CS480: Computer Graphics
PA3: Distributed Ray Tracing

TA

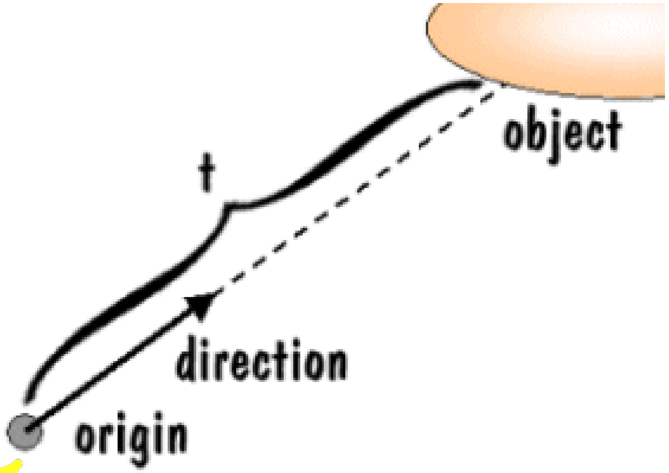
Course URL:
http://jupiter.kaist.ac.kr/~sungeui/CG/

Design of a Ray Tracer

- Building a ray tracer is simple

- We start with a convenient vector algebra library.
  - E.g., with vector and matrix of vecmat.h
- Ray object (defs.h)
  - Origin and direction
  - Trace (.)
    - Find a closest intersection point
  - Shade (.)
    - Perform shading

- Light sources (defs.h)
  - Supports directional light.



Renderable

- **Every object in our ray tracer must be able to**
  - **Intersect itself with a ray.**
  - **Shade itself (determine the color it reflects along the given ray).**

```
Class MyObject
{
          .
      Surface* surface;
          .
      intersect (ray):    # returns boolean
      shade(ray,  lightList,  objectList, bgndColor):     #returns (r,g,b)
          .
}
```

  - **Current code has a renderable sphere object (sphere.h).**

Surface Object (surface.cpp)

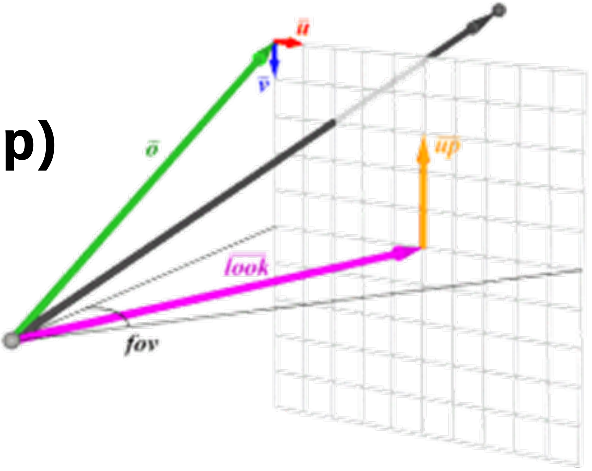
- Contains various material properties.

```
class Surface
{
        RGBColor baseColor; // base color of the surface
        float ka, kd, ks;   // ambient, diffuse, specular coefficients
        float ns;           // Shineness power
        float kr;           // reflection coef.
        float kt;           // transparency coef.
        float ior;          // index of refraction
}
```

- Surface shader
  - Accumulate contributions from lights.
  - Handle reflection, refractions and other things.

Ray Tracing Application

- **Generate primary rays.**
  - **Refer to renderLine(.) (RayTrace.cpp)**



- That’s basically all we need to write a ray tracer.
  - Compared to a graphics pipeline, the code is very simple and easy to understand.

Display List Parser

- **We can use a simple input parser similar to the one used for Wavefront OBJ files. Here is an example input file.**

```
eye 0 2 10
lookat 0 0 0
up 0 1 0
fov 30
background 0.2 0.8 0.9
light 1 1 1 ambient
light 1 1 1 directional -1 -2 -1
light 0.5 0.5 0.5 point -1 2 -1

surface 0.7 0.2 0.8 0.5 0.4 0.2 10.0 0.0 0.0 1.0
sphere -2 -3 -2 1.5
sphere  0 -3 -2 1.5
sphere  2 -3 -2 1.5
sphere -1 -3 -1 1.5
sphere  1 -3 -1 1.5
sphere -2 -3  0 1.5
sphere  0 -3  0 1.5
sphere  2 -3  0 1.5
sphere -1 -3  1 1.5
sphere  1 -3  1 1.5
sphere -2 -3  2 1.5
sphere  0 -3  2 1.5
sphere  2 -3  2 1.5
```

```
surface 0.7 0.2 0.2 0.5 0.4 0.2 3.0 0.0 0.0 1.0
sphere -1 -3 -2 1.5
sphere  1 -3 -2 1.5
sphere -2 -3 -1 1.5
sphere  0 -3 -1 1.5
sphere  2 -3 -1 1.5
sphere -1 -3  0 1.5
sphere  1 -3  0 1.5
sphere -2 -3  1 1.5
sphere  0 -3  1 1.5
sphere  2 -3  1 1.5
sphere -1 -3  2 1.5
sphere  1 -3  2 1.5

surface 0.4 0.4 0.4 0.1 0.1 0.6 100.0 0.8 0.0 1.0
sphere  0 0 0 1
```

Usage of Codes

- RT.exe balls.ray

- Extend codes to support PA3 requirements.
  - Please go over lecture materials.

Requirements

- **Extend the surface shader to handle refraction.**
  - **For Sphere case, note that the ray can hit inside of the Sphere.**

Requirements

- Implement a “Triangle” Object.
  - Add texture mapping.
  - Reflection/refraction is not required the textured triangles.

Requirements

- Add a randomized sampling method for enhanced rendering.
  - Antialiasing: perform jittered sampling on the pixel area.
  - Soft-shadows: imitate rectangular area light.

4*4 jittered sampling for antialiasing and soft-shadows.