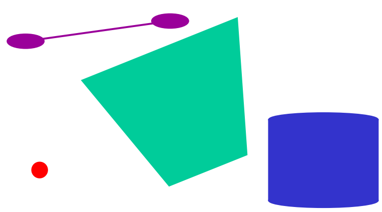
cs155 – z sweedyk

constituents of
3d scene

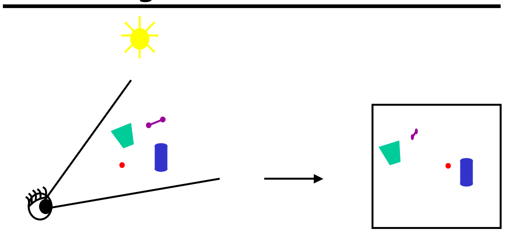
3d graphics

- Modeling
- Rendering

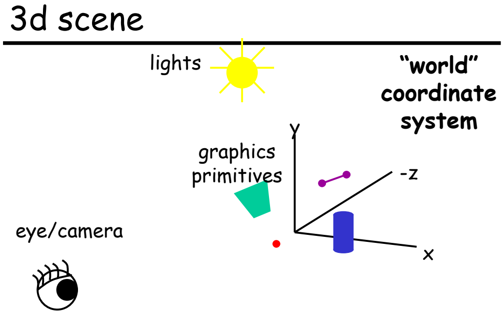
modeling



rendering



3d scene



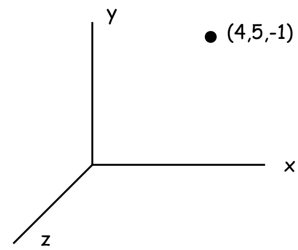
background: spaces

- linear space
  - scalar
  - vectors
- affine space
  - scalar
  - vector
  - points

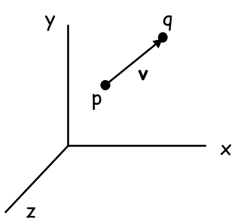
scalars: real numbers

3.8
2.7
4.1
-1000.2
5

point: position in (3d) space

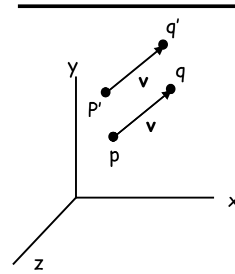


vector: distance & direction in (3d) space



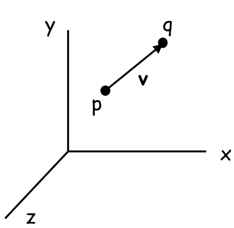
v: the way you get from p to q

vector: magnitude & direction in (3d) space



A vector does not have a position in space !

vector: distance & direction in (3d) space

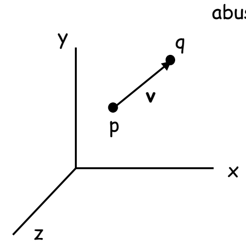


v: the way you get from p to q

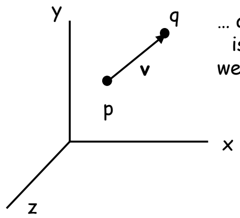
$\mathbf{v} = \langle q_x-p_x, q_y-p_y, q_z- p_z \rangle$

vector: notation

abusive but useful notation **v**=q-p



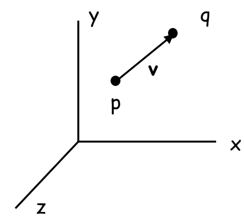
vector: distance & direction in (3d) space



abusive but useful notation **v**=q-p

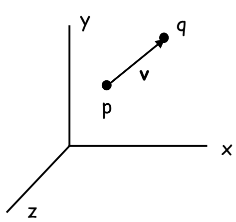
… abusive because addition of points is not defined and, even if it was, we'd expect the result to be a point, not a vector

vector operating on a point



- $p=(p_x,p_y,p_z)$
- $\mathbf{v}=\langle v_x,v_y,v_z\rangle$
- $q=(p_x+v_x, p_y+v_y, p_z+v_z)$

vector operating on a point



abusive but useful notation
q=p+**v**

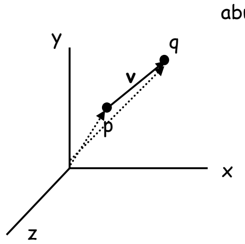
points vs. vectors

- (x,y,z) is a point in space
- <x,y,z> is the distance/direction you have to travel from the origin to get to the point (x,y,z)

sometimes it is convenient to blur the distinction

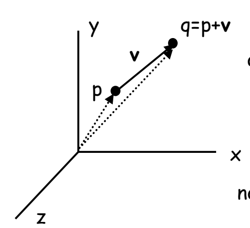
vector: distance & direction in (3d) space

abusive but useful notation **v**=q-p



this notation makes perfect sense if we think of q and p as vectors

vector operating on a point



abusive but useful notation
q=p+**v**

notation makes sense if p and q are vectors

points vs. vectors

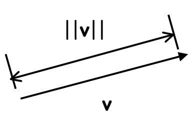
- (x,y,z) is a point in space
- <x,y,z> is the distance/direction you have to travel from the origin to get to the point (x,y,z)

sometimes it is convenient to blur the distinction … sometimes not…

vector operations

- vector norm
- scalar multiplication
- vector addition
- dot product
- cross product (for 3d)
- linear transforms

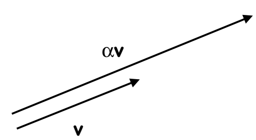
magnitude, length, euclidean norm



- $\mathbf{v}=\langle v_x,v_y,v_z\rangle$
- $||\mathbf{v}||=(v_x^2+v_y^2+v_z^2)^{\frac{1}{2}}$

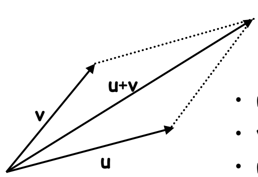
v is a unit vector if $||\mathbf{v}||=1$

scalar multiplication



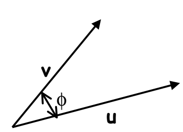
- $\mathbf{v}=\langle v_x,v_y,v_z\rangle$
- $\alpha\mathbf{v}=\langle\alpha v_x, \alpha v_y, \alpha v_z\rangle$

vector addition



- $\mathbf{u}=\langle u_x,u_y,u_z\rangle$
- $\mathbf{v}=\langle v_x,v_y,v_z\rangle$
- $\mathbf{u}+\mathbf{v}=\langle u_x+v_x, u_y+v_y, u_z+v_z\rangle$

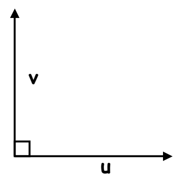
dot (inner) product



- $\mathbf{u}=\langle u_x,u_y,u_z\rangle$
- $\mathbf{v}=\langle v_x,v_y,v_z\rangle$
- $\mathbf{u}\bullet\mathbf{v}= u_xv_x+u_yv_y+u_zv_z$
  $= ||\mathbf{u}||_*||\mathbf{v}|| \cos\phi$

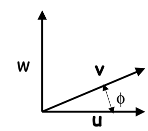
note: $\mathbf{v}\bullet\mathbf{v}=||\mathbf{v}||^2$

orthogonal vectors



- non-zero vectors **u** and **v** are orthogonal if **u•v** = 0

cross product (3d)

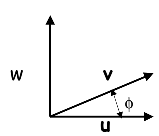


u X **v** = **w**

magnitude: ||**w**|| = ||**u**||*||**v**|| sin ϕ

direction: **w** is orthogonal to both u and v in direction defined by right hand rule

cross product (3d)



- **u**=$\langle u_x, u_y, u_z \rangle$
- **v**=$\langle v_x, v_y, v_z \rangle$

$$w = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}$$

linear transform

f: $\Re^n \to \Re^n$ is a linear transform if for any scalars α,β and any vectors **u**,**v** f satisfies the following:

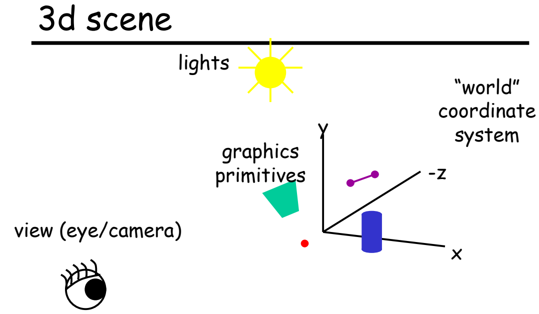
$$f(\alpha\mathbf{u}+\beta\mathbf{v}) = \alpha f(\mathbf{u})+\beta f(\mathbf{v})$$

Linear transforms are the functions that can be expressed as matrix multiplication!!!!

Linear transform

$$\begin{bmatrix} s & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} sx \\ ty \\ w \end{bmatrix}$$

3d scene



overview

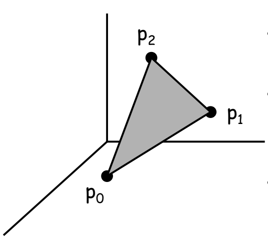
- models
  - **simple shapes** (for now)
  - transforms
- lights and material properties
- view
  - viewpoint
  - projection type/view volume
- illumination models
  - local
  - global

primitives

simple shapes
- triangle
- sphere

later in the course we'll cover modeling in greater depth

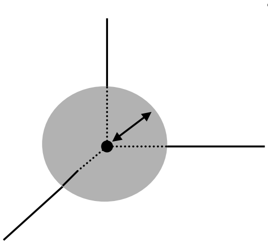
triangle



- defined by three vertices
- vertex order (counterclockwise) defines "front"
- necessarily planar

triangle mesh

sphere

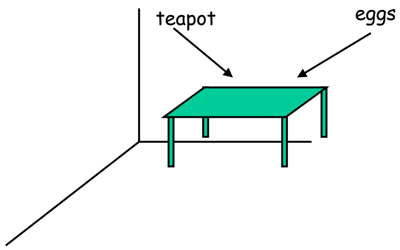


- specified by center and radius

overview

- models
  - simple primitives (for now)
  - **transforms**
- lights and material properties
- frustum
  - Projection
- illumination models
  - local
  - global

transforms

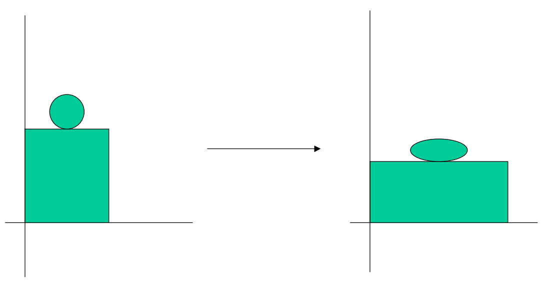


transforms

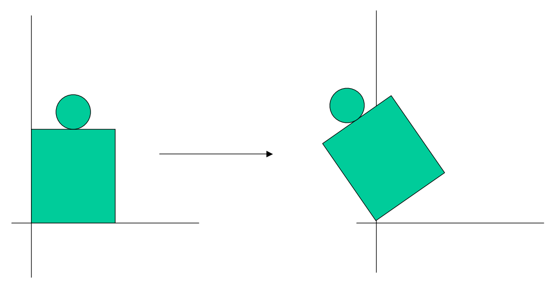
- scale
- rotate
- translate

let's start with 2D

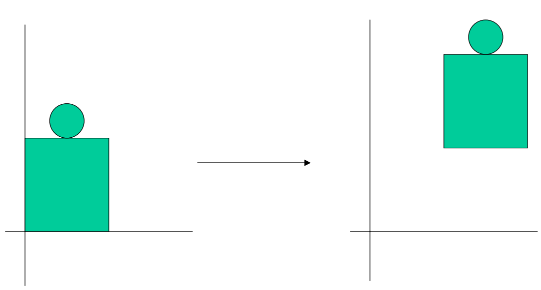
scale



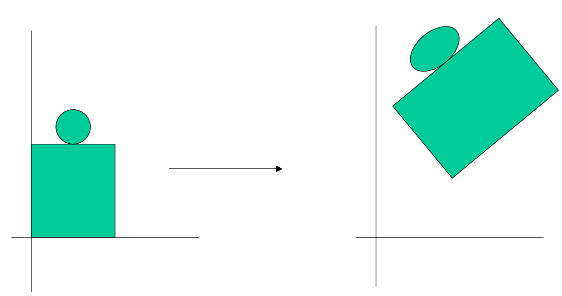
rotate



translate



composite transforms

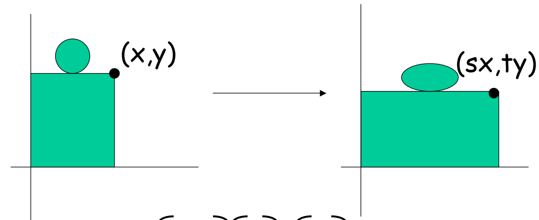


computing scaled coordinates

$$\begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ ty \end{bmatrix}$$

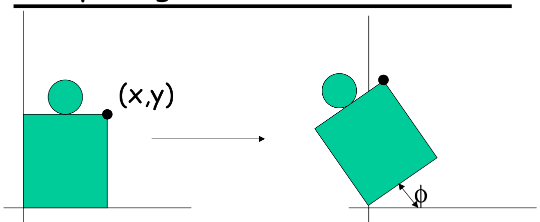
scale

again we are blurring the distinction of a point and vector



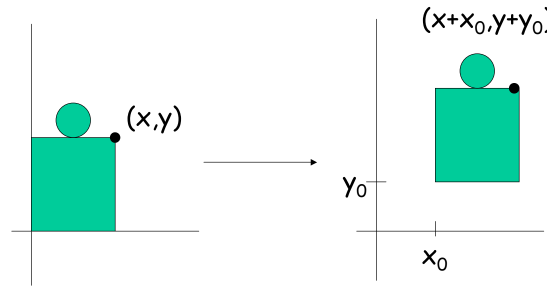
$$\begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ ty \end{bmatrix}$$

computing rotated coordinates



$$\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \end{bmatrix}$$

computing translated coordinates

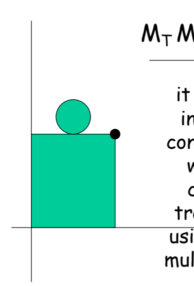


linear transformation

- $f(\mathbf{v})$ is linear if
  $f(\alpha\mathbf{u}+\beta\mathbf{v}) = \alpha f(\mathbf{u})+\beta f(\mathbf{v})$

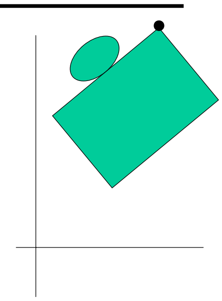
- translation is not a linear transform: define $T(v) = v+w_0$ where w_0 is a non-zero vector
  $T(u+v) = u + v + w_0$
  $T(u)+T(v) = u+v+2w_0$

transform composition

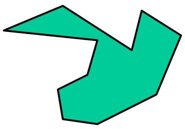


$M_T M_R M_S\, p = p'$

it would be incredibly convenient if we could compose transforms using matrix multiplication



transform polygon mesh

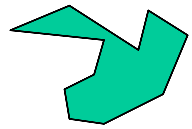


1,000,000
vertices

10 transforms

10,000,000
computations

transform polygon mesh



1,000,000
vertices

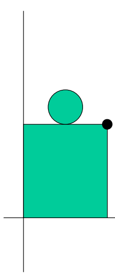
~~10 transforms~~

1 composite transform

1,000,000

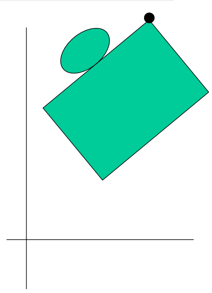
~~10,000,000~~
computations

transform composition

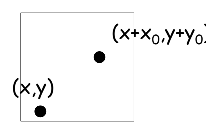


$M_T M_R M_S p = p'$

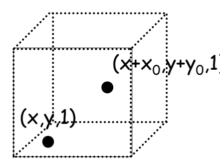
it would be incredibly convenient if we could compose transforms using matrix multiplication



a little trick …



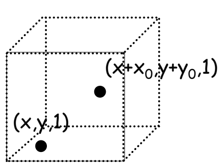
not a linear xfm



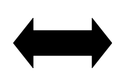
linear xfm

a little trick …

$$\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+x_0 \\ y+y_0 \\ 1 \end{bmatrix}$$



homogenous coordinates

(x,y)  (x,y,1)

scale

$$\begin{bmatrix} s & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} sx \\ ty \\ 1 \end{bmatrix}$$

rotate

$$\begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \\ 1 \end{bmatrix}$$

translate

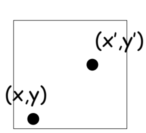
$$\begin{bmatrix} s & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} sx \\ ty \\ 1 \end{bmatrix}$$

transform form

$$\begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ \hline 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

we are not alone…

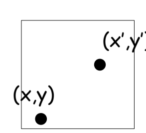
the parallel universe view of homogenous coordinates



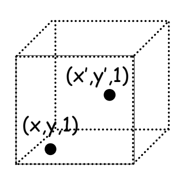
we live in this universe

we are not alone…

the parallel universe view of homogenous coordinates



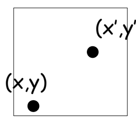
we live in this universe



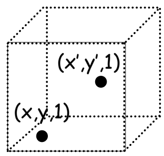
it's not the only one, but it is the only one we can experience!

and its better not to think about it …

the parallel universe view of homogenous coordinates

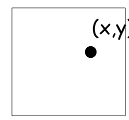


our universe has center (0,0)

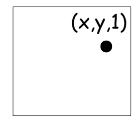


center?

2d and 2d homogenous

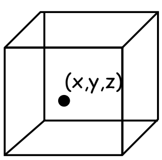


our universe

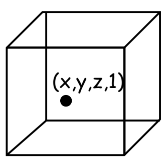


our universe when it comes to computing modeling transforms

3d and 3d homogenous



our universe



our universe when it comes to computing modeling transforms

scale

$$\begin{pmatrix} s & 0 & 0 & 0 \\ 0 & t & 0 & 0 \\ 0 & 0 & u & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} sx \\ ty \\ uz \\ 1 \end{pmatrix}$$

rotate about z axis

$$\begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \\ z \\ 1 \end{pmatrix}$$

rotate about x & y axes are similar

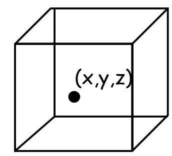
translate

$$\begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+x_0 \\ y+y_0 \\ z+z_0 \\ 1 \end{pmatrix}$$

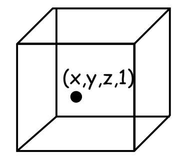
transform form

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z \\ 1 \end{bmatrix}$$

3d and 3d homogenous



our universe

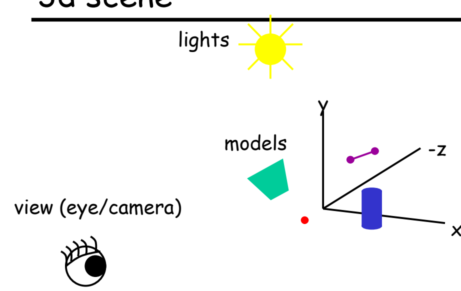


our universe when it comes to computing modeling transforms

overview

- models
  - simple primitives (for now)
  - transforms
- **lights and material properties**
- camera
  - frustum
  - Projection
- illumination models
  - local
  - global

3d scene



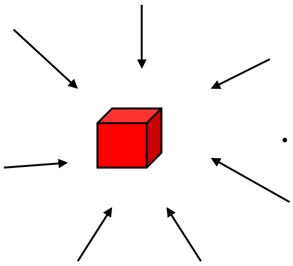
objective

approximate the effects of light/materials as we perceive them in a computationally efficient way

light sources (in cs155)

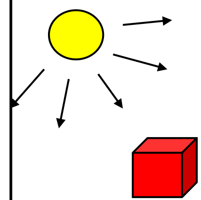
- ambient light
- point light
- spot light
- directional light

ambient light



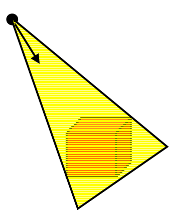
- "source-less" light that arrives uniformly from all directions at all points in the scene
- specification
  - red, green, and blue intensity

point light



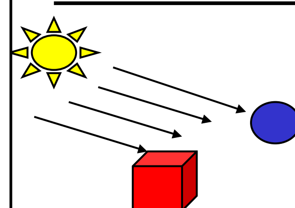
- light emanates uniformly in all directions
- specification
  - location
  - red, green, and blue intensity
  - how the light drops off with distance

spot light



- light emanates in a cone
- specifications
  - location in world coordinates
  - red, green, and blue intensity
  - how the light drops off with distance
  - center axis how light drops off with angle from center

directional light

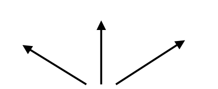


- light positioned at "infinity"; intensity and incident angle are constant for all points in scene
- specification
  - direction
  - red, green, and blue intensity

material properties

how does the surface material
- emit light
- reflect light
- transmit light

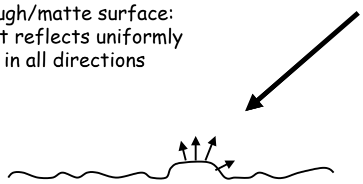
surface emitter



- Light emanates from (front/outward) surface of object in scene

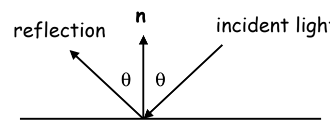
diffuse reflection

rough/matte surface:
light reflects uniformly
in all directions



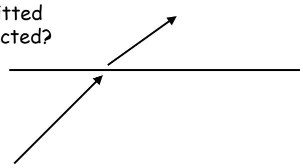
specular reflection

smooth (mirror) surfaces: light
reflects in one (primary) direction



transparency

can light be
transmitted
through surface?
is transmitted
light refracted?



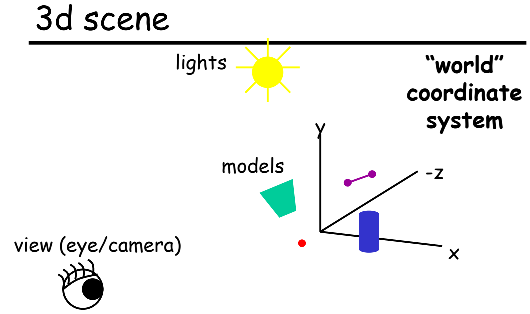
material property specification

1. red, green, and blue emission
2. ambient reflectivity for red, green, and blue light
3. diffuse reflectivity for red, green, and blue light
4. specular reflectivity for red, green, and blue light
5. transparency and refractive index
6. cheap tricks

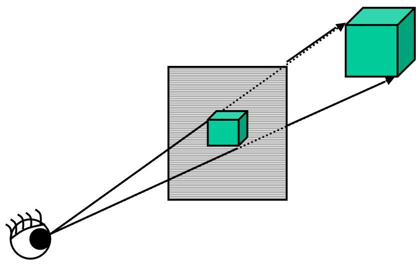
overview

- models
  - **simple shapes** (for now)
  - transforms
- lights and material properties
- **camera**
  - frustum
  - projection
- illumination models
  - local
  - global

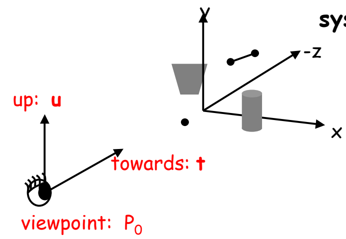
3d scene



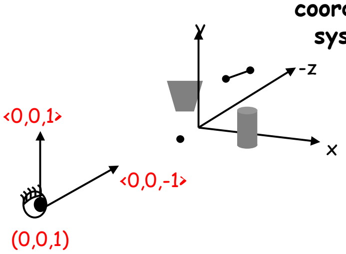
pinhole camera model



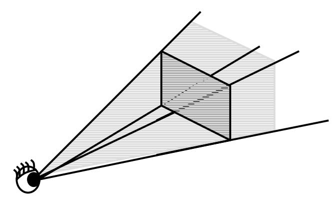
how is eye situated?



standard configuration

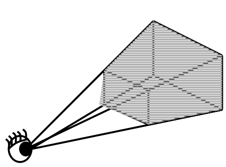


how much of the world is seen?



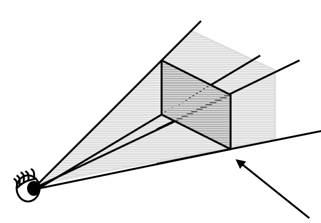
a pyramid

how much of the world is seen?



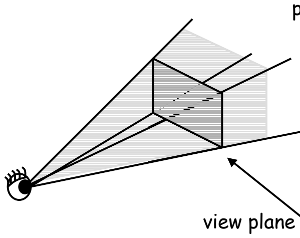
a pyramid, sometimes truncated in front and/or back, called the frustum

projection



everything in the frustum is projected onto a view plane

view plane



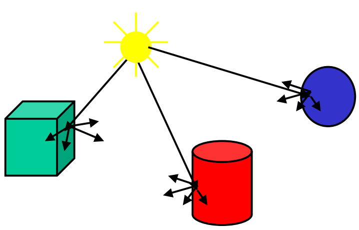
almost always the view plane is:

- orthogonal to the toward vector **t**
- some specified distance *d* from the viewpoint P_0

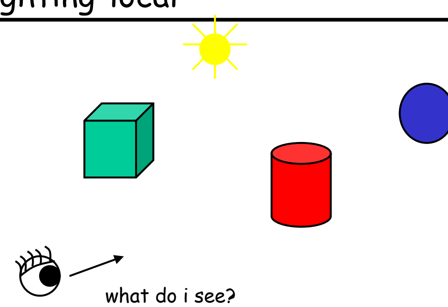
overview

- models
  - simple shapes (for now)
  - transforms
- lights and material properties
- camera
  - frustum
  - Projection
- **illumination models**
  - local
  - global

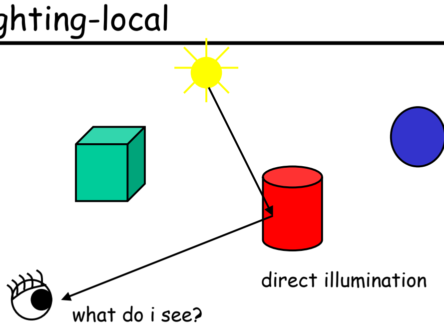
lighting-global



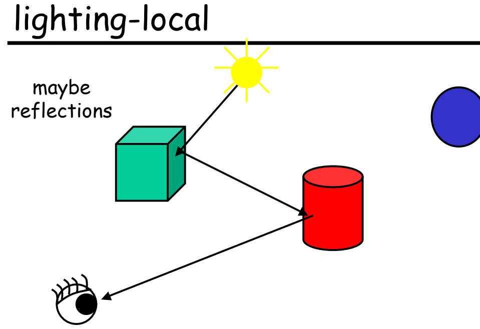
lighting-local



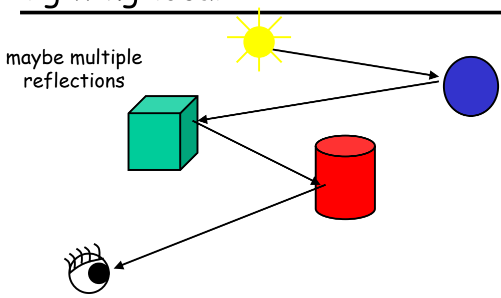
lighting-local



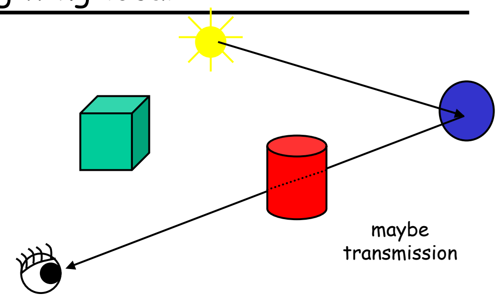
lighting-local



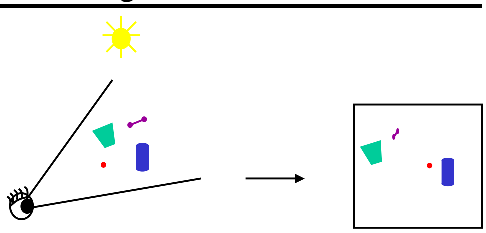
lighting-local



lighting-local



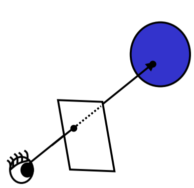
rendering



rendering

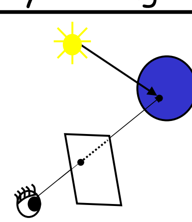
- image based algorithms
  - ray casting/tracing
- object-based algorithms
  - vertex pipeline

ray casting



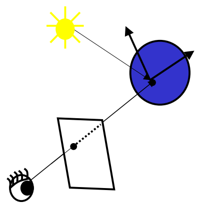
- cast ray through pixel into scene
- find closest intersection (if any)
- compute luminance at intersection

ray casting



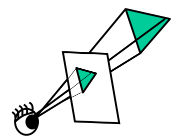
- cast ray through pixel into scene
- find closest intersection (if any)
- compute luminance at intersection
  - direct illumination

ray tracing



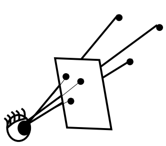
- cast ray through pixel into scene
- find closest intersection (if any)
- compute luminance at intersection
  - direct illumination
  - reflections
  - transmission

object-based



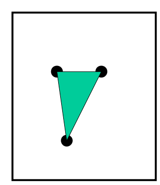
- project each object
- hidden surface removal

vertex pipeline



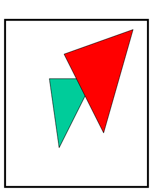
- project vertices each polygon

vertex pipeline



- project vertices of each polygon
- turn on “inside” pixels

vertex pipeline



- project vertices of each polygon
- turn on inside pixels
- use hidden surface removal to resolve conflicts