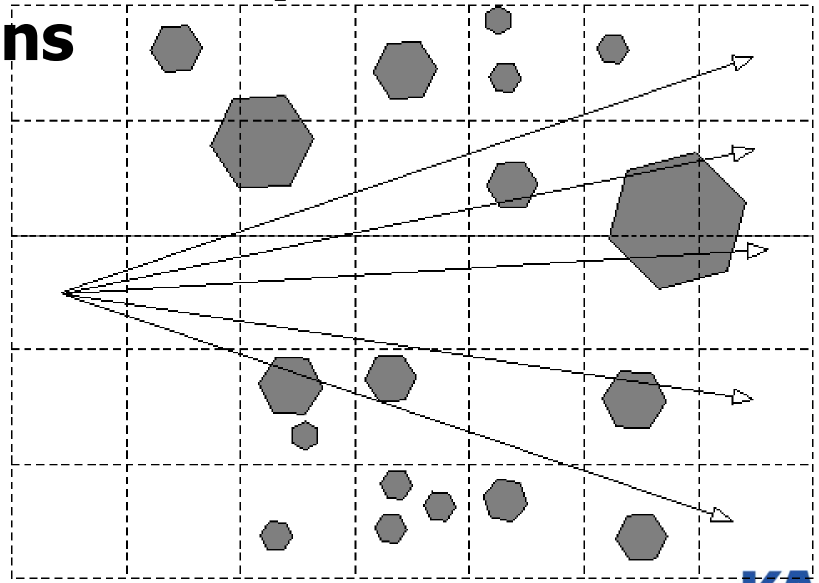
Main results: In this paper, we present a simple and efficient algorithm for interactive ray tracing of dynamic scenes of deformable objects, i.e. where the number of primitives does not change. We analyze many issues with respect to computation and incremental updates of hierarchies. Our algorithm uses bounding volume hierarchies (BVHs) of axis-aligned bounding boxes (AABBs), for which we describe efficient techniques to recompute or update these hierarchies during each frame. In practice, rebuilding of BVHs can be expensive, so we minimize these computations by measuring BVH quality degradation between successive frames. We also apply the ray coherence techniques developed for kd-trees to BVHs and obtain similar speedups. Finally, we describe techniques to parallelize these computations on multi-core architectures and improve the cache efficiency of the resulting algorithms. We have implemented our algorithm and highlight its performance on several dynamic scenes. Our system can render these datasets with

[*]e-mail: cl@cs.unc.edu
[†]e-mail:sungeui@llnl.gov
[‡]e-mail:tuft@cs.unc.edu
[§]e-mail:dm@cs.unc.edu

Spatial Subdivision

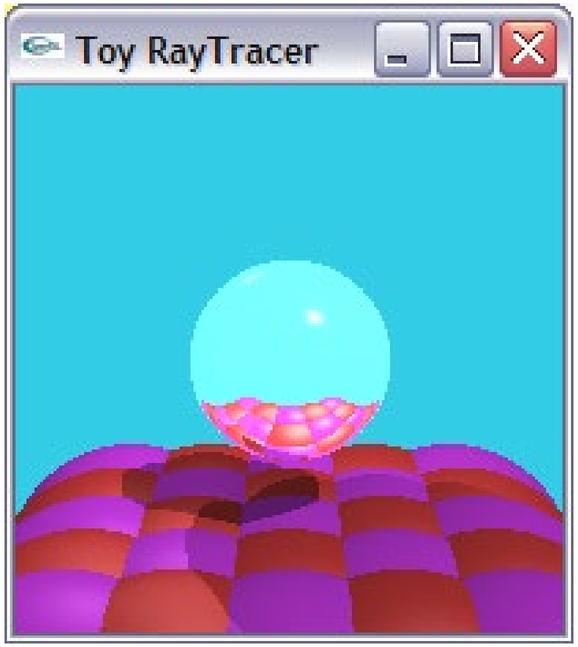
Idea: Divide space in to subregions

- 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

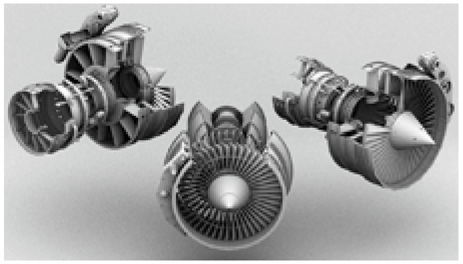
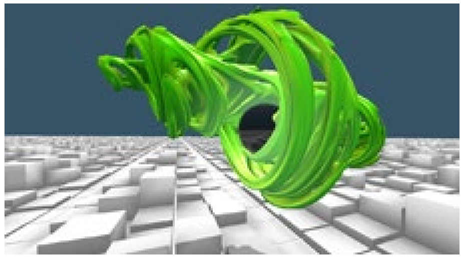
- **Problems with classic ray tracing**
  - **Not realistic**
  - **View-dependent**

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

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

Interactive Ray Tracing Kernels

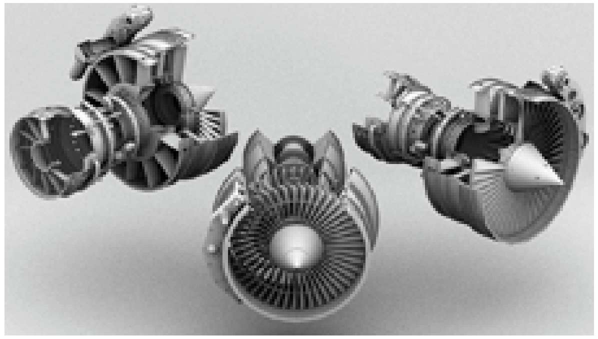
- OptiX, Nvidia
  - Utilize GPU computing architectures and CUDA

- 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