Computer Graphics

Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

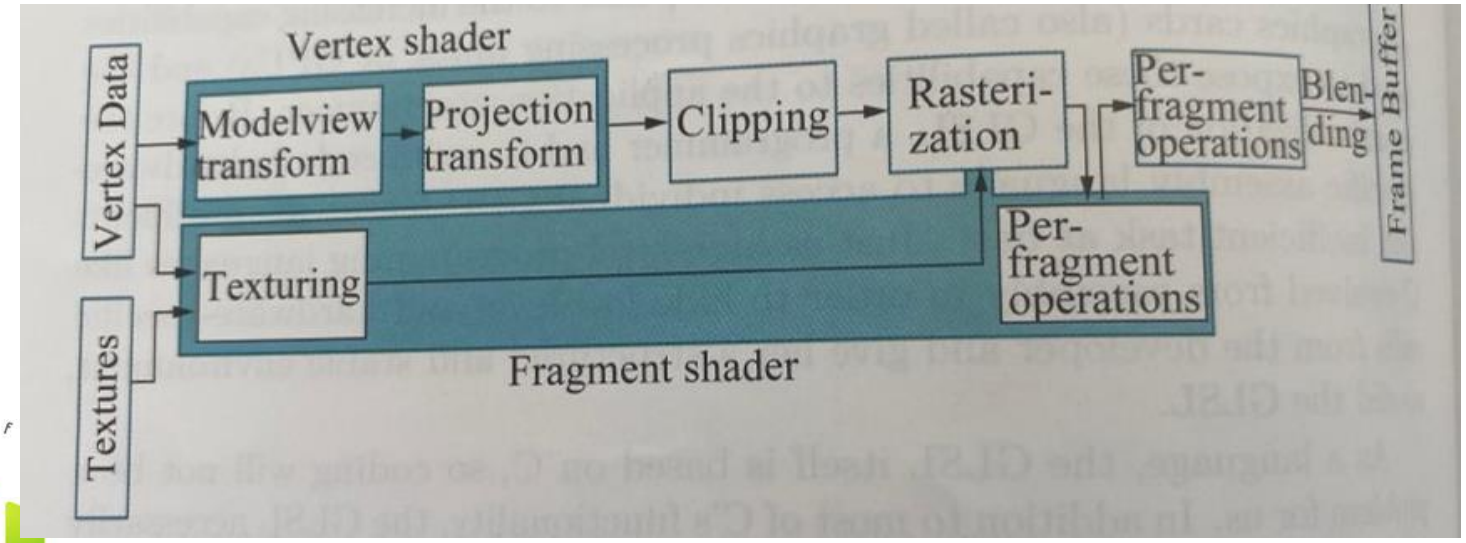
- **Shader and GLSL**

- Shader: GPU에서 동작하는 작은 프로그램. graphics pipeline에서 특정 부분을 맡아서 실행
- 동기: CPU 혼자 모든 그래픽 처리를 하는데 시간이 너무 오래 걸림. 픽셀 별로 병렬화 해서 동시에 수행할 수 없을까?
- GPU와 같은 그래픽 카드 기능을 지원 하고 쉽게 병렬화
- OpenGL Shading Language (GLSL)
- OpenGL에서 shader를 지원하기 위해 나옴. C언어 기반
- 대표적으로 vertex shader, fragment (pixel) shader가 있음

- The vertex shader operates on incoming vertex data (including coordinates, normal values, colors, etc) before handling over its output for clipping and rasterization
- Vertex shader는 각 vertex의 transformation을 수행한다. 3D 물체를 구성하는 vertex의 수만큼 실행

- The fragment shader operates on fragments in the raster before passing them on to the bottom stage of the classical pipeline
- fragment shader는 vertex rasterization, rasterization이 끝난 데이터를 통해 각 픽셀의 색을 계산하고 결정. 오직 한 픽셀만을 연산

- **GLSL simplified pipeline**
- **Vertex data > Vertex Shader > Clipping > Rasterization > Fragment Shader**
- **파란색 부분이 GLSL에서 프로그램 작성이 가능한 부분**



- **Ray tracing basic idea**

- Ray tracing은 전형적인 전역 조명 모델이다
- 여러 개의 표면 다각형 사이에서 일어나는 상호 간의 반사까지 고려하는 전역 조명 모델이다
- 1980년 초반 Kay 와 Whitted가 제안한 것이다
- 빛이 지나가는 경로를 ray라고 부른다
- 광원에서 나오는 모든 광선을 모두 추적하는 방법을 순방향 ray tracing (forward ray tracing)이라고 부르는데 이는 너무 많은 계산이 필요하기 때문에 실제로 구현이 어렵다

- **순방향 ray tracing의 문제점**
- **To follow light rays from each source as they interact with the scene till they finally reach the eye is not a particularly well-advised endeavor due to following reasons**
- **1. There is an infinite continuum of light rays emanating from each source**
- **2. Only a fraction reach the viewer (아래 그림 참고)**

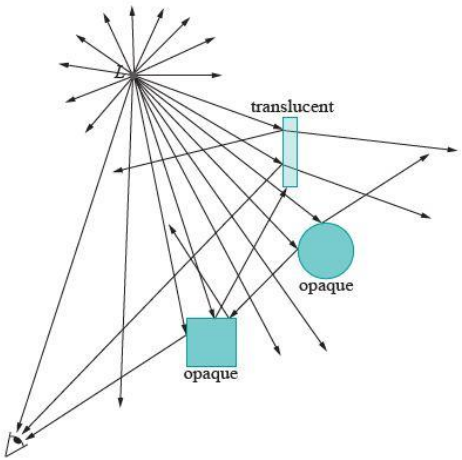


Figure 21.12: Tracing rays *from* a light source L – only few reach the eye.

- 따라서, 그래픽스에서는 역으로 우리 눈에서 광선이 발사된다고 가정하고 추적해 들어가는, 역방향 ray tracing을 주로 사용한다. 이를 일반적인 ray tracing이라 한다
- Rays are traced from the eye, one through each pixel, so that no computation is expended on rays which are ultimately invisible
- Each ray is followed through the pixel and into and around the scene, possibly bouncing off opaque objects and passing through translucent ones, for a finite amount of time, till a determination is made of its color

- An implementation has to “cut off” each ray after a finite number of steps and determine the color
- 아래 예: virtual screen
- 1. rays go off to infinity (출동 없이). 이러한 경우 배경색 할당
- 2. ray가 물체에 부딪힘. 이 물체에 색 할당. 이 물체의 색은 어떻게 결정 되나? 예: Phong 지역 조명 모델

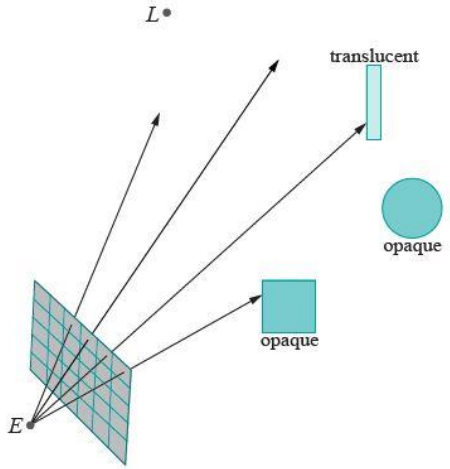
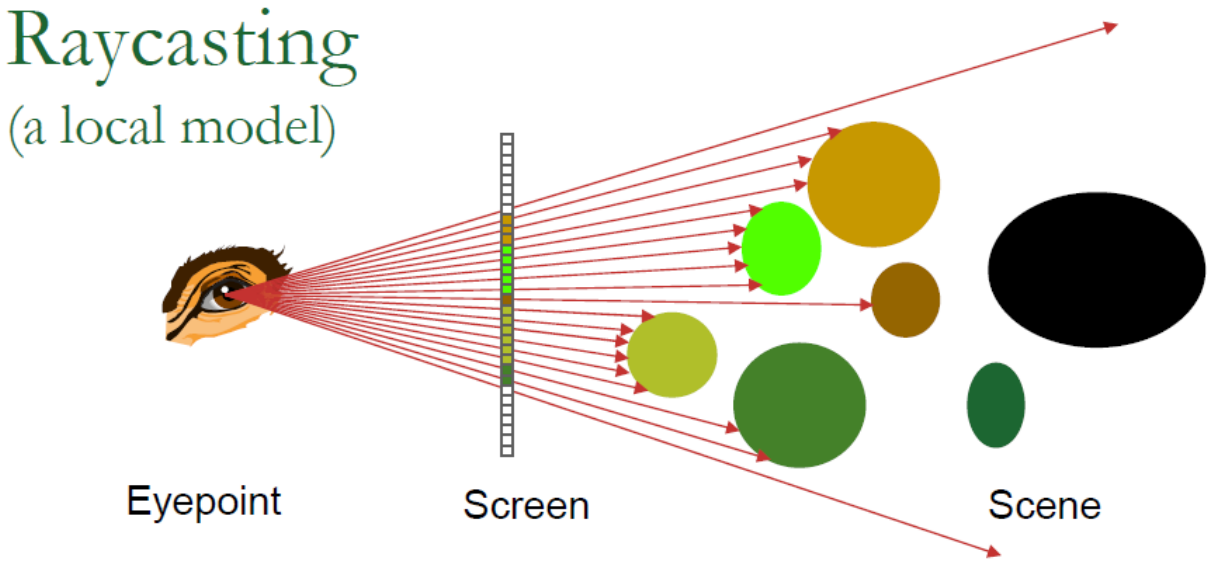


Figure 21.13: Tracing rays *from* the eye E, one through each pixel. Rays are “stopped” when they strike an object. LIght source is L.

기본 Ray Tracing 알고리즘 (local model)




```
for each pixel in the image
  cast a ray from the eye through the pixel
  determine the first intersection with a scene object
  apply the lighting model (e.g. Phong) at the intersection
  paint the pixel in the computed color
```