

1

# 3D Rendering

Adam Finkelstein  
Princeton University  
COS 426, Fall 2001

2

## Course Syllabus

- I. Image processing
- II. Rendering
- III. Modeling
- IV. Animation


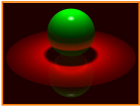




Image Processing  
*(Rusty Coleman, CS426, Fall99)*



Rendering  
*(Michael Bostock, CS426, Fall99)*



Modeling  
*(Dennis Zorin, CalTech)*



Animation  
*(Angel, Plate 1)*

3

## Where Are We Now?

- I. Image processing
- II. Rendering**
- III. Modeling
- IV. Animation


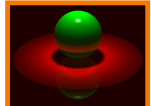




Image Processing  
*(Rusty Coleman, CS426, Fall99)*



Rendering  
*(Michael Bostock, CS426, Fall99)*



Modeling  
*(Dennis Zorin, CalTech)*




Animation  
*(Angel, Plate 1)*

4

## Rendering

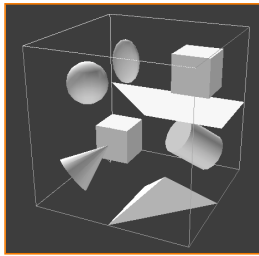
- Generate an image from geometric primitives



Geometric Primitives
Rendering
Raster Image

5

## 3D Rendering Example

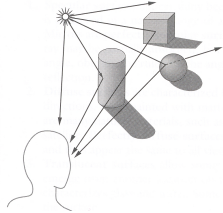


What issues must be addressed by a 3D rendering system?

6

## Overview

- 3D scene representation
- 3D viewer representation
- Visible surface determination
- Lighting simulation

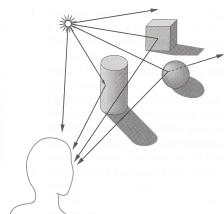


7

### Overview

- » 3D scene representation
- 3D viewer representation
- Visible surface determination
- Lighting simulation

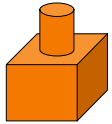
How is the 3D scene described in a computer?



8

### 3D Scene Representation

- Scene is usually approximated by 3D primitives
  - Point
  - Line segment
  - Polygon
  - Polyhedron
  - Curved surface
  - Solid object
  - etc.




9


### 3D Point

- Specifies a location
  - Represented by three coordinates
  - Infinitely small

```
typedef struct {
  Coordinate x;
  Coordinate y;
  Coordinate z;
} Point;
```



(x,y,z)

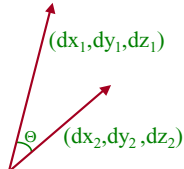


10

### 3D Vector

- Specifies a direction and a magnitude
  - Represented by three coordinates
  - Magnitude  $\|V\| = \sqrt{dx^2 + dy^2 + dz^2}$
  - Has no location
- Dot product of two 3D vectors
  - $V_1 \cdot V_2 = dx_1 dx_2 + dy_1 dy_2 + dz_1 dz_2$
  - $V_1 \cdot V_2 = \|V_1\| \|V_2\| \cos(\theta)$

```
typedef struct {
  Coordinate dx;
  Coordinate dy;
  Coordinate dz;
} Vector;
```

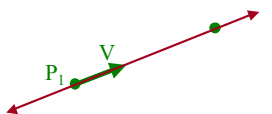



11

### 3D Line

- Line segment with both endpoints at infinity
  - Parametric representation:
    - »  $P = P_1 + tV, \quad (-\infty < t < \infty)$

```
typedef struct {
  Point P1;
  Vector V;
} Line;
```



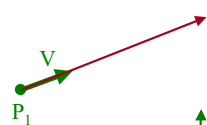



12

### 3D Ray

- Line segment with one endpoint at infinity
  - Parametric representation:
    - »  $P = P_1 + tV, \quad (0 \leq t < \infty)$

```
typedef struct {
  Point P1;
  Vector V;
} Ray;
```





### 3D Line Segment

13

- Specifies a linear combination of two points
  - Parametric representation:
    - $P = P_1 + t(P_2 - P_1)$ , ( $0 \leq t \leq 1$ )

```
typedef struct {
  Point P1;
  Point P2;
} Segment;
```

### 3D Plane

14

- Specifies a linear combination of three points
  - Implicit representation:
    - $P \cdot N + d = 0$ , or
    - $ax + by + cz + d = 0$

```
typedef struct {
  Vector N;
  Distance d;
} Plane;
```

- N is the plane "normal"
  - Unit-length vector
  - Perpendicular to plane

### 3D Polygon

15

- Area "inside" a sequence of coplanar points
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting

```
typedef struct {
  Point *points;
  int npoints;
} Polygon;
```

Points are in counter-clockwise order

- Holes (use > 1 polygon struct)

### 3D Sphere

16

- All points at distance "r" from point " $(c_x, c_y, c_z)$ "
  - Implicit representation:
    - $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r^2$
  - Parametric representation:
    - $x = r \cos(\phi) \cos(\theta) + c_x$
    - $y = r \cos(\phi) \sin(\theta) + c_y$
    - $z = r \sin(\phi) + c_z$

```
typedef struct {
  Point center;
  Distance radius;
} Sphere;
```

### 3D Geometric Primitives

17

- More detail on 3D modeling later in course
  - Point
  - Line segment
  - Polygon
  - Polyhedron
  - Curved surface
  - Solid object
  - etc.

H&B Figure 10.46

### Overview

18

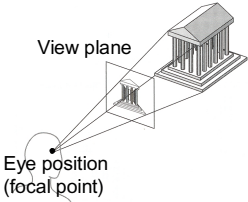
- 3D scene representation
  - » 3D viewer representation
- Visible surface determination
- Lighting simulation

How is the viewing device described in a computer?

## Camera Models

19

- The most common model is pin-hole camera
  - All captured light rays arrive along paths toward focal point without lens distortion (everything is in focus)
  - Sensor response proportional to radiance



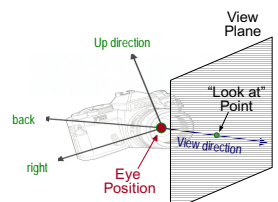
Other models consider ...

- Depth of field
- Motion blur
- Lens distortion

## Camera Parameters

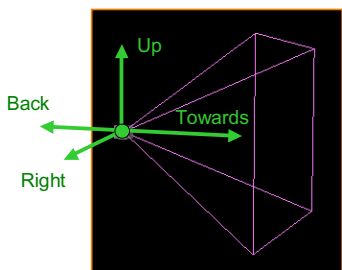
20

- Position
  - Eye position (px, py, pz)
- Orientation
  - View direction (dx, dy, dz)
  - Up direction (ux, uy, uz)
- Aperature
  - Field of view (xfov, yfov)
- Film plane
  - "Look at" point
  - View plane normal



## Moving the camera

21



View Frustum

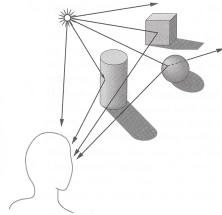
[Demo](#)

## Overview

22

- 3D scene representation
- 3D viewer representation
- » **Visible surface determination**
- Lighting simulation

How can the front-most surface be found with an algorithm?

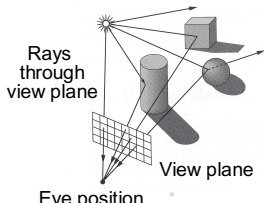


## Visible Surface Determination

23

- The color of each pixel on the view plane depends on the radiance emanating from visible surfaces

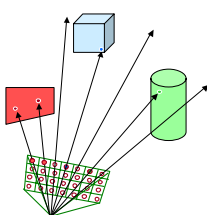
Simplest method is ray casting



## Ray Casting

24

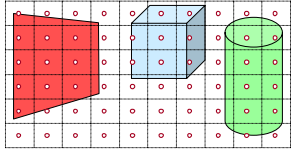
- For each sample ...
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance



## Ray Casting

25

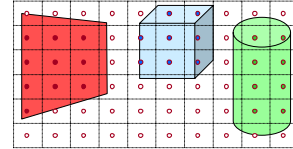
- For each sample ...
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance



## Visible Surface Determination

26

- For each sample ...
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance

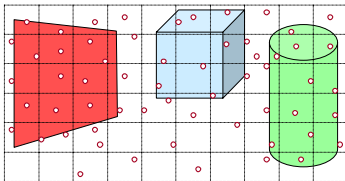


More efficient algorithms utilize spatial coherence!

## Rendering Algorithms

27

Rendering is a problem in sampling and reconstruction!

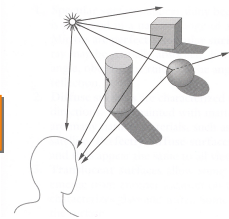


## Overview

28

- 3D scene representation
- 3D viewer representation
- Visible surface determination
- » **Lighting simulation**

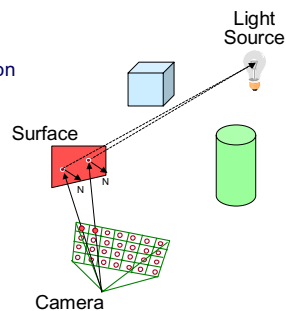
How do we compute the radiance for each sample ray?



## Lighting Simulation

29

- Lighting parameters
  - Light source emission
  - Surface reflectance
  - Atmospheric attenuation
  - Camera response

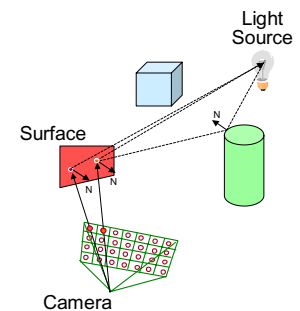


## Lighting Simulation

30

- Direct illumination
  - Ray casting
  - Polygon shading
- Global illumination
  - Ray tracing
  - Monte Carlo methods
  - Radiosity methods

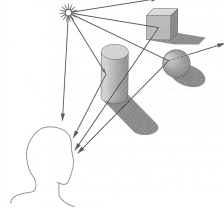
More on these methods later!



## Summary

31

- Major issues in 3D rendering
  - 3D scene representation
  - 3D viewer representation
  - Visible surface determination
  - Lighting simulation

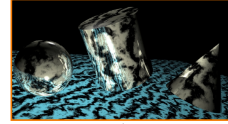


- Concluding note
  - Accurate physical simulation is complex and intractable
    - » Rendering algorithms apply many approximations to simplify representations and computations

## Next Lecture

32

- Ray intersections
- Light and reflectance models
- Indirect illumination



Render Boy  
*(R. Kalman & H. Orlin,  
Assignment 3, CS 426, Fall198,  
Princeton University)*

For assignment #2, you will write a ray tracer!