3D Rendering

Adam Finkelstein
Princeton University
COS 426, Spring 2005

Course Syllabus

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

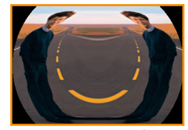
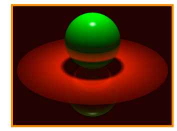
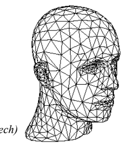


Image Processing
(Rusty Coleman, CS426, Fall99)


Rendering
(Michael Bostock, CS426, Fall99)

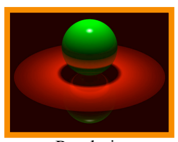

Modeling
(Dennis Zorin, CalTech)


Animation
(Jon Beyer, CS426, Spring04)

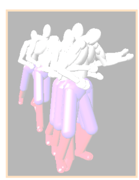
Where Are We Now?

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


Image Processing
(Rusty Coleman, CS426, Fall99)

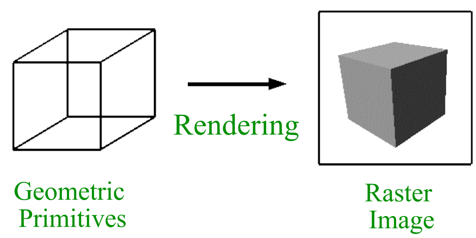

Rendering
(Michael Bostock, CS426, Fall99)

Modeling
(Dennis Zorin, CalTech)

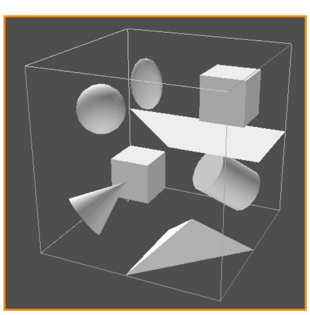

Animation
(Jon Beyer, CS426, Spring04)

Rendering

- Generate an image from geometric primitives



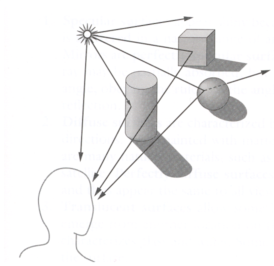
3D Rendering Example



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

Overview

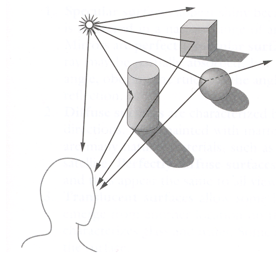
- 3D scene representation
- 3D viewer representation
- Visible surface determination
- Lighting simulation



Overview

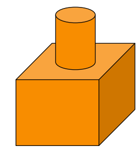
- » **3D scene representation**
- 3D viewer representation
- Visible surface determination
- Lighting simulation

How is the 3D scene described in a computer?



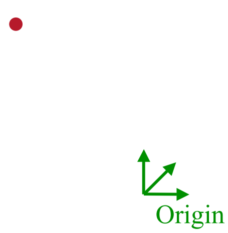
3D Scene Representation

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



3D Point

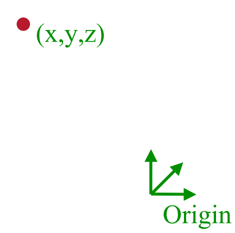
- Specifies a location



3D Point

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

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



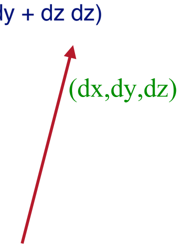
3D Vector

- Specifies a direction and a magnitude

3D Vector

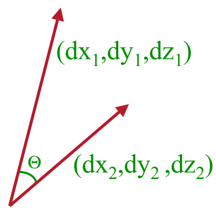
- Specifies a direction and a magnitude
  - Represented by three coordinates
  - Magnitude ||V|| = sqrt(dx dx + dy dy + dz dz)
  - Has no location

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



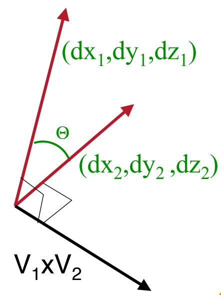
3D Vector

- Dot product of two 3D vectors
  - $V_1 \cdot V_2 = ||V_1|| \, ||V_2|| \cos(\Theta)$



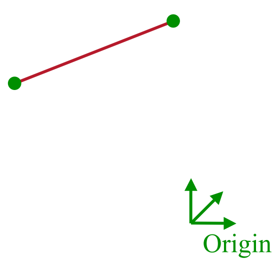
3D Vector

- Cross product of two 3D vectors
  - $V_1 \cdot V_2 = (dy_1dx_2 - dz_1dy_2, dz_1dx_2 - dx_1dz_2, dx_1dy_2 - dy_1dx_2)$
  - $V_1 x V_2$ = vector perpendicular to both V_1 and V_2
  - $||V_1 x V_2|| = ||V_1|| \, ||V_2|| \sin(\Theta)$



3D Line Segment

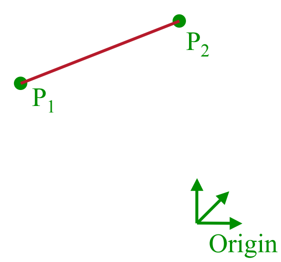
- Linear path between two points



3D Line Segment

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

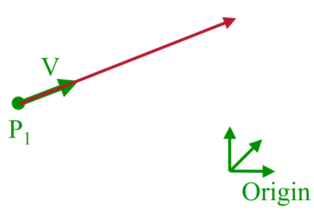
```
typedef struct {
    Point P1;
    Point P2;
} Segment;
```



3D Ray

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

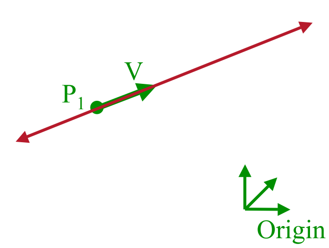
```
typedef struct {
    Point P1;
    Vector V;
} Ray;
```



3D Line

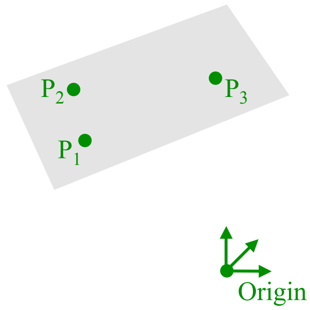
- Line segment with both endpoints at infinity
  - Parametric representation:
    - » $P = P_1 + t\,V$, $(-\infty < t < \infty)$

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



3D Plane

- A linear combination of three points

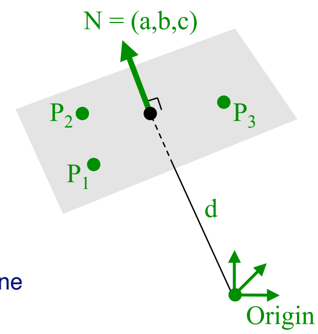


3D Plane

- 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



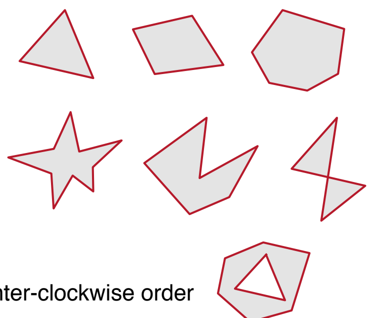
3D Polygon

- 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

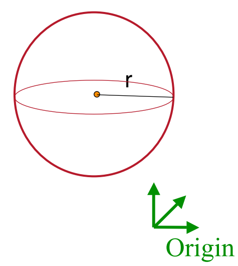
  - Holes (use > 1 polygon struct)



3D Sphere

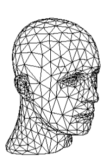
- All points at distance "r" from point "(c_x, c_y, c_z)"
  - Implicit representation:
    - » $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r^2$
  - Parametric representation:
    - » $x = r \cos(\phi) \cos(\Theta) + c_x$
    - » $y = r \cos(\phi) \sin(\Theta) + c_y$
    - » $z = r \sin(\phi) + c_z$

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

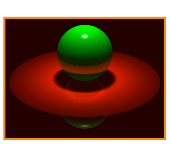


3D Scenes

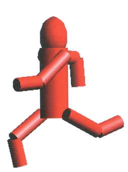
- Comprise set of geometric primitives



(Dennis Zorin, CalTech)



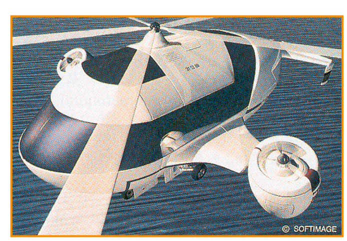
(Michael Bostock, CS426, Fall99)



(Angel, Plate 1)

Other Geometric Primitives

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

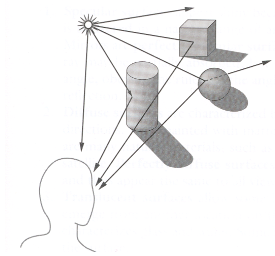


H&B Figure 10.46

Overview

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

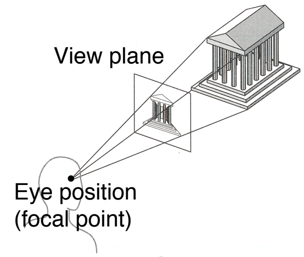
How is the viewing device described in a computer?



Camera Models

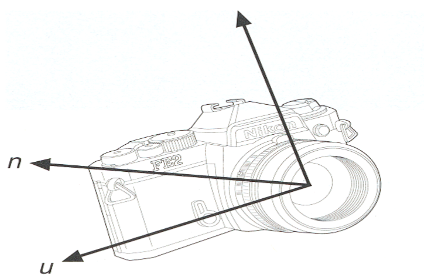
- The most common model is pin-hole camera
  - o All captured light rays arrive along paths toward focal point without lens distortion (everything is in focus)
  - o Sensor response proportional to radiance

Other models consider ...
  Depth of field
  Motion blur
  Lens distortion



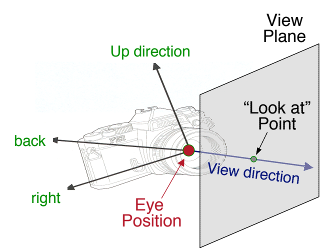
Camera Parameters

- What are the parameters of a camera?

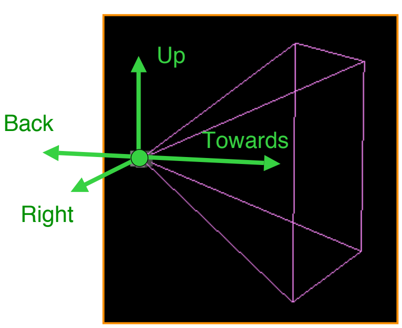


Camera Parameters

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



View Frustum

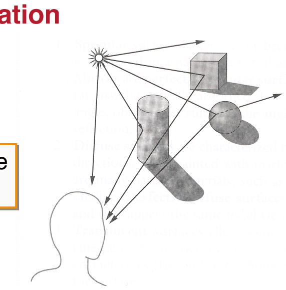


View Frustum

Overview

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

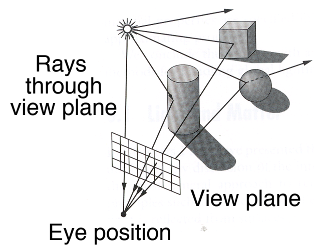
How can the front-most surface be found with an algorithm?



Visible Surface Determination

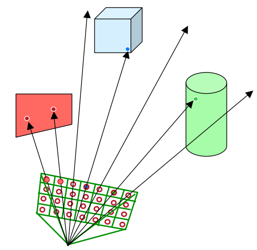
- The color of each pixel on the view plane depends on the radiance emanating from visible surfaces

Simplest method is ray casting



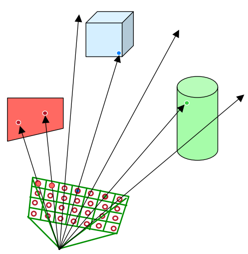
Ray Casting

- 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

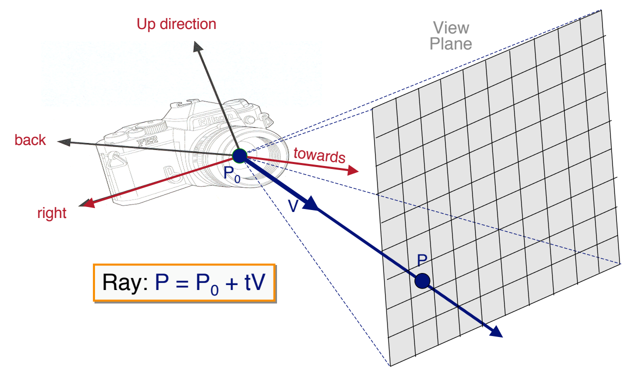


Ray Casting

- 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

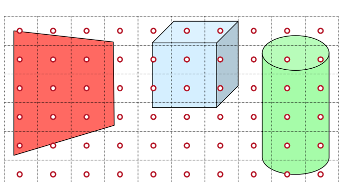


Construct Ray

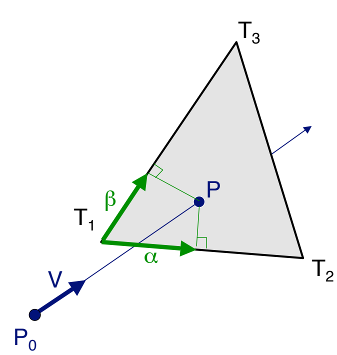


Ray Casting

- 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

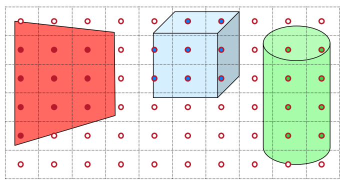


Find First Surface Intersection



Visible Surface Determination

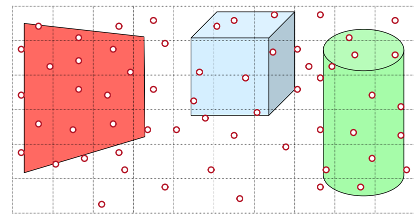
- For each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - ➢ Compute color of sample based on surface radiance



More efficient algorithms utilize spatial coherence!

Rendering Algorithms

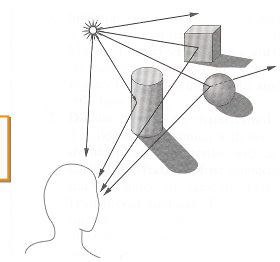
- Any samples can be used!
  - Rendering is a problem in sampling and reconstruction



Overview

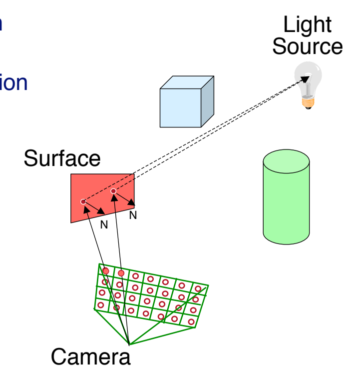
- 3D scene representation
- 3D viewer representation
- Visible surface determination
- » **Lighting simulation**

How do we compute the radiance for each sample ray?

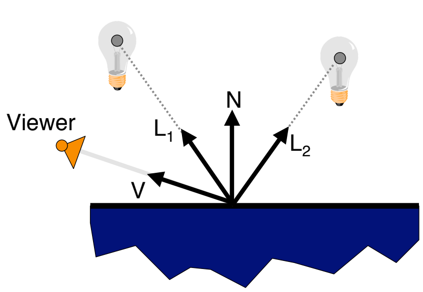


Lighting Simulation

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



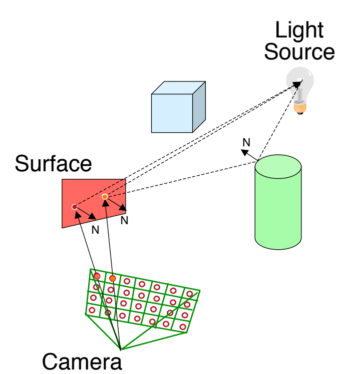
Lighting Simulation



Lighting Simulation

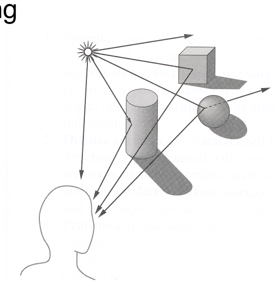
- Direct illumination
  - Ray casting
  - Polygon shading
- Global illumination
  - Ray tracing
  - Monte Carlo methods
  - Radiosity methods

More on these methods later!



Summary

- 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

- Ray intersections
- Light and reflectance models
- Indirect illumination

Tricycle
(James Percy, CS 426, Fall99)

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