3D Rendering

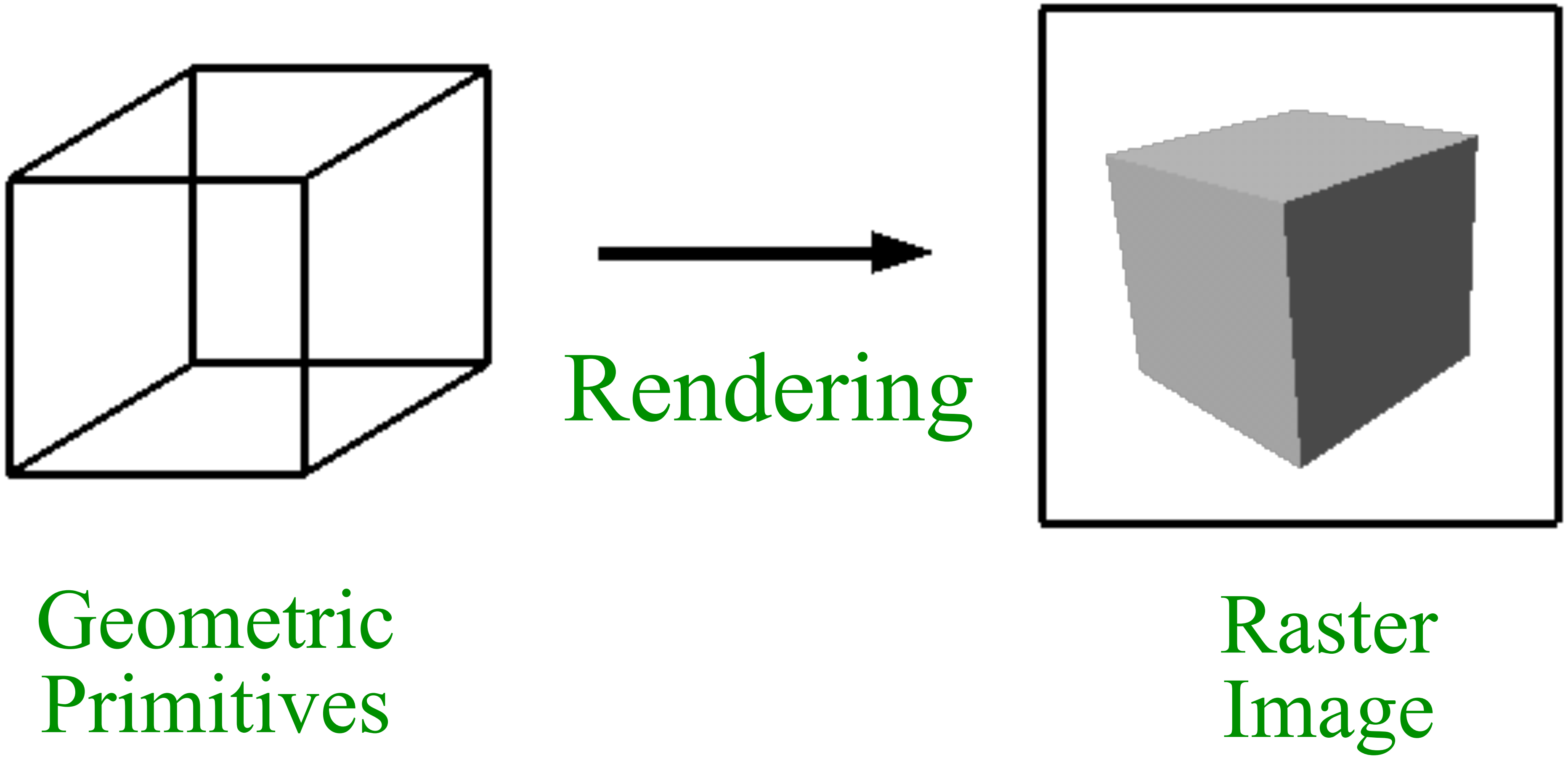
Connelly Barnes

CS 4810: Graphics

Acknowledgment: slides by Jason Lawrence, Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin

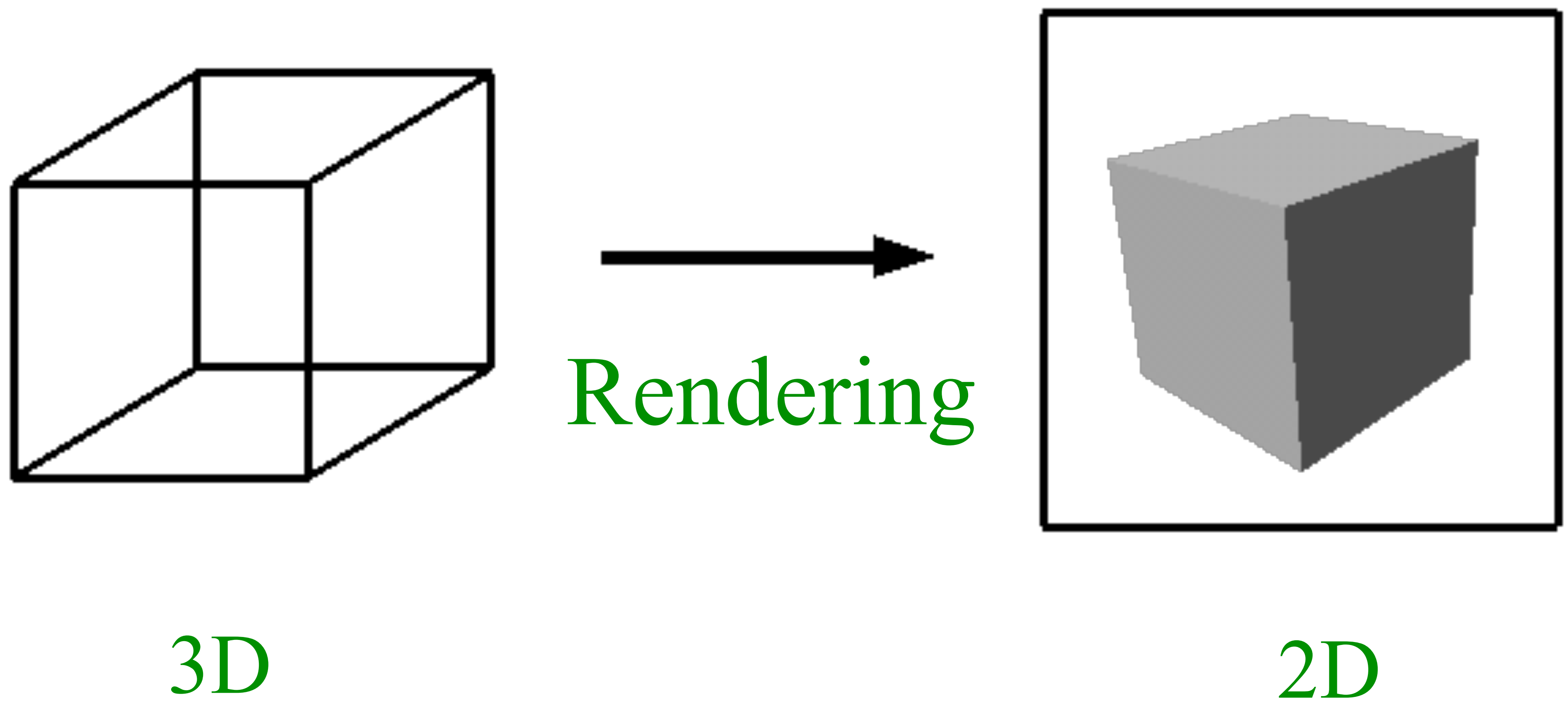
Rendering

- Generate an image from geometric primitives

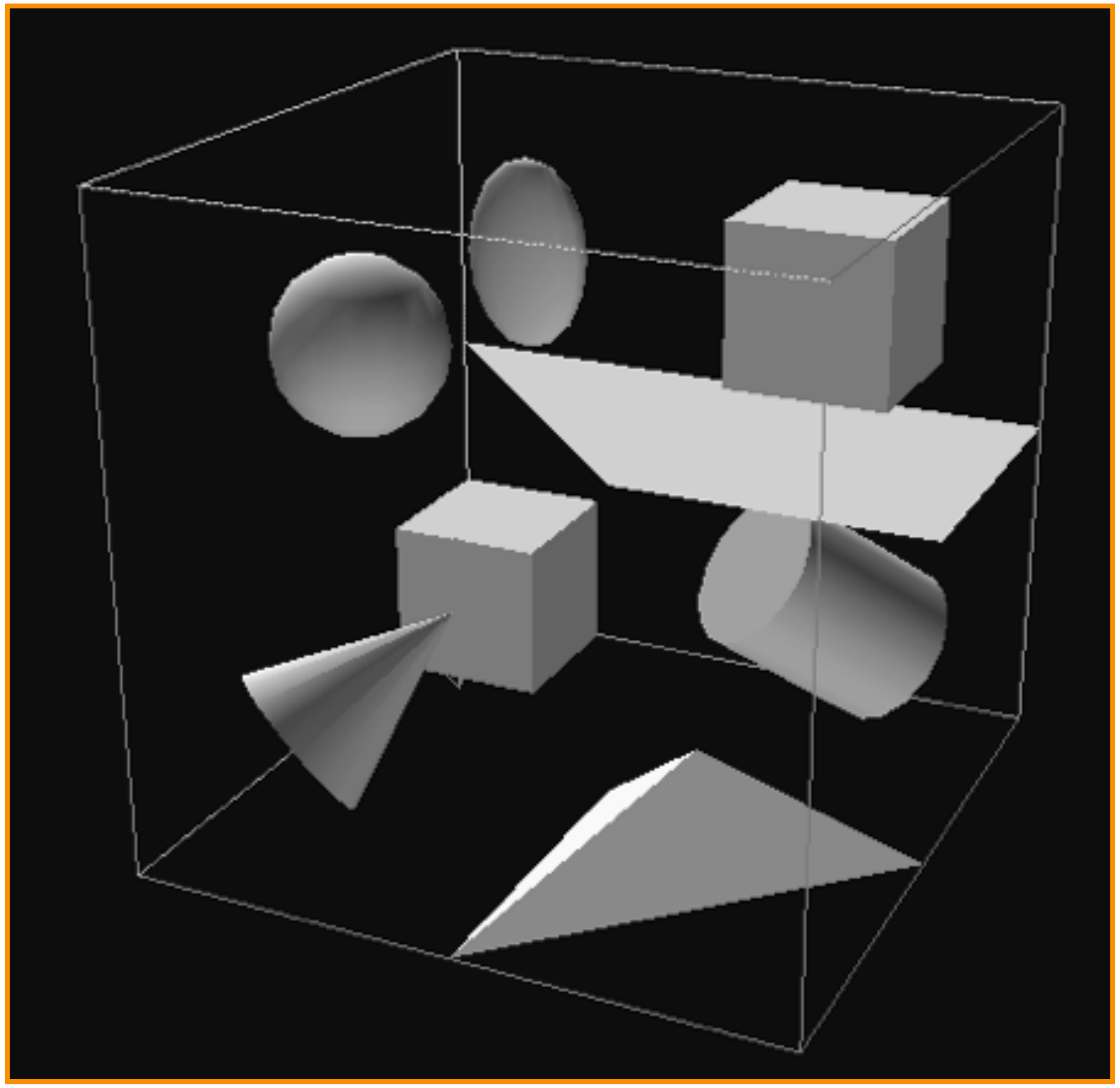


Rendering

- Generate an image from geometric primitives



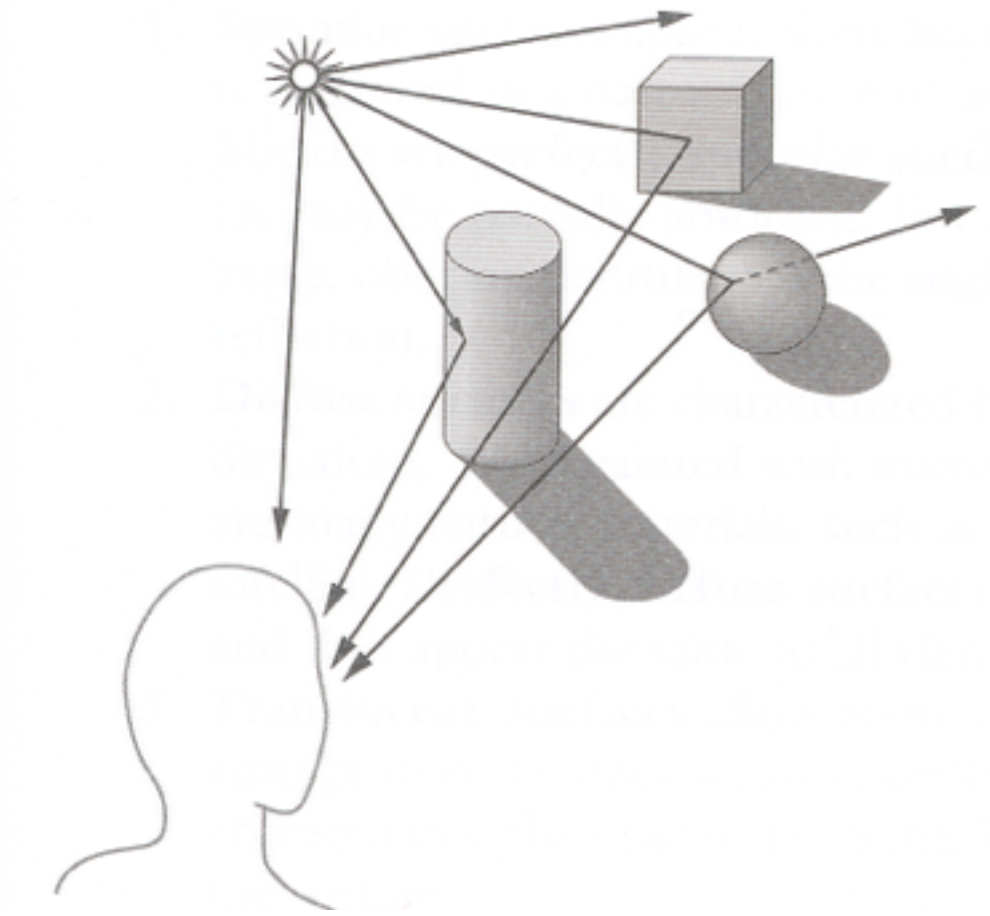
3D Rendering Example



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

Overview

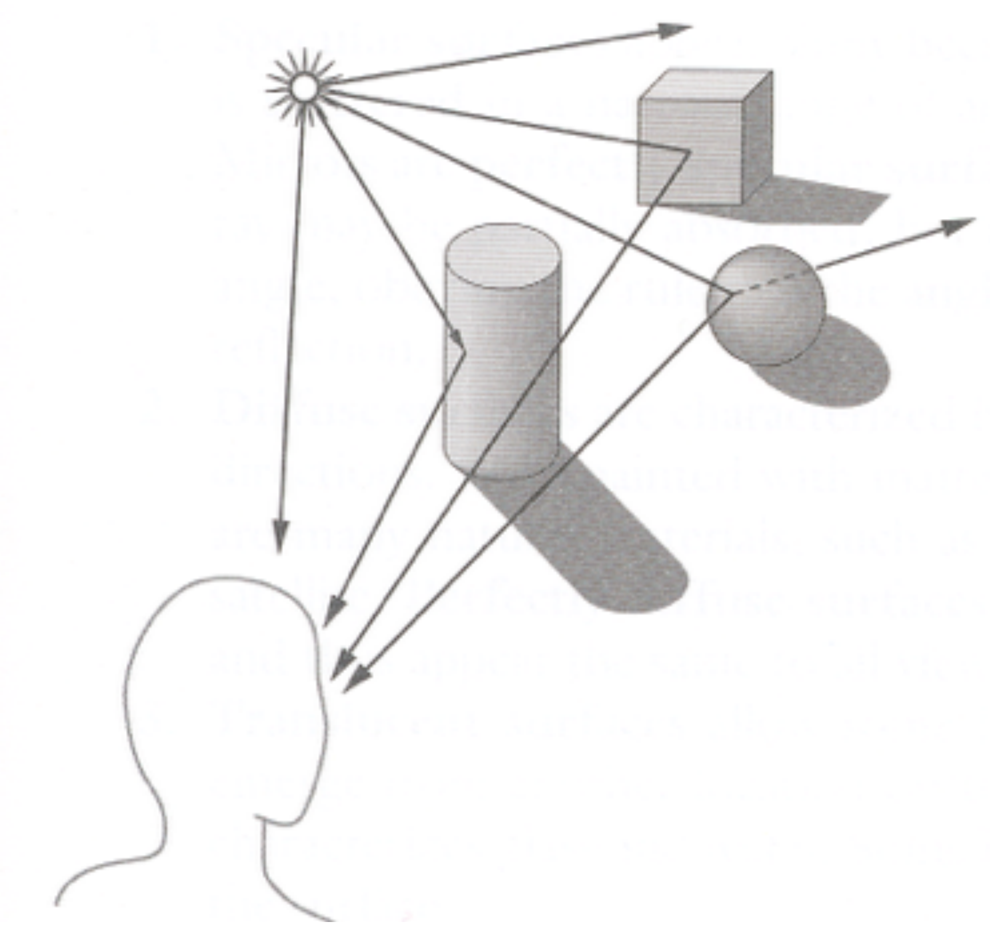
- 3D scene representation
- 3D viewer representation
- Visible surface determination
- Lighting simulation



Overview

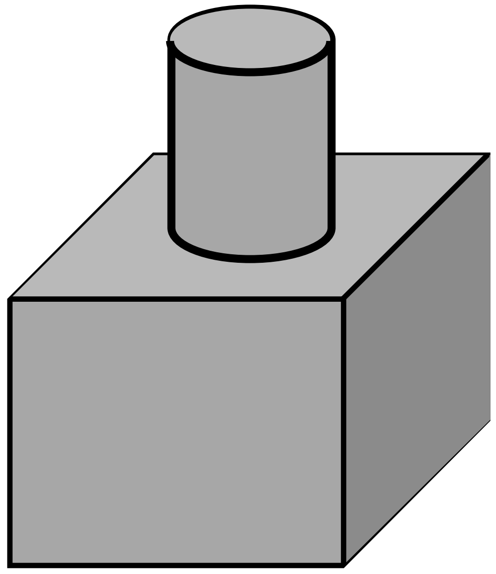
- 3D scene representation
- 3D viewer representation
- Visible surface determination
- Lighting simulation

How is the 3D scene described in a computer?



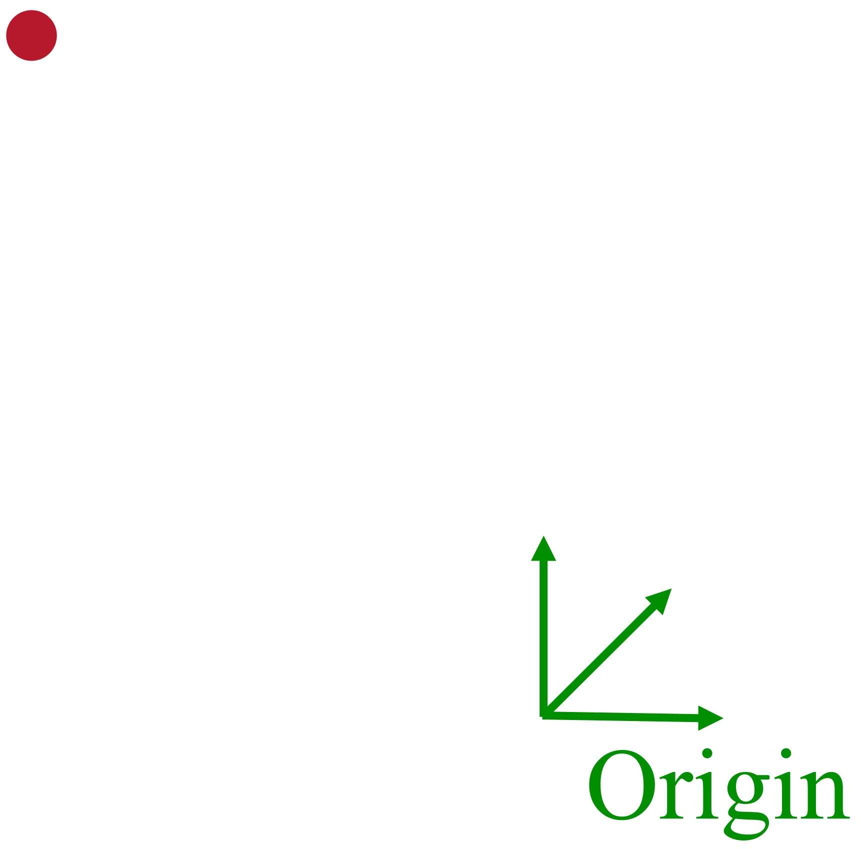
3D Scene Representation

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



3D Point

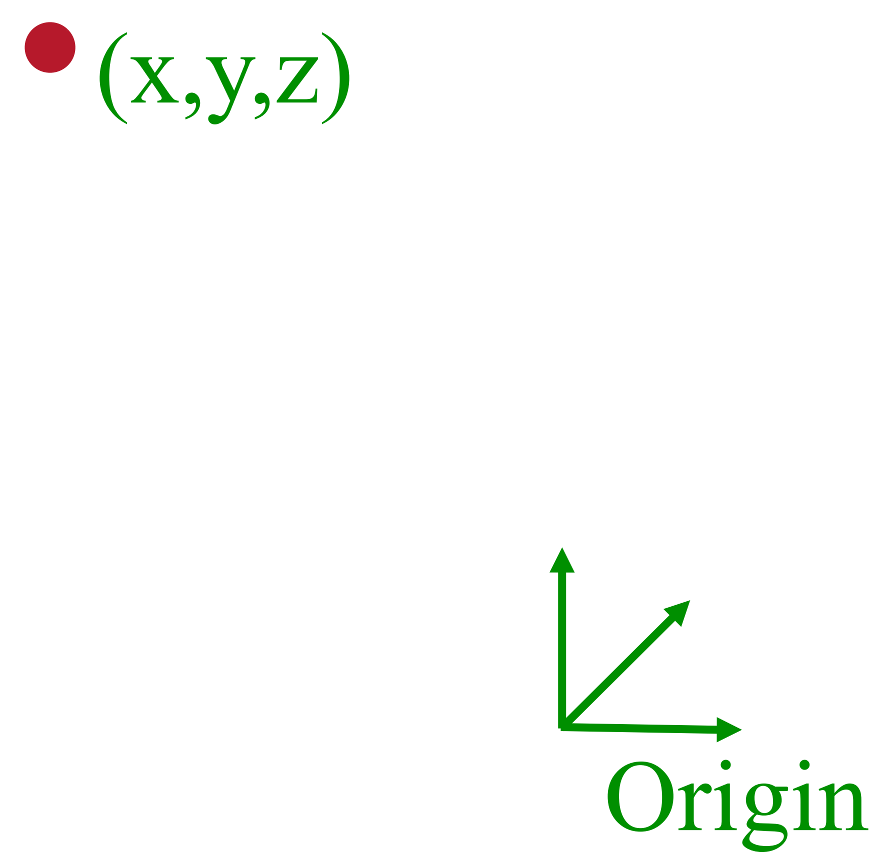
- Specifies a location



3D Point

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

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

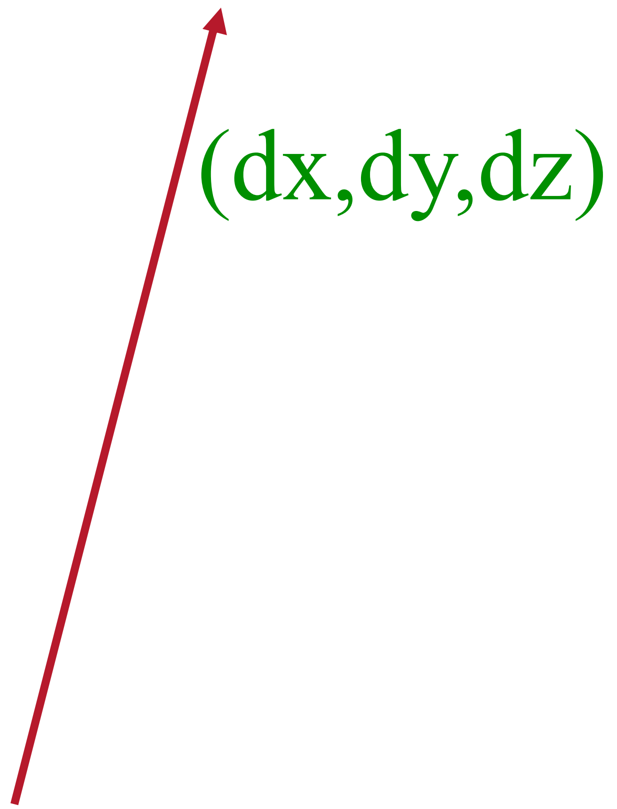


3D Vector

- 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

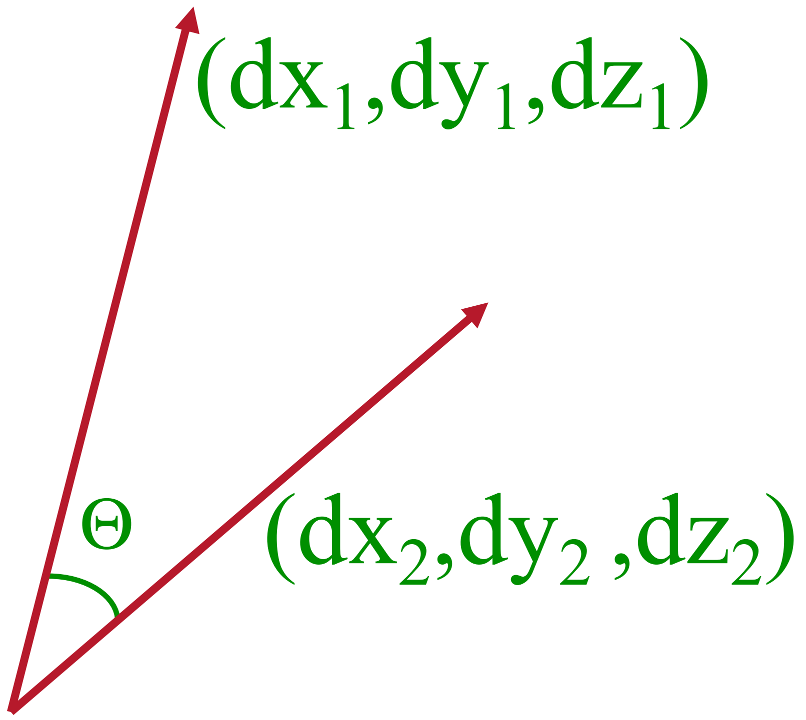



```
typedef struct {
    Coordinate dx;
    Coordinate dy;
    Coordinate dz;
} Vector;
```

3D Vector

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

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



- Dot product of two 3D vectors
  - $V_1 \cdot V_2 = dx_1 dx_2 + dy_1 dy_2 + dz_1 dz_2$
  - $V_1 \cdot V_2 = ||V_1||\ ||V_2|| \cos(\Theta)$

Linear Algebra: a Little Review

- What is…?
- $V_1 \cdot V_1 = ?$

Linear Algebra: a Little Review

- What is…?
- $V_1 \cdot V_1 = dx\,dx + dy\,dy + dz\,dz$

Linear Algebra: a Little Review

- What is…?
- $V_1 \cdot V_1 = (\text{Magnitude})^2$

Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 =$

Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = ||V_1|| \; || V_1 || \cos(\Theta)$

Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = ||V_1|| \; || V_1 || \cos(\Theta) = \cos(\Theta)$

Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = \|V_1\| \, \|V_1\| \cos(\Theta) = \cos(\Theta) = \cos(0)$

Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = 1$

Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = 1$
- $V_1 \cdot V_2 =$

Linear Algebra: a Little Review

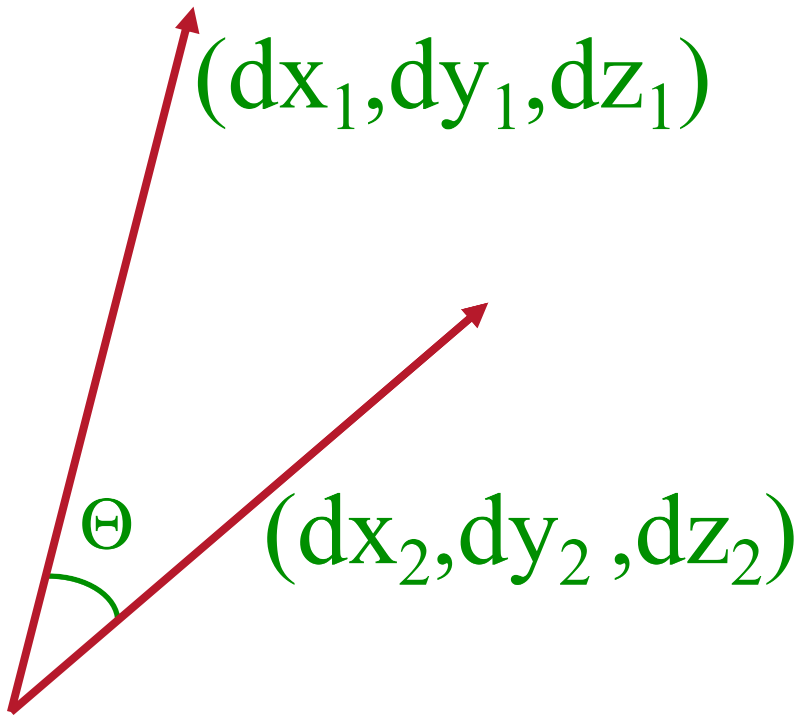
- $V_1 \cdot V_1$ = (Magnitude)^2
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = 1$
- $V_1 \cdot V_2 = ||V_1|| \; || V_2 || \cos(\Theta)$

Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = 1$
- $V_1 \cdot V_2 = ||V_1|| \; || V_2 || \cos(\Theta) = \cos(\Theta)$

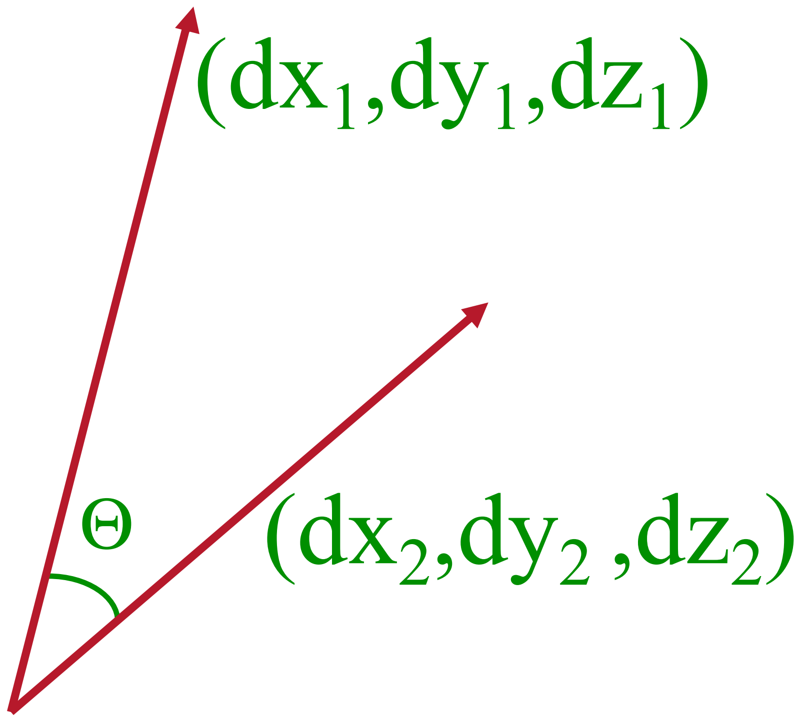
Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = 1$
- $V_1 \cdot V_2 = \cos(\Theta) = (\text{adjacent} / \text{hyp})$



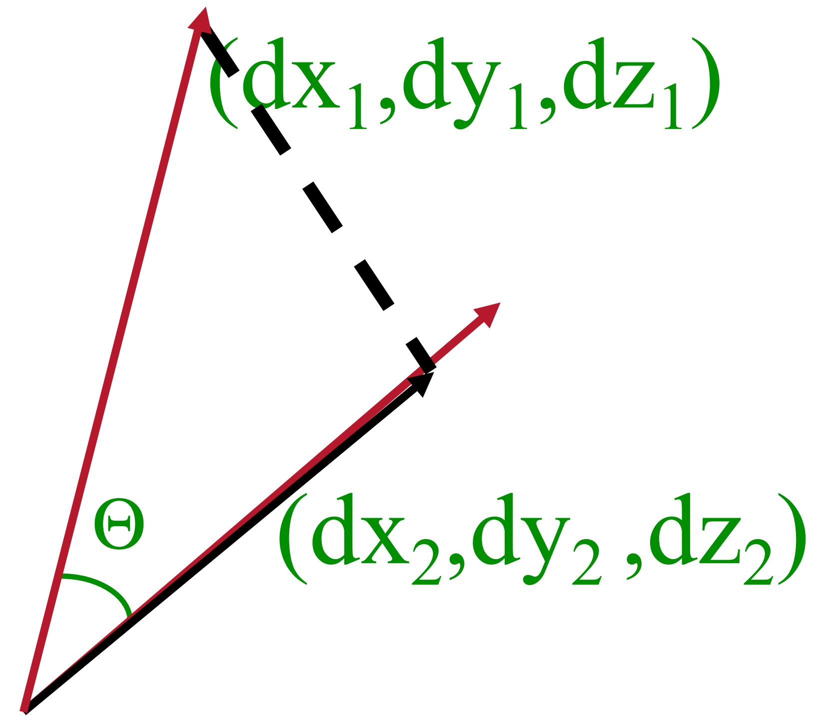
Linear Algebra: a Little Review

- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = 1$
- $V_1 \cdot V_2 = (\text{adjacent} / 1)$



Linear Algebra: a Little Review

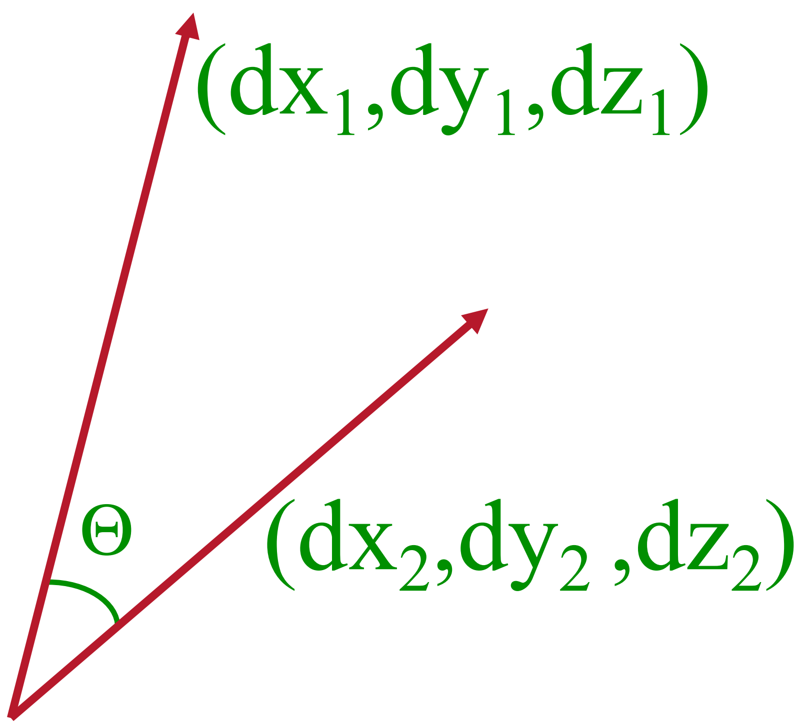
- $V_1 \cdot V_1 = (\text{Magnitude})^2$
- Now, let V_1 and V_2 both be unit-length vectors.
- What is…?
- $V_1 \cdot V_1 = 1$
- $V_1 \cdot V_2$ = length of V_1 projected onto V_2 (or vice-versa)



3D Vector

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

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



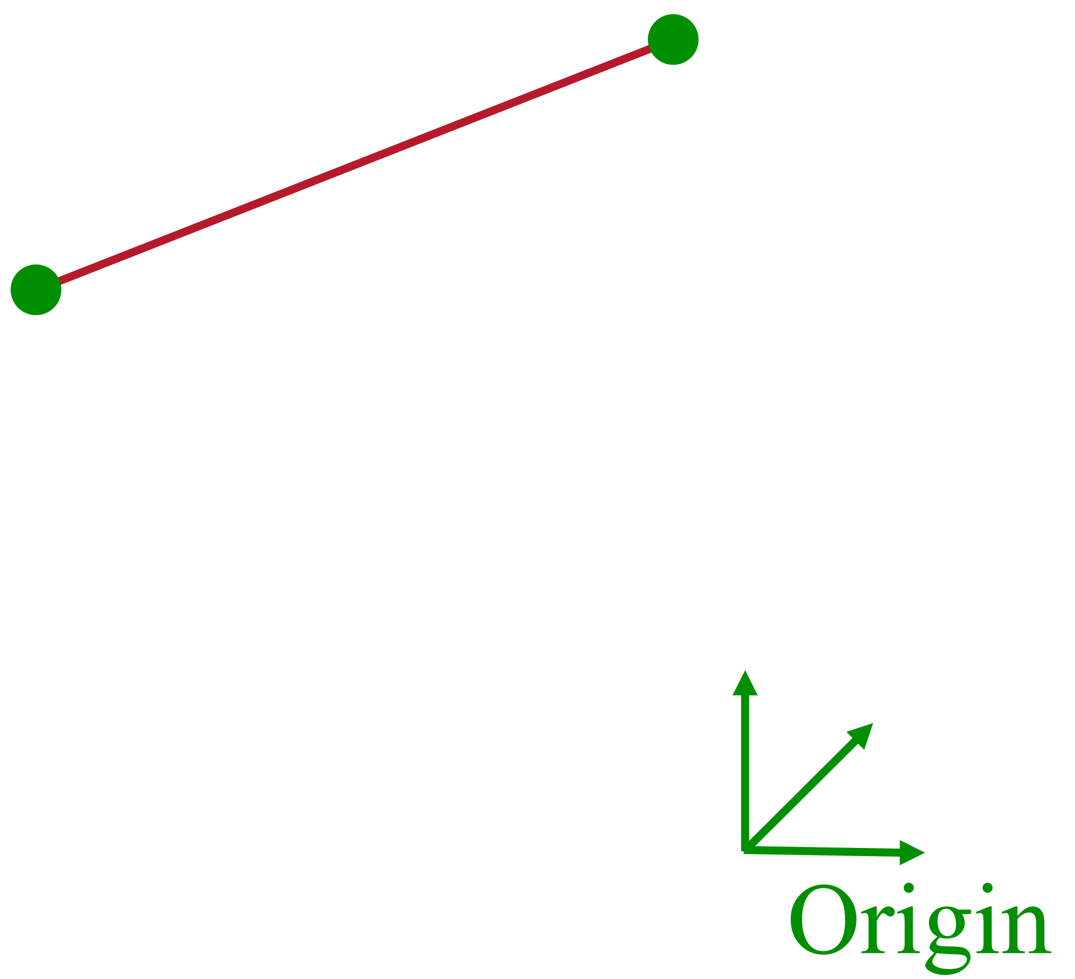
- Cross product of two 3D vectors
  - $V_1 \times V_2$ = Vector normal to plane V_1 , V_2
  - $||\ V_1 \times V_2\ ||\ = ||V_1||\ ||\ V_2\ ||\ \sin(\Theta)$

Linear Algebra: More Review

- Let $C = A \times B$:
  - $C_x = A_y B_z - A_z B_y$
  - $C_y = A_z B_x - A_x B_z$
  - $C_z = A_x B_y - A_y B_x$
- $A \times B = - B \times A$ (remember "right-hand" rule)
- We can do similar derivations to show:
  - $V_1 \times V_2 = \|V_1\| \, \|V_2\| \sin(\Theta) \, n$, where n is unit vector normal to V_1 and V_2
  - $\| V_1 \times V_1 \| = 0$
  - $\| V_1 \times (-V_1) \| = 0$
- http://physics.syr.edu/courses/java-suite/crosspro.html

3D Line Segment

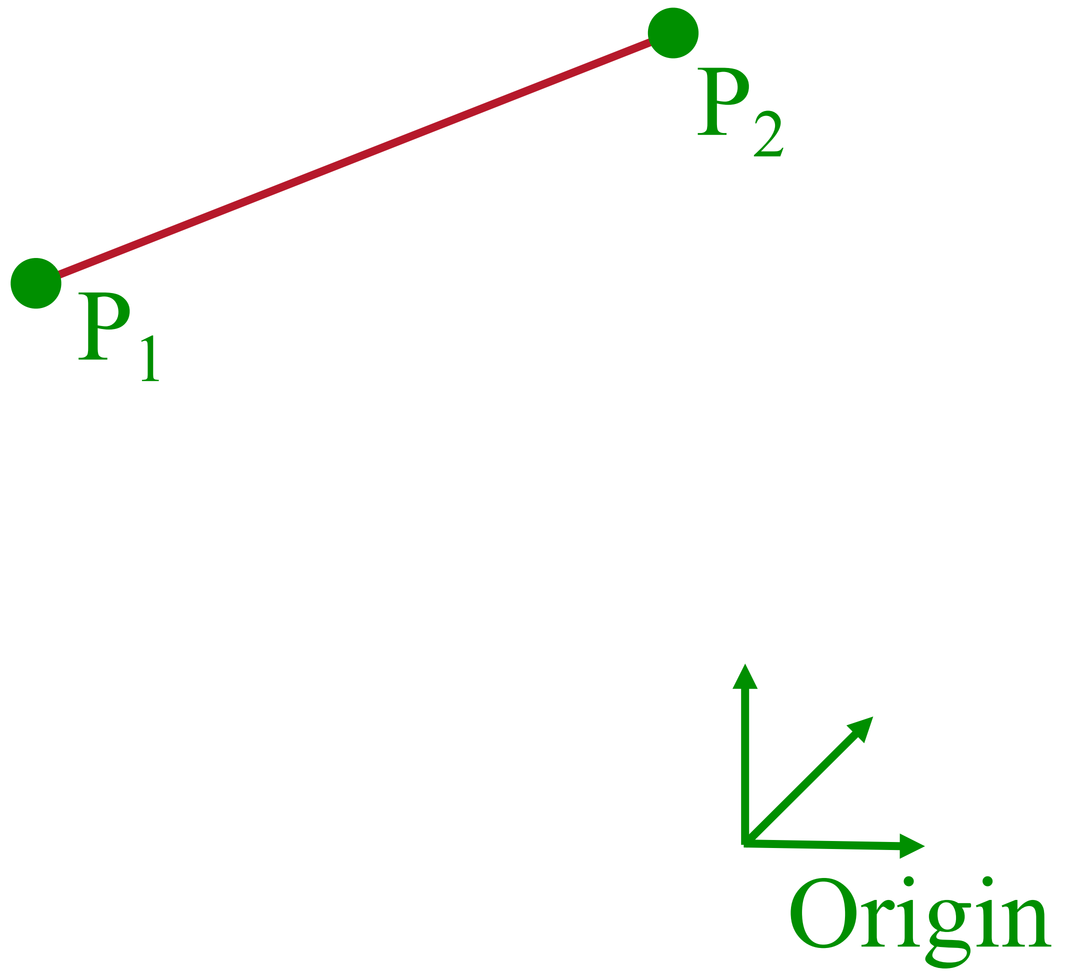
- Linear path between two points



3D Line Segment

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

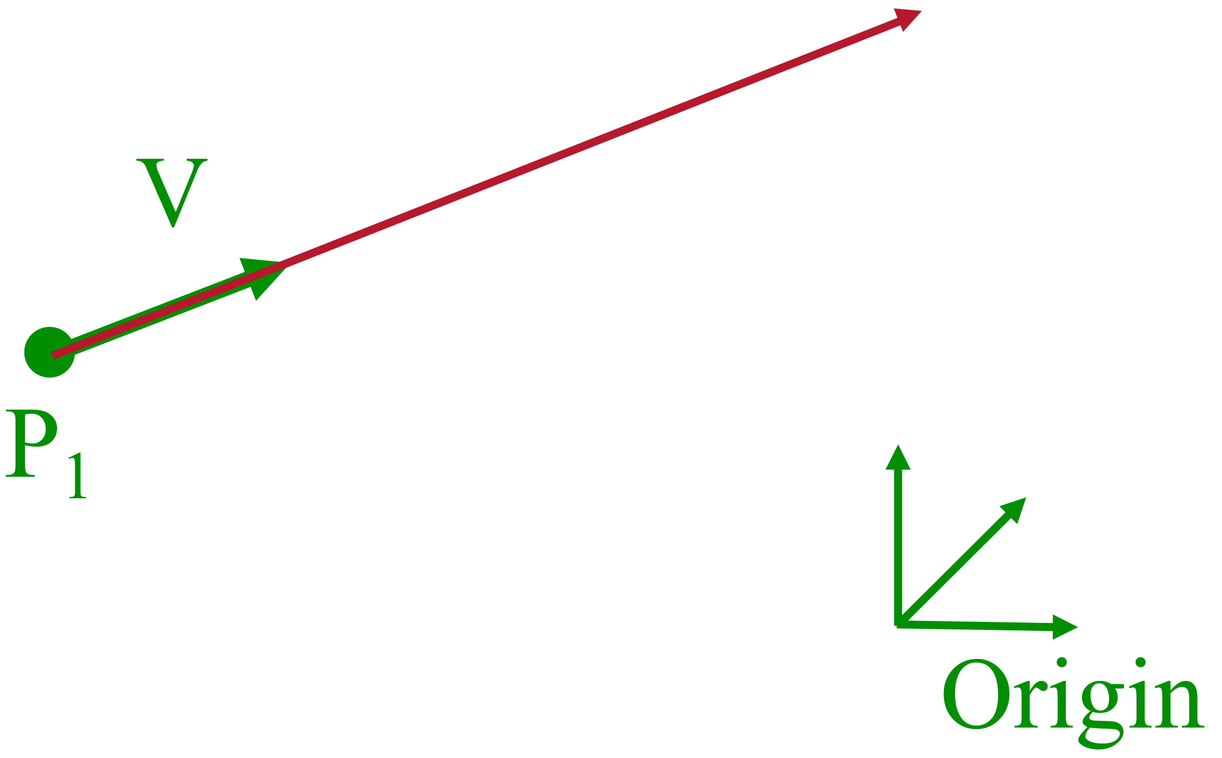
```
typedef struct {
    Point P1;
    Point P2;
} Segment;
```



3D Ray

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

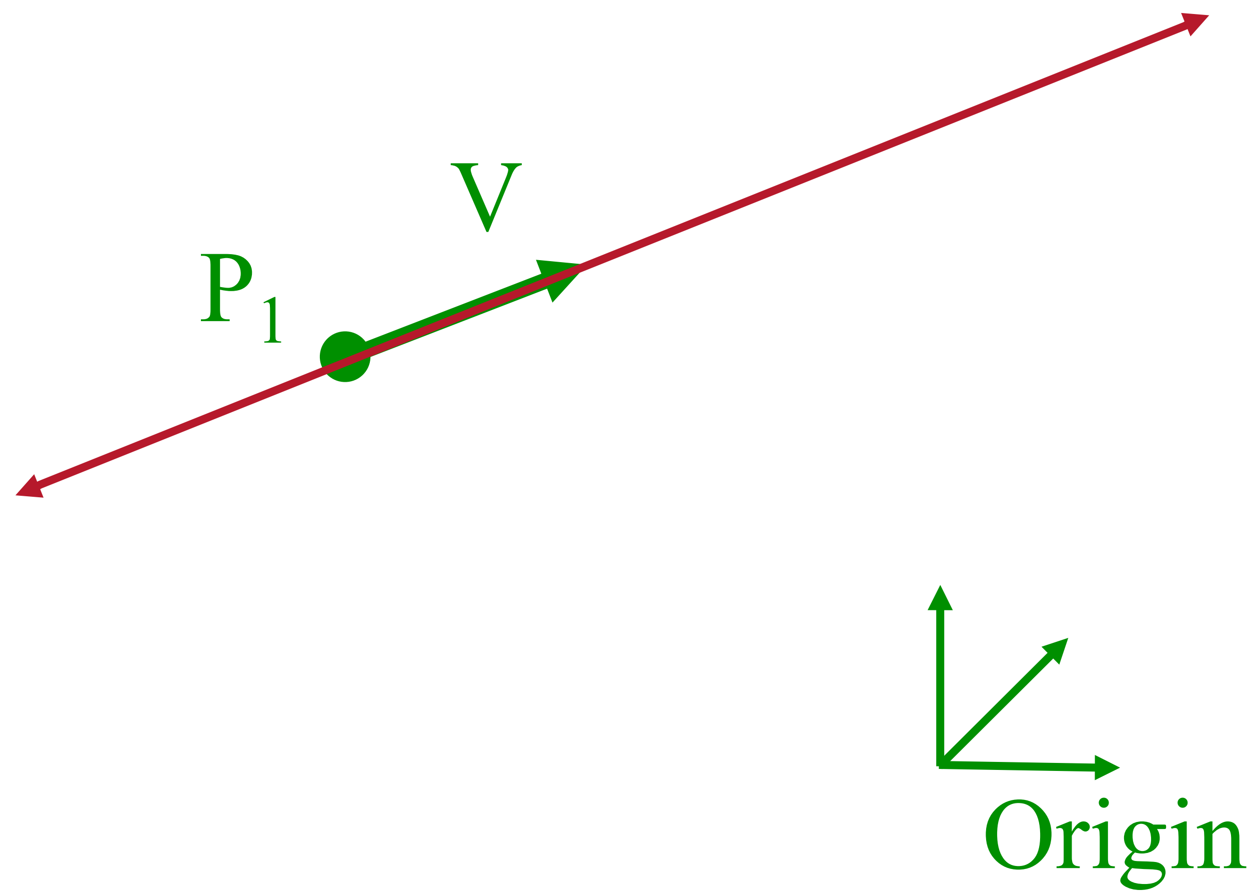
```
typedef struct {
    Point P1;
    Vector V;
} Ray;
```



3D Line

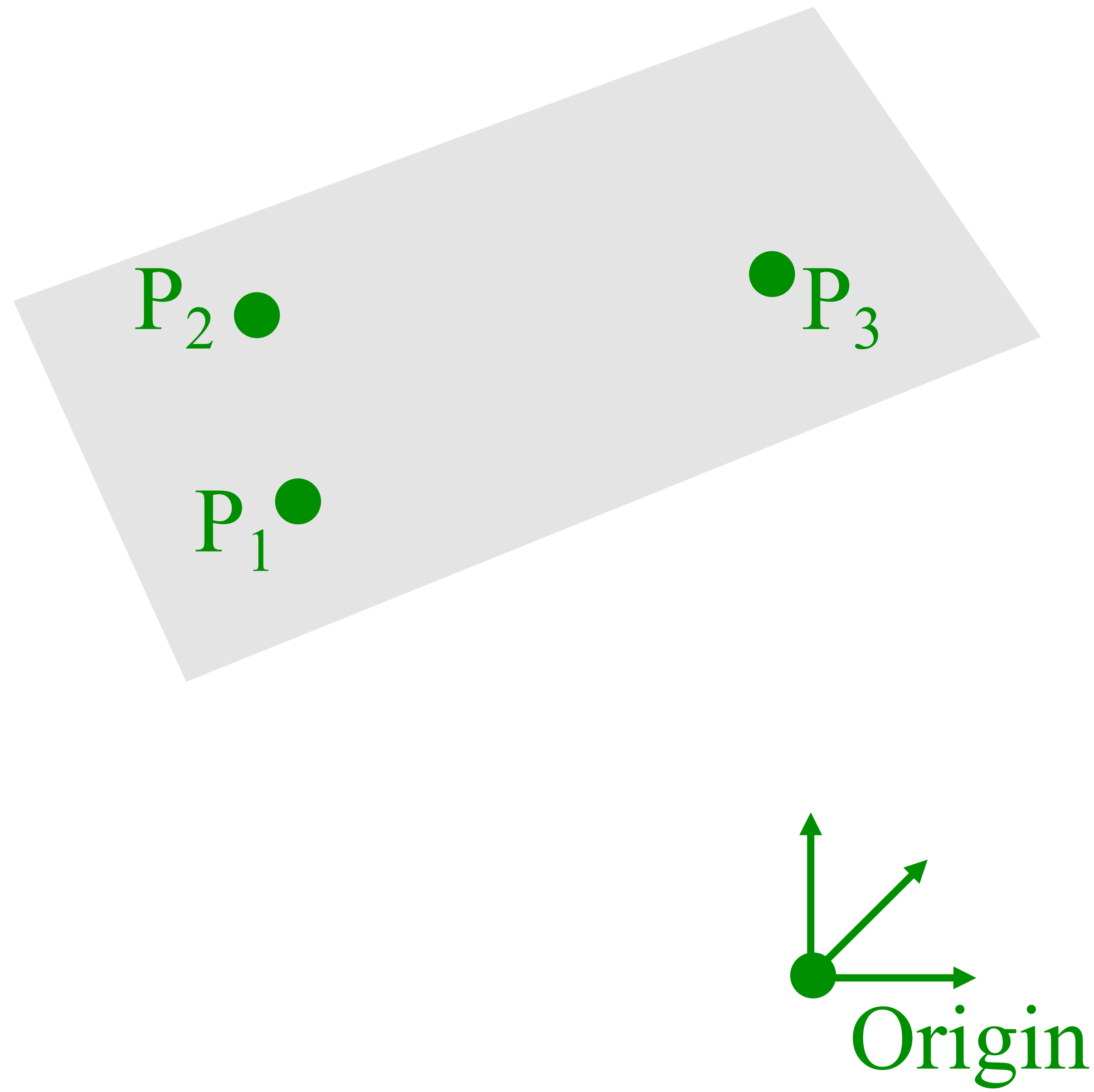
- Line segment with both endpoints at infinity
  - o Parametric representation:
    - » $P = P_1 + t\,V, \quad (-\infty < t < \infty)$

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



3D Plane

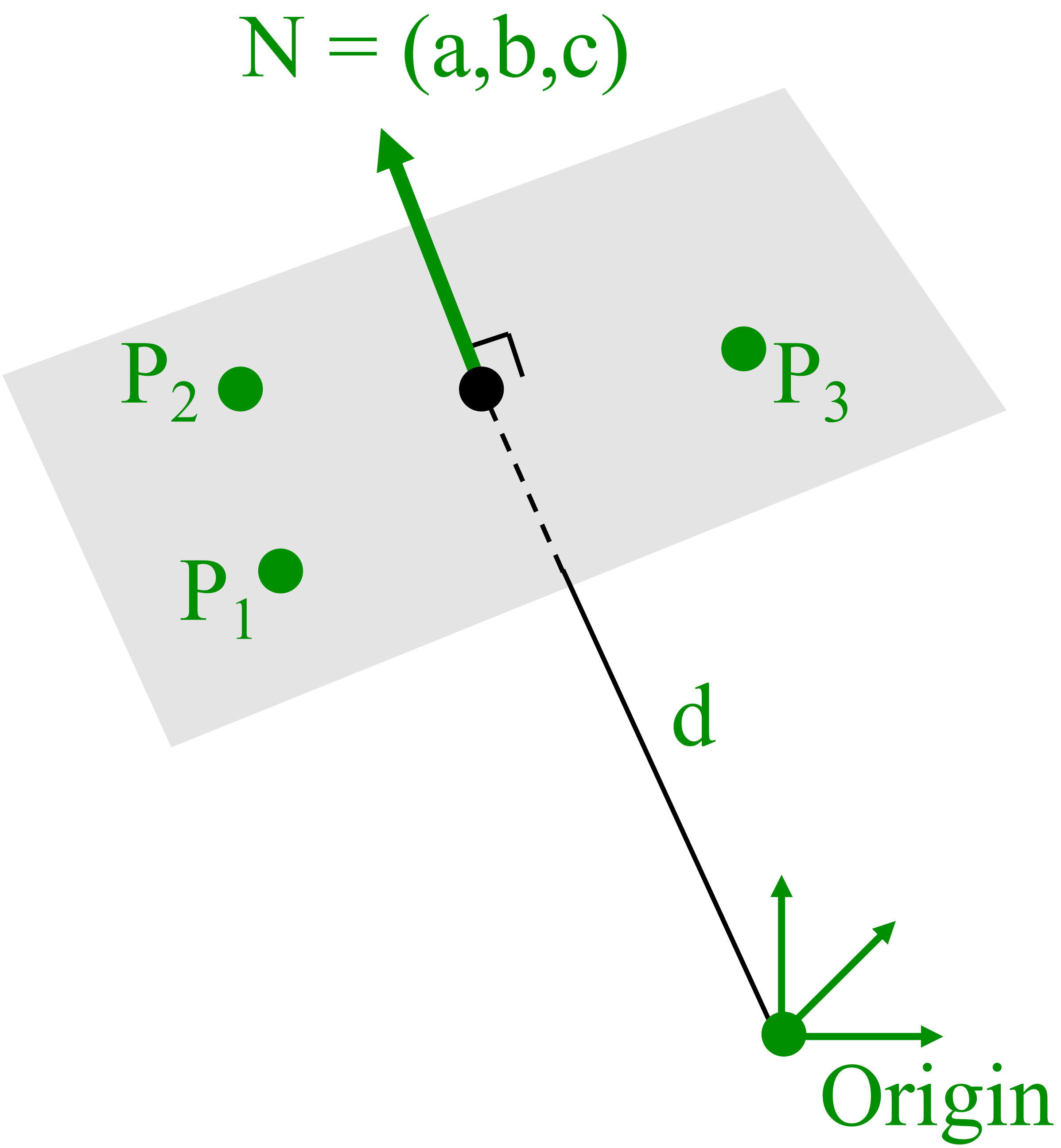
- A linear combination of three points



3D Plane

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

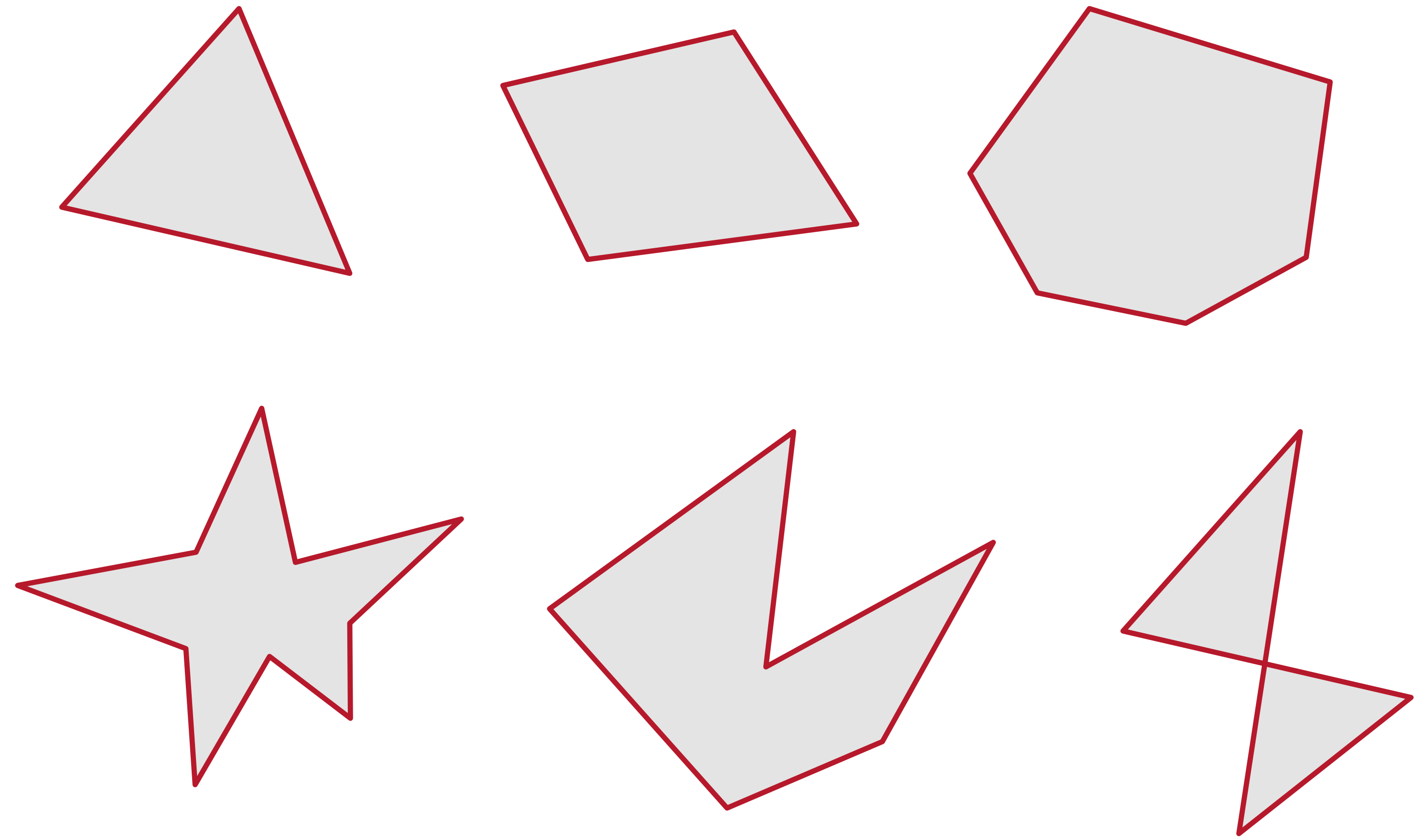
```
typedef struct {
    Vector N;
    Distance d;
} Plane;
```



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

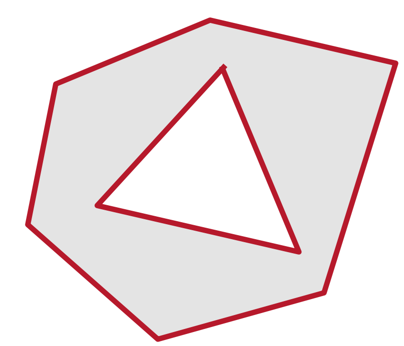
3D Polygon

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



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

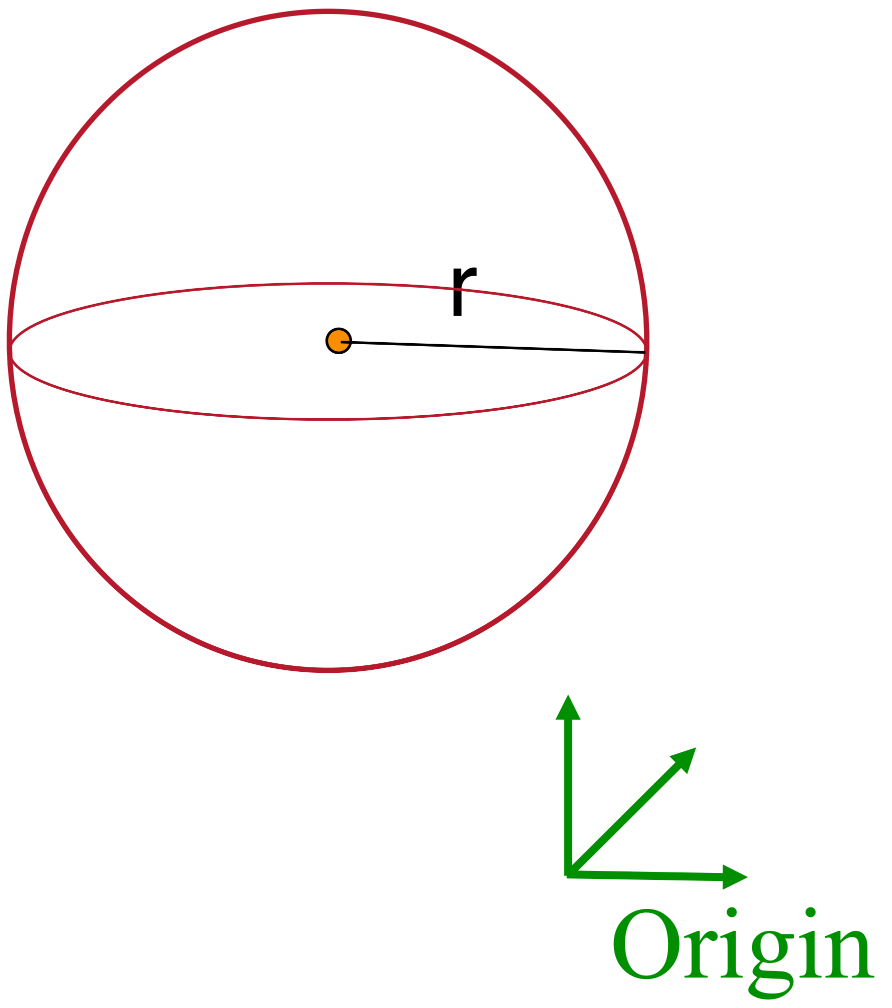
Points are in counter-clockwise order



  - Holes (use > 1 polygon struct)

3D Sphere

- All points at distance "r" from point "(c_x, c_y, c_z)"
  - **o** Implicit representation:
    - » $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r^2$
  - **o** 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;
```

Other 3D primitives

- Cone
- Cylinder
- Ellipsoid
- Box
- Etc.

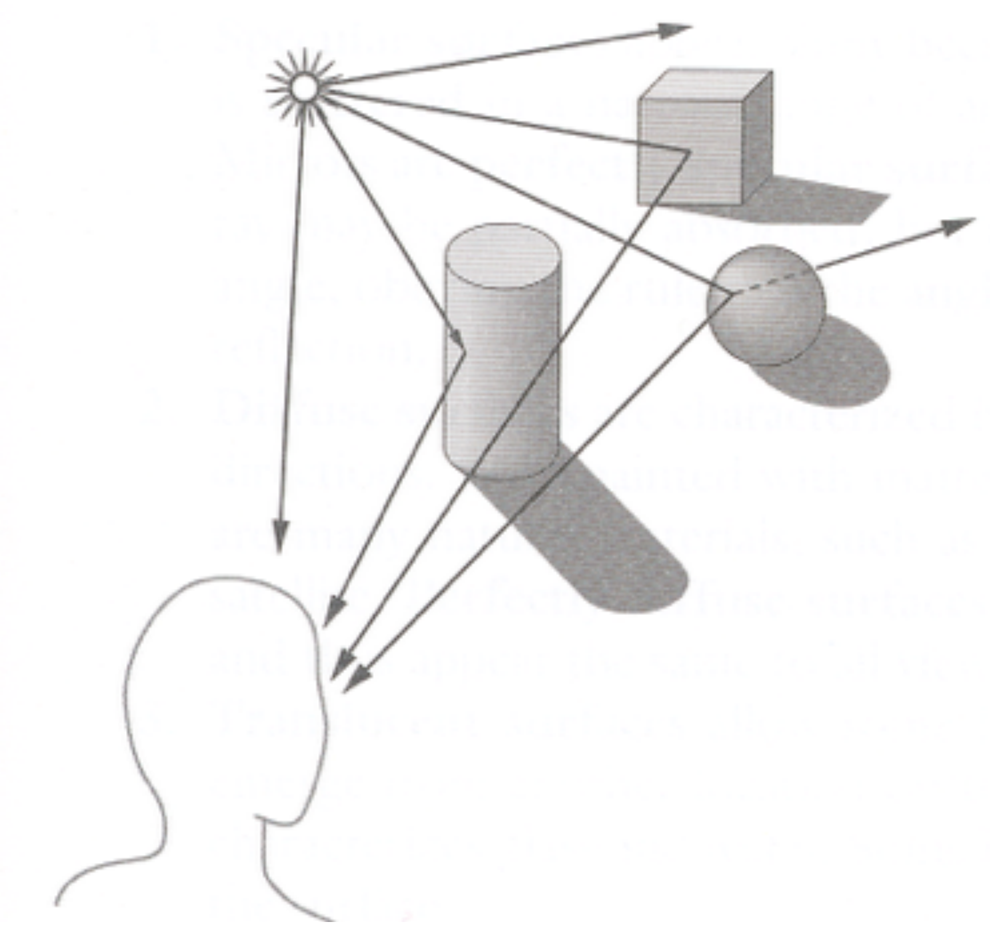
3D Geometric Primitives

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

Overview

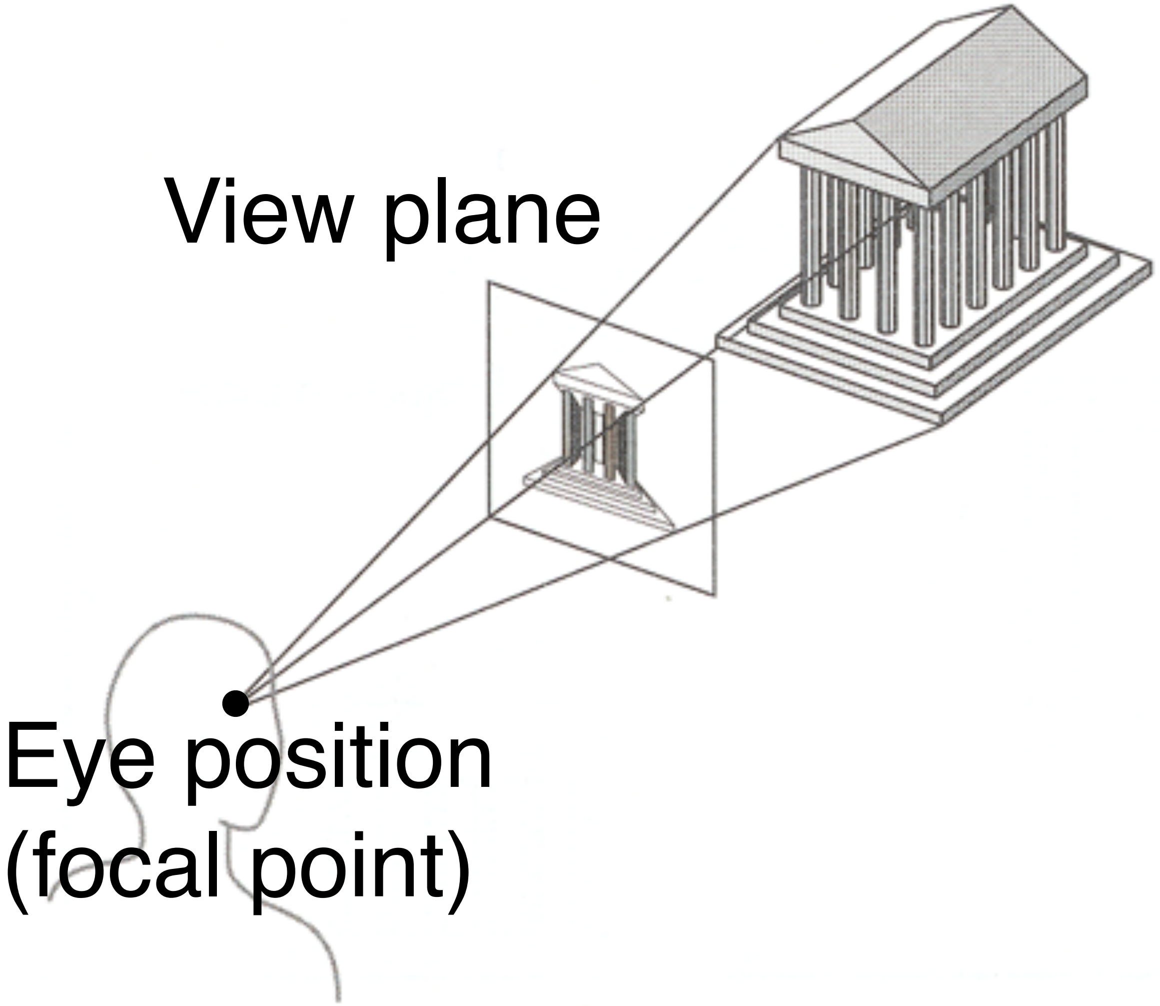
- 3D scene representation
- 3D viewer representation
- Visible surface determination
- Lighting simulation

How is the viewing device described in a computer?



Camera Models

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

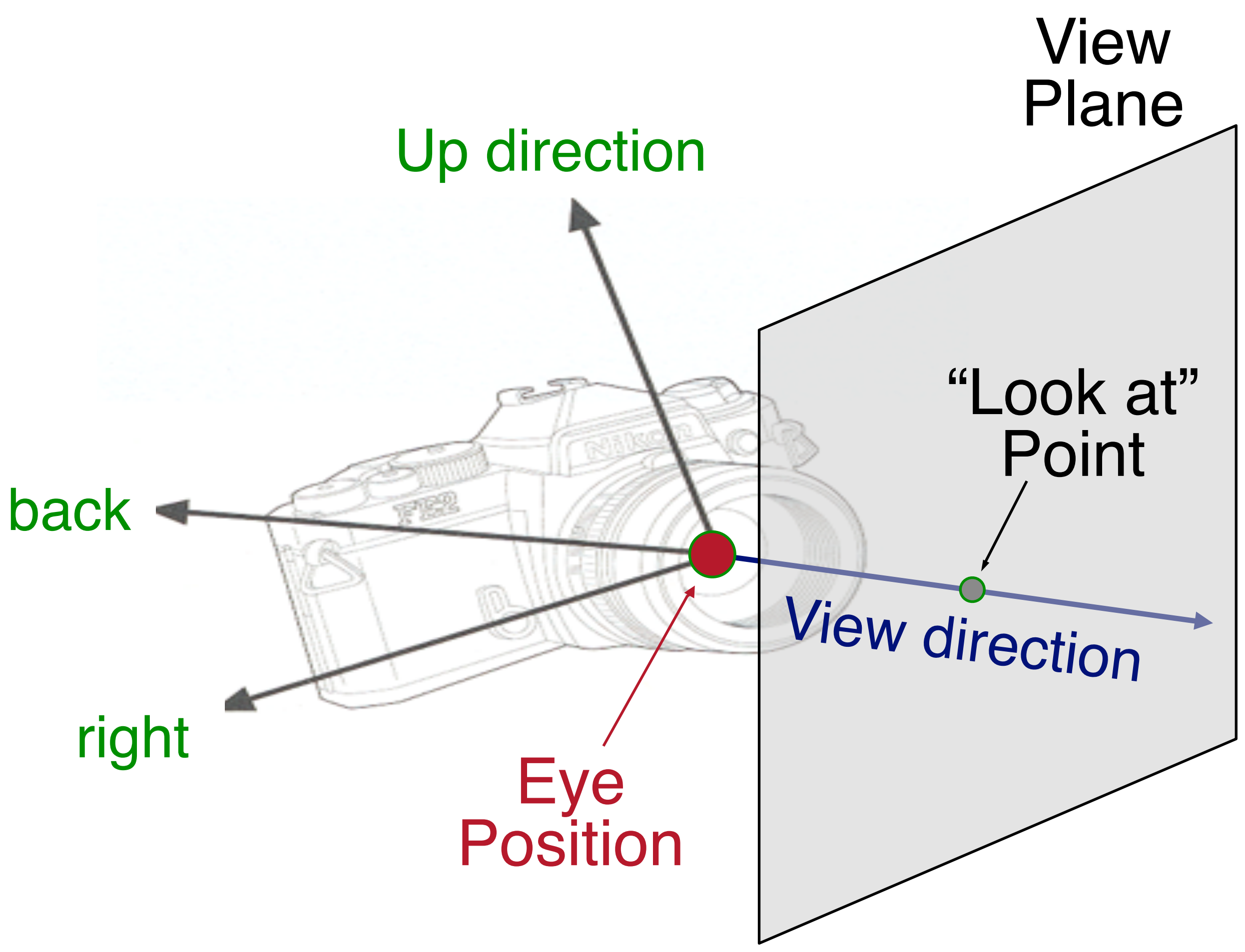


Camera Parameters

- What are the parameters of a camera?

Camera Parameters

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



Other Models: Depth of Field

Close Focused

Distance Focused

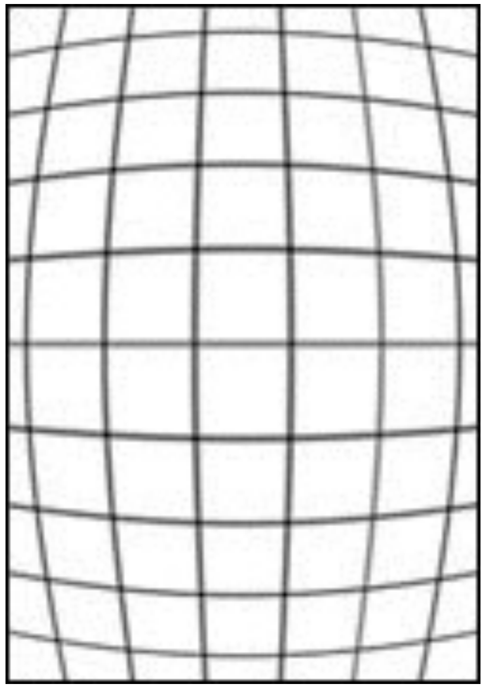
P. Haeberli

Other Models: Motion Blur

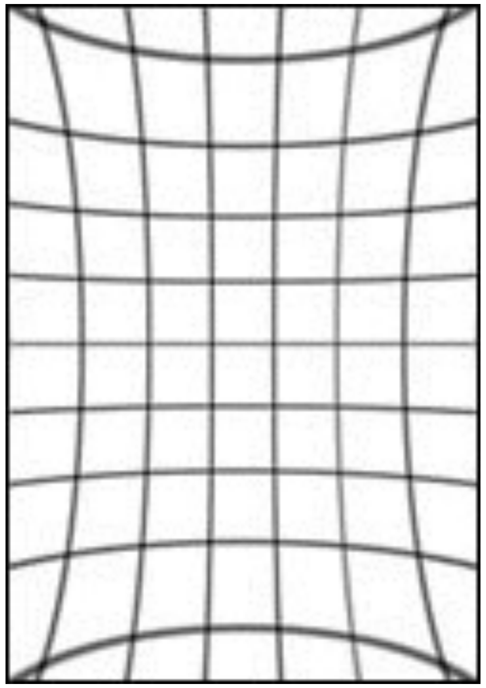
- Mimics effect of open camera shutter
- Gives perceptual effect of high-speed motion
- Generally involves temporal super-sampling

Other Models: Lens Distortion

- Camera lens bends light, especially at edges
- Common types are barrel and pincushion



Barrel Distortion



Pincushion Distortion

Other Models: Lens Distortion

- Camera lens bends light, especially at edges
- Common types are barrel and pincushion

Barrel Distortion

No Distortion

Other Models: Lens Distortion

Lens flares are another kind of distortion

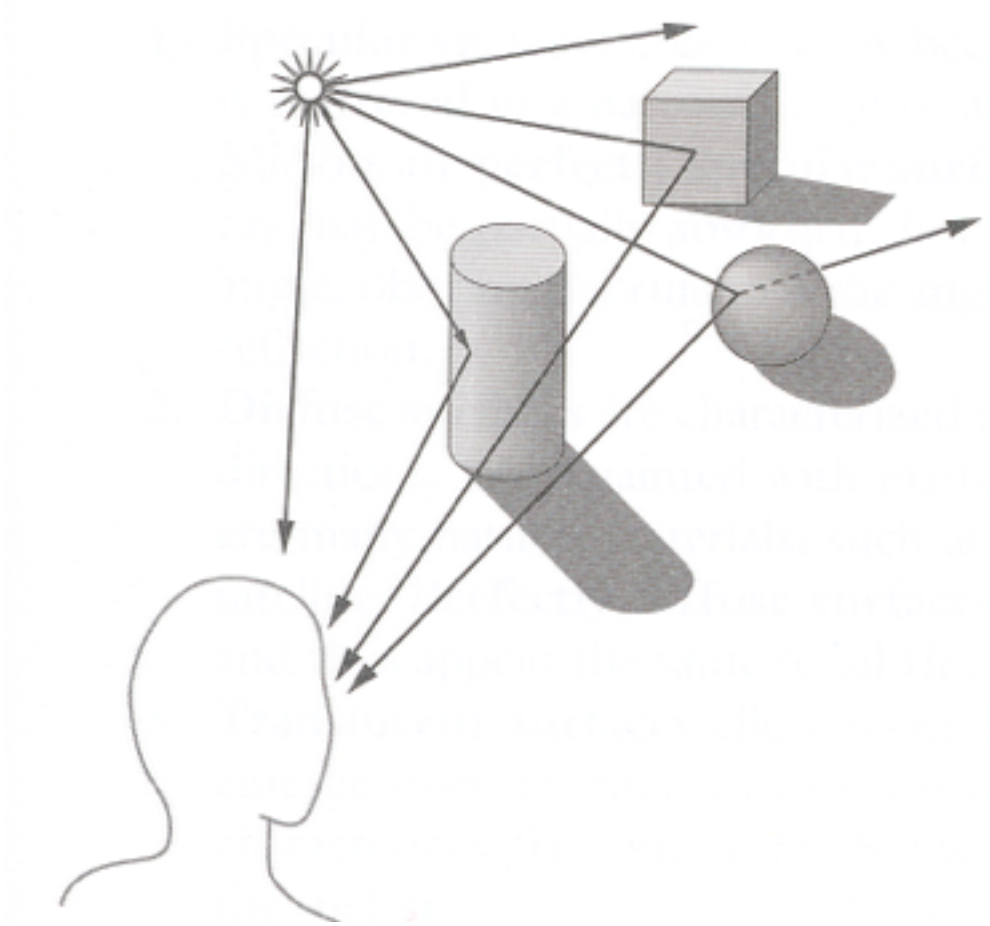
Star Wars: Knights of the Old Republic

(BioWare)

Overview

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

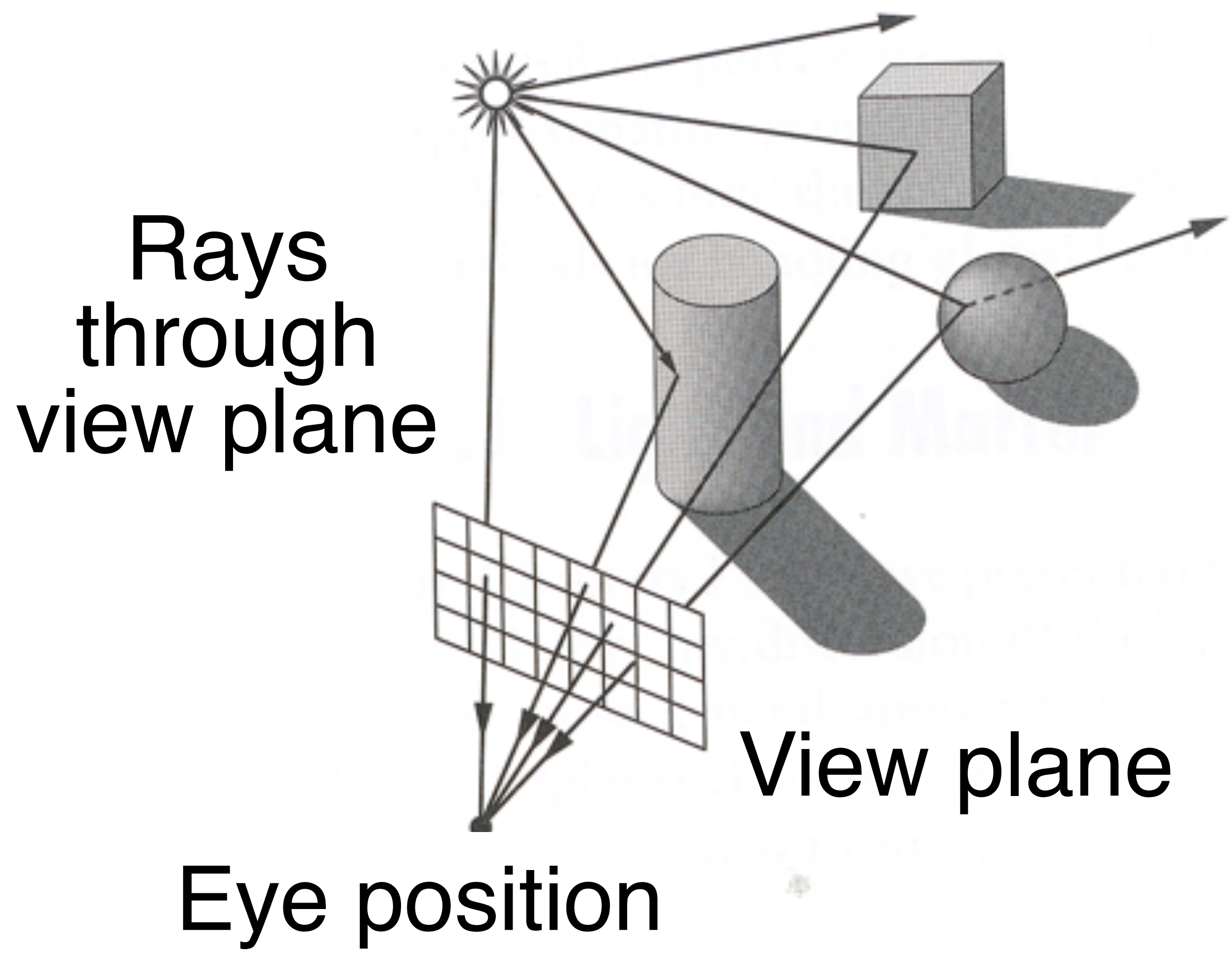
How can the front-most surface be found with an algorithm?



Visible Surface Determination

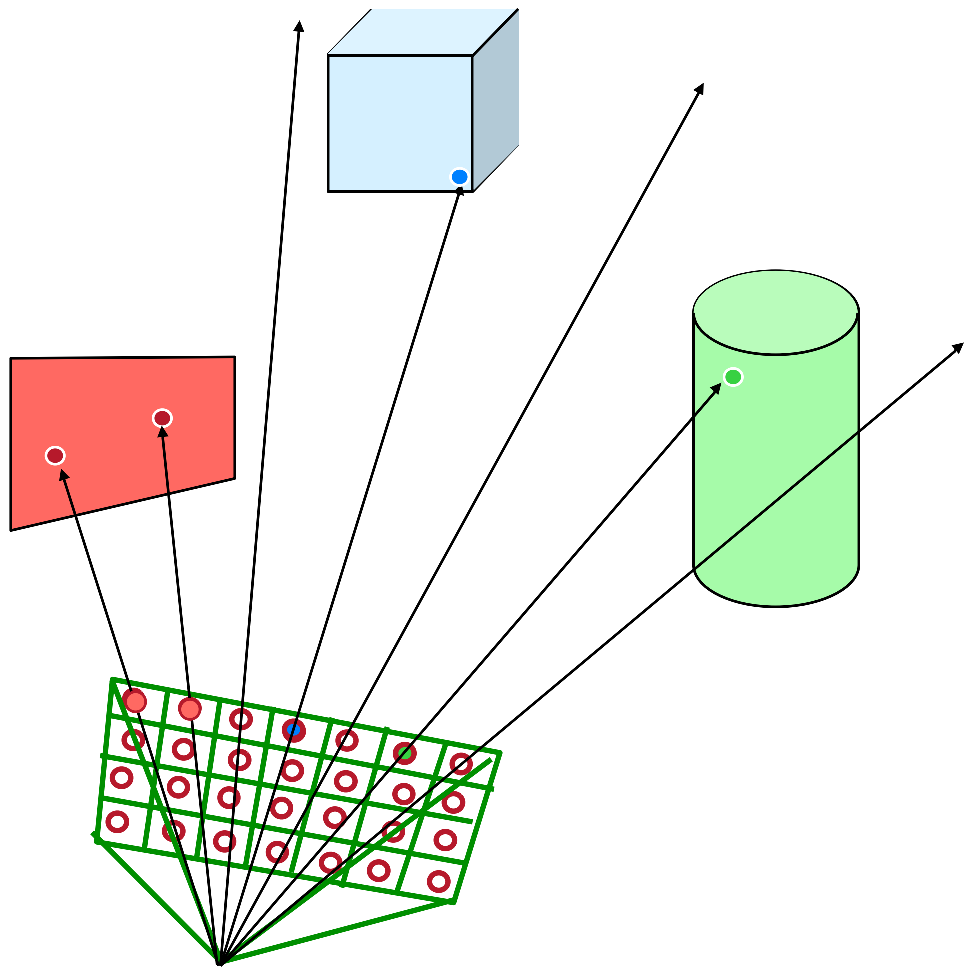
- The color of each pixel on the view plane depends on the radiance emanating from visible surfaces

Simplest method is ray casting



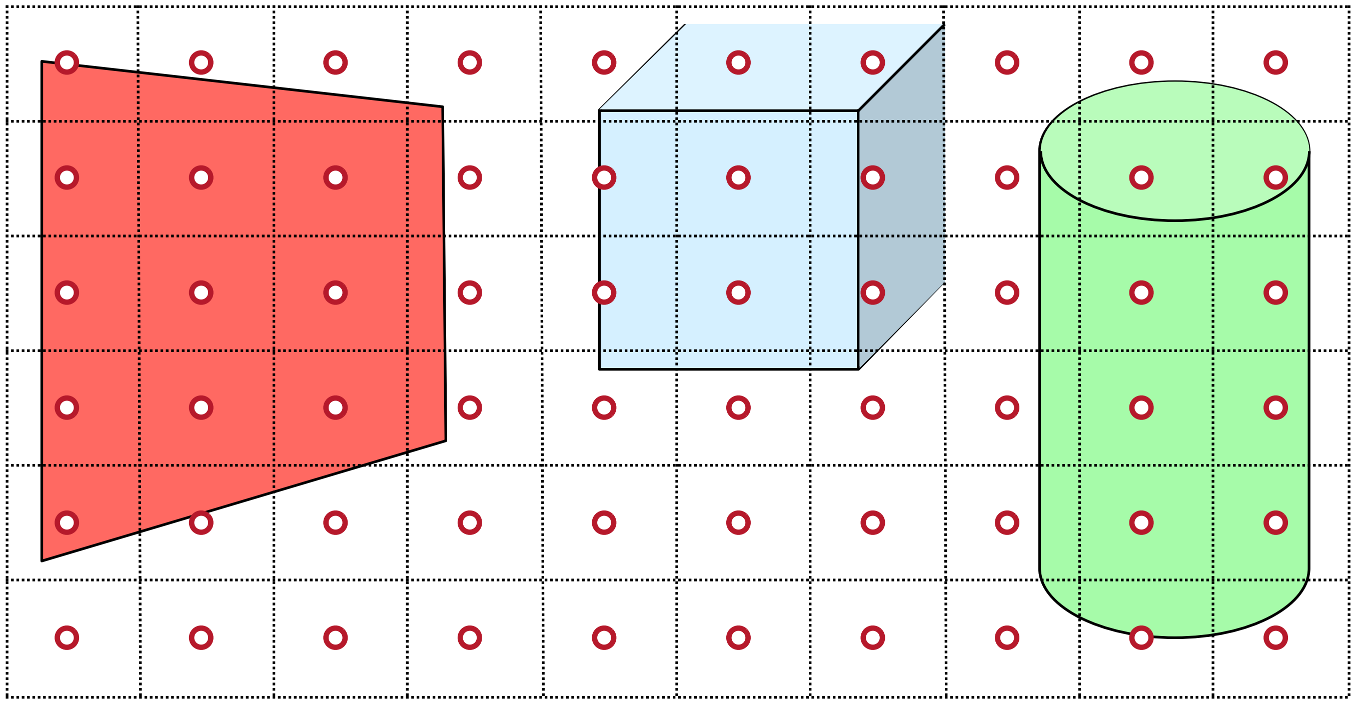
Ray Casting

- 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



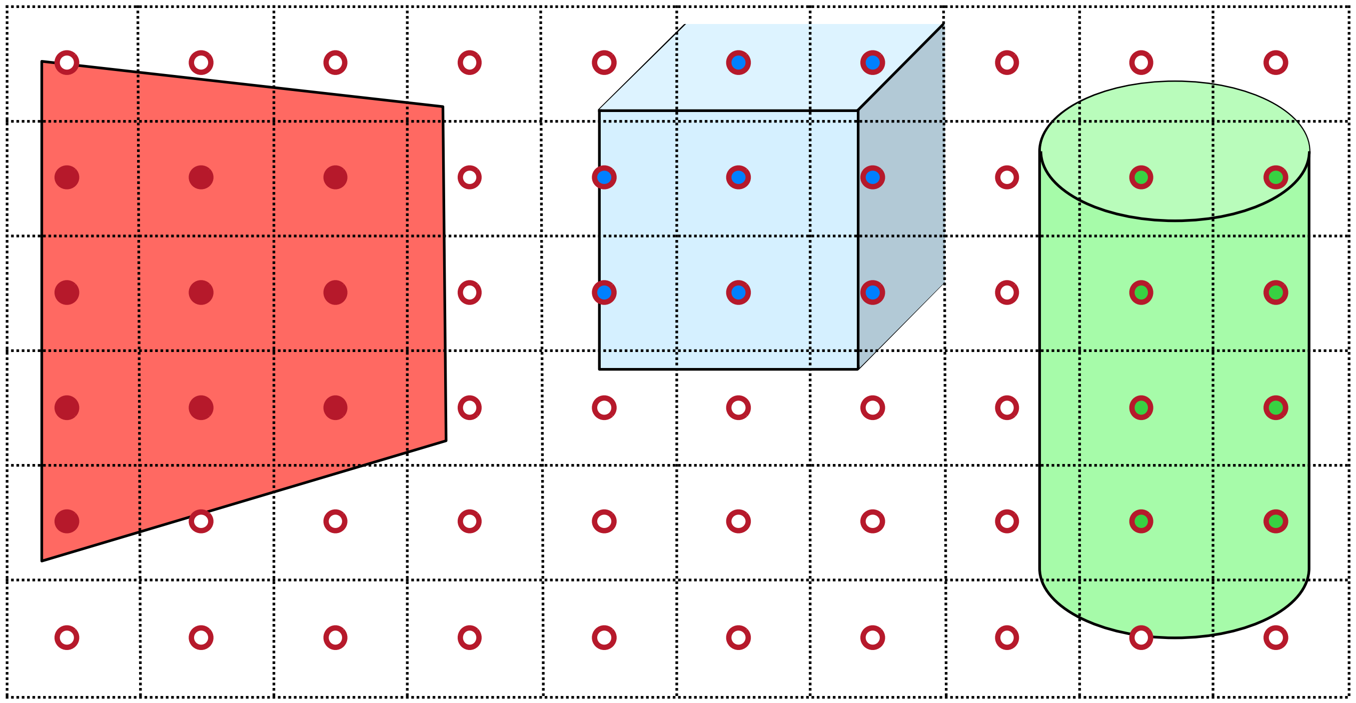
Ray Casting

- 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



Visible Surface Determination

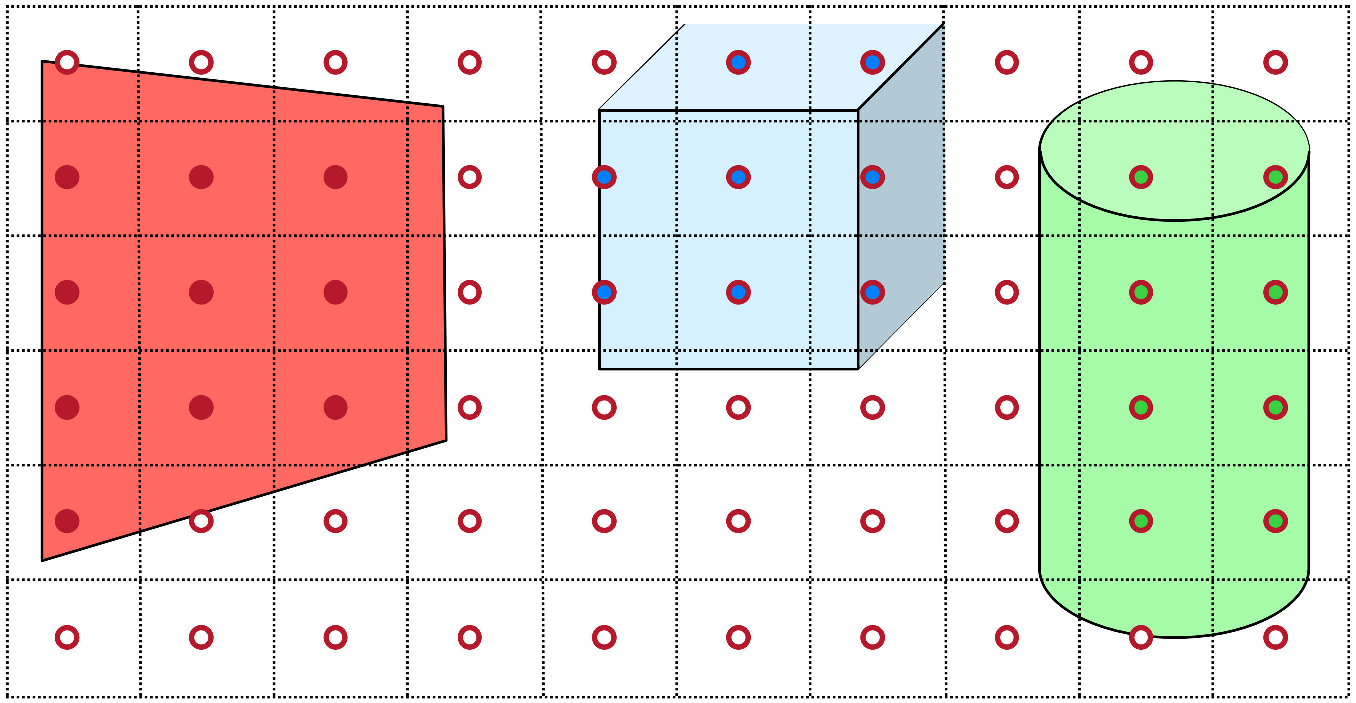
- For each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance



More efficient algorithms utilize spatial coherence!

Rendering Algorithms

Rendering is a problem in **sampling** and **reconstruction**!

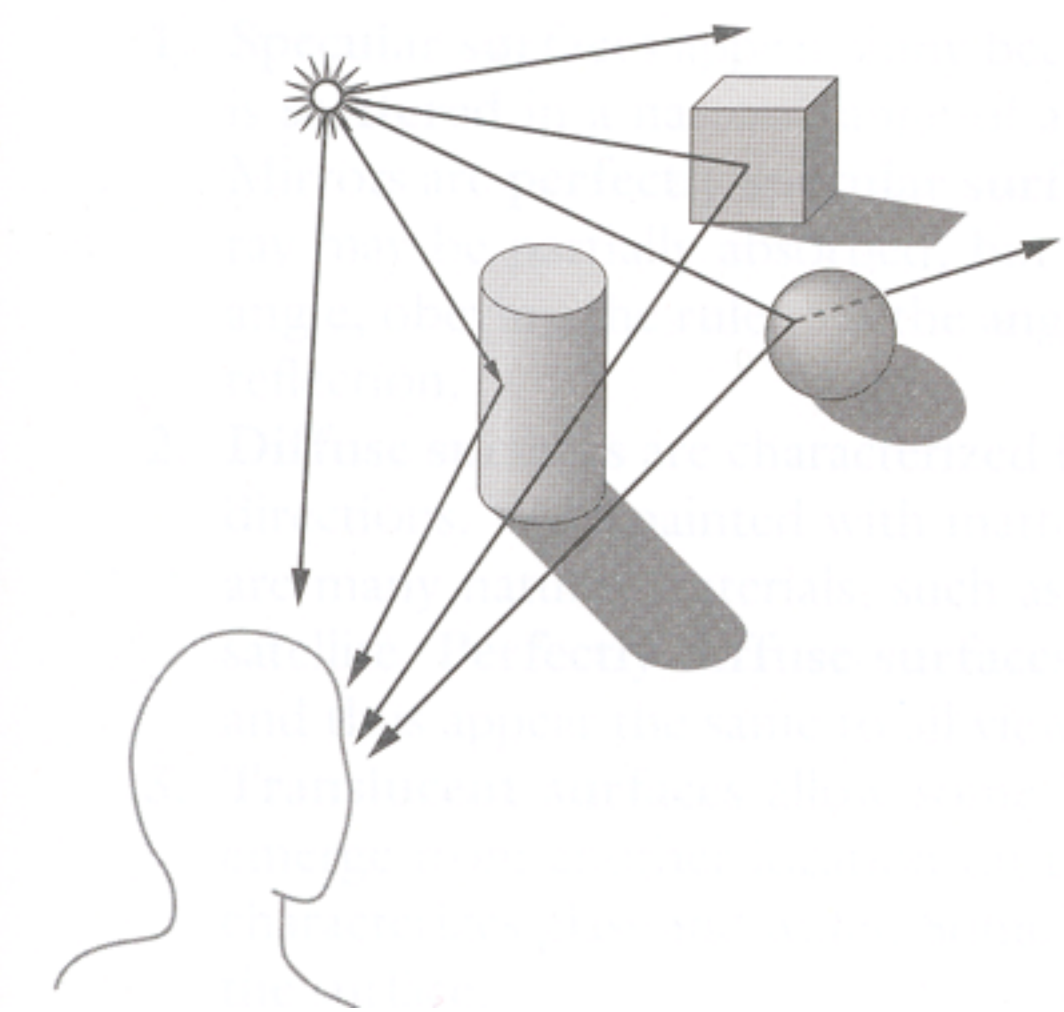


Overview

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

» **Lighting simulation**

How do we compute the radiance for each sample ray?

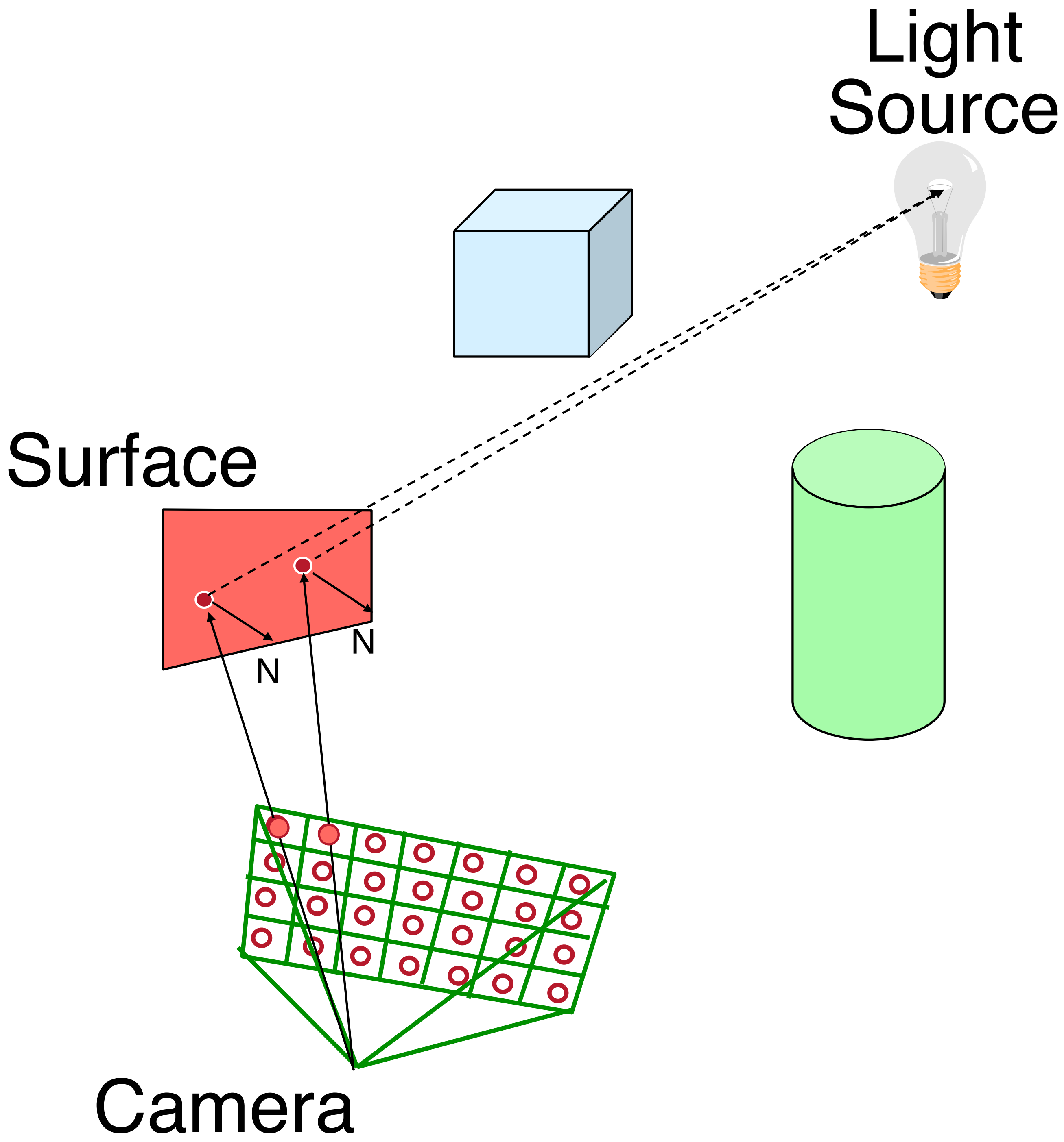


Lighting Simulation

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

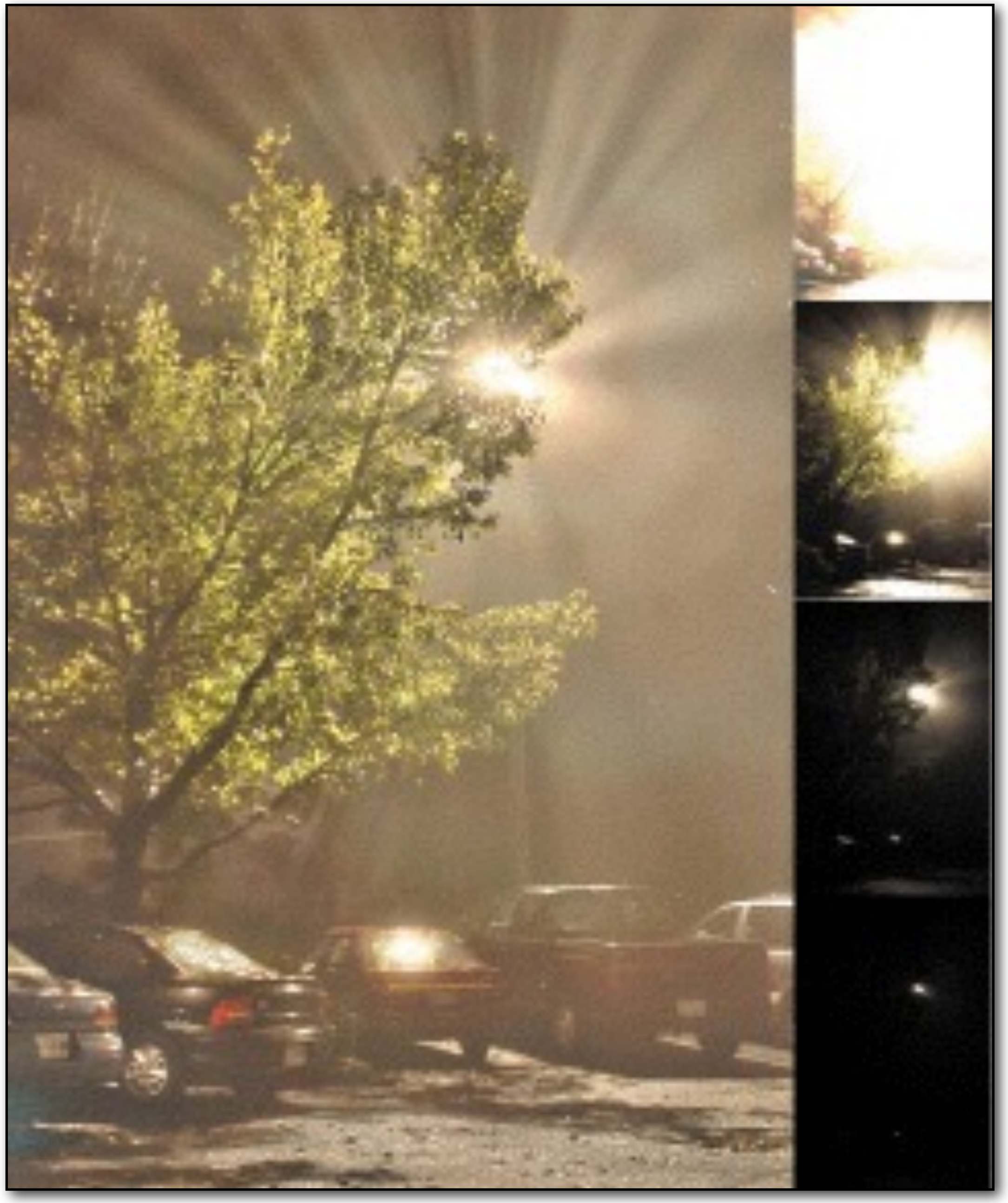
Lighting Simulation

- Lighting parameters
  - o Light source emission
  - o Surface reflectance
  - o Atmospheric attenuation



Lighting Simulation

- Lighting parameters
  - o Light source emission
  - o Surface reflectance
  - o Atmospheric attenuation

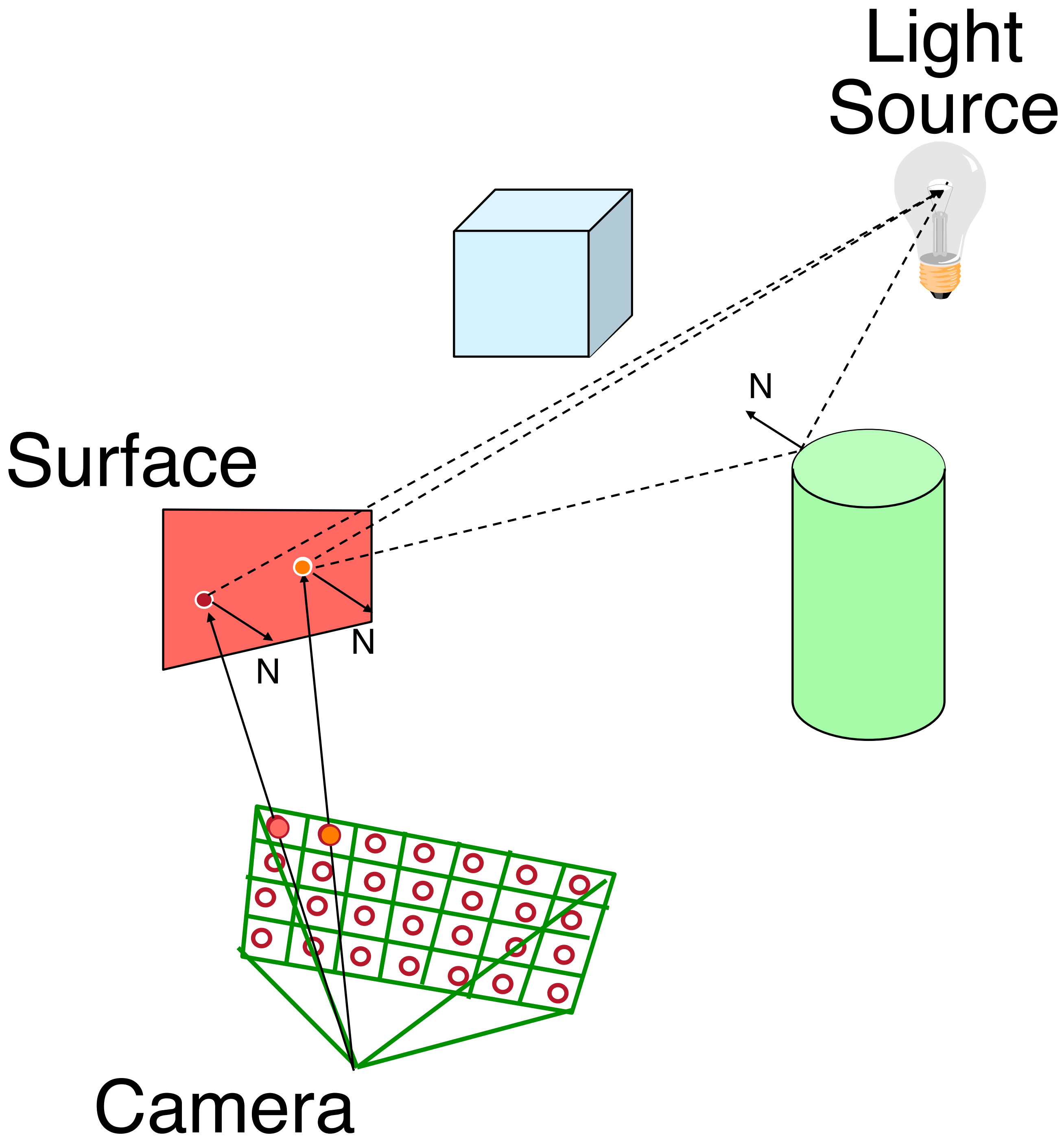


Durand & Dorsey Siggraph '02

Real-Time Volumetric Shadows paper [Chen et al. 2011]

Lighting Simulation

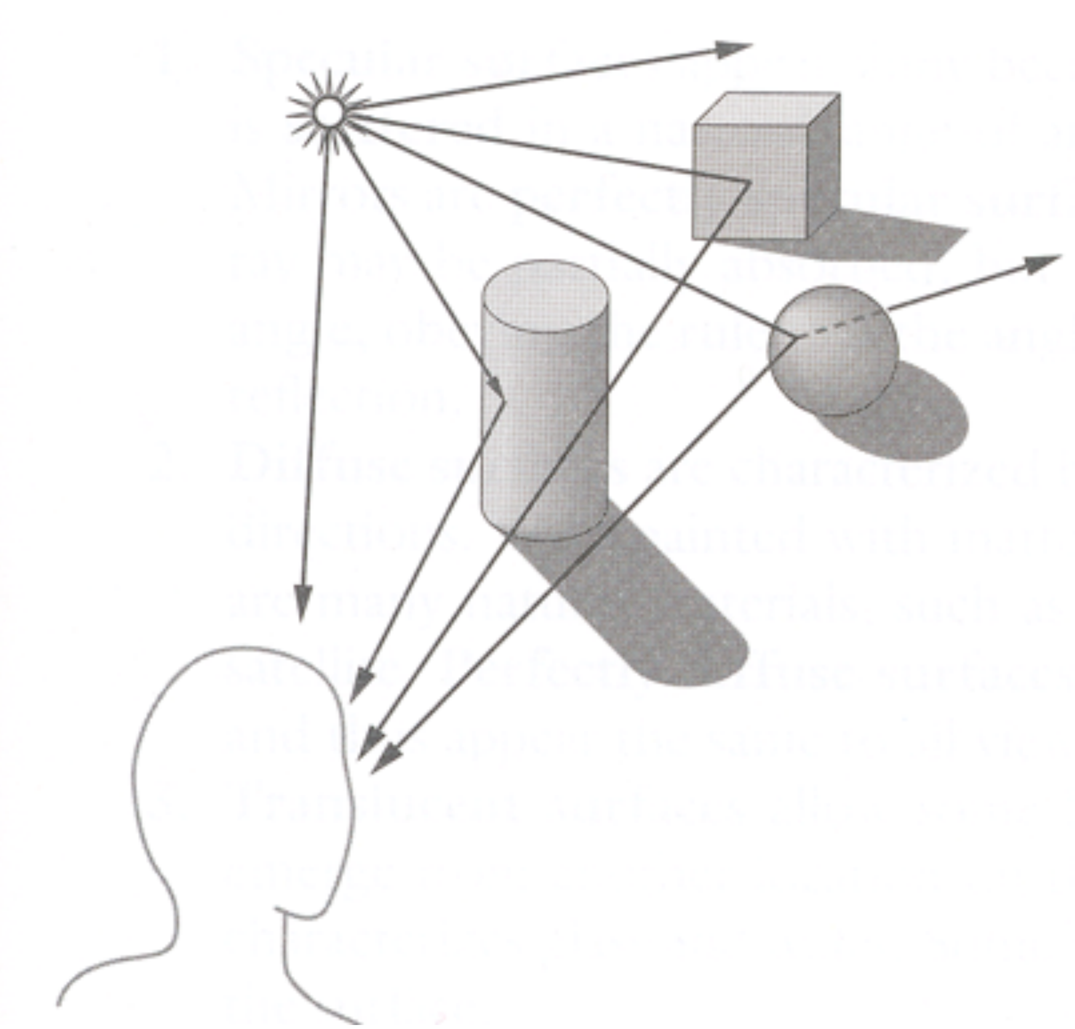
- Direct illumination
  - Ray casting
  - Polygon shading
- Global illumination
  - Ray tracing
  - Monte Carlo methods
  - Radiosity methods



More on these methods later!

Summary

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

Rendered by
Tor Olav Kristensen
using POV-Ray

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