

Computer Graphics

Computer Science & Information Technology
Yung-Yu Chuang

Introduction



- Instructor: Yung-Yu Chuang (莊永裕)
- E-mail: cyy@csie.ntu.edu.tw
- Office: CSIE 527
- Grading: a MatchMove project

What is computer graphics ?



- Definition
 - the pictorial *synthesis* of real or imaginary objects from their computer-based models

		OUTPUT	
		descriptions	images
INPUT	descriptions	Computer Graphics	
	images	Computer Vision	Image Processing

Computer graphics



- Create a 2D image/animation of a 3D world

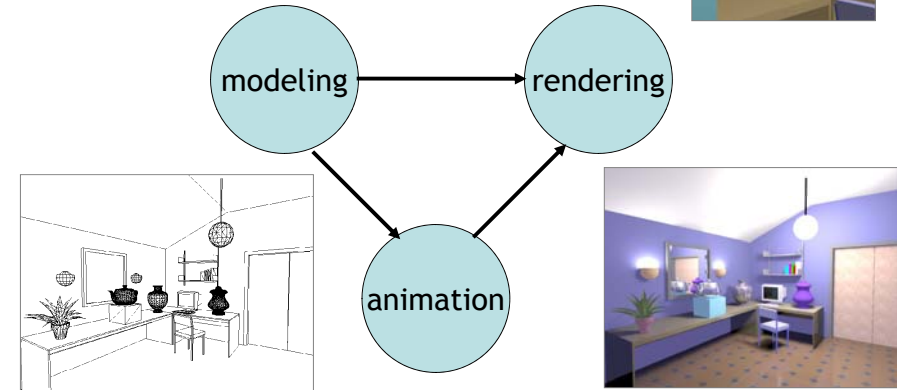
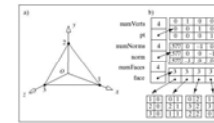


Applications

- Movies
- Interactive entertainment
- Industrial design
- Architecture
- Culture heritage

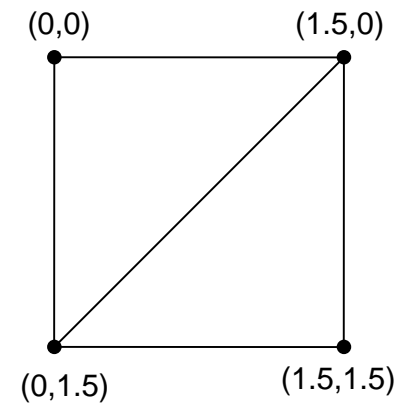


Computer graphics



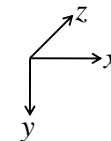
Modeling

A simple example



vertices
 0.0, 0.0, 0.0
 1.5, 0.0, 0.0
 0.0, 1.5, 0.0
 1.5, 1.5, 0.0

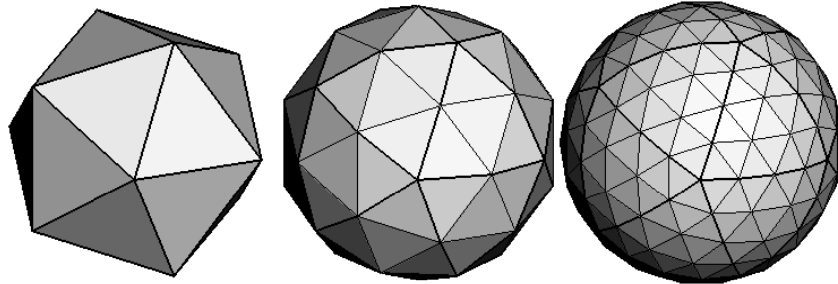
triangles
 0, 2, 1
 1, 2, 3



The power of triangles



- Every thing can be represented by triangles to a degree of precision.

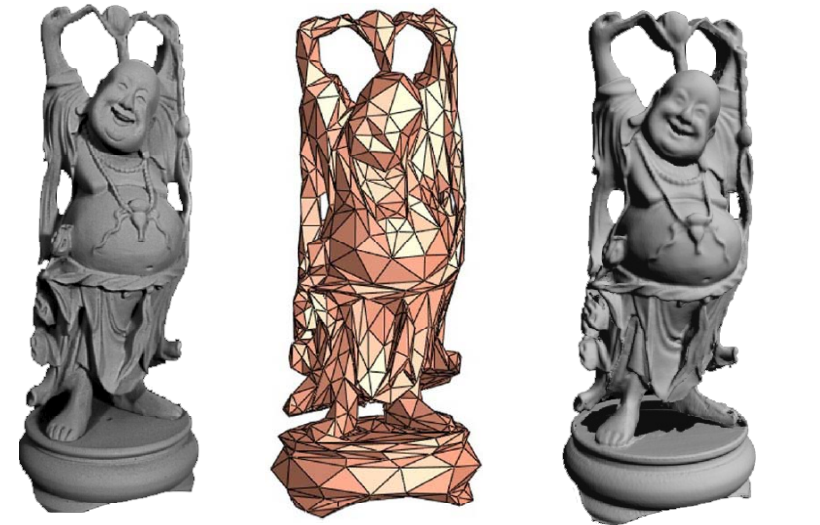


20 triangles

80 triangles

320 triangles

More complex examples



a real buddha

4K mesh

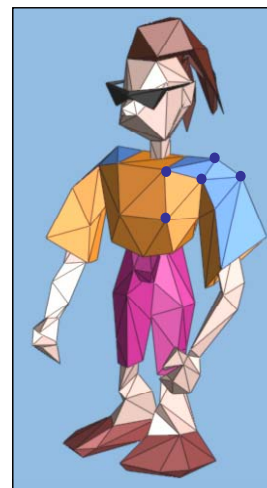
rendered 2.4M mesh

Modeling



- The position of the model can be acquired by 3D scanner or made by artists using modeling tools.
- There are other ways for representing geometric objects, but triangles have many advantages.

Triangle meshes



Copyright © 1998, Microsoft

$\{f_1\} : \{v_1, v_2, v_3\}$

$\{f_2\} : \{v_3, v_2, v_4\}$

...

$\{v_1\} : (x, y, z)$

$\{v_2\} : (x, y, z)$

...

$\{f_1\} : \text{"skin material"}$

$\{f_2\} : \text{"brown hair"}$

...

$\{v_2, f_1\} : (n_x, n_y, n_z) (u, v)$

$\{v_2, f_2\} : (n_x, n_y, n_z) (u, v)$

...

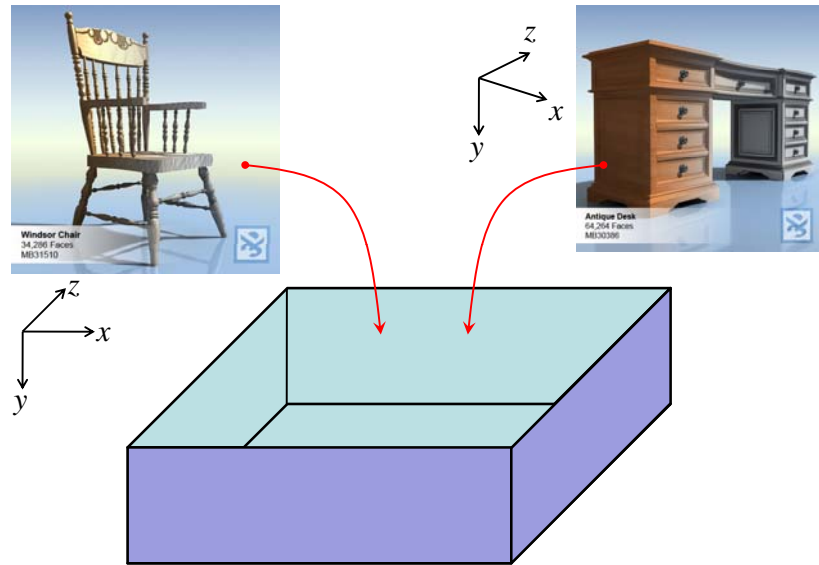
connectivity

geometry

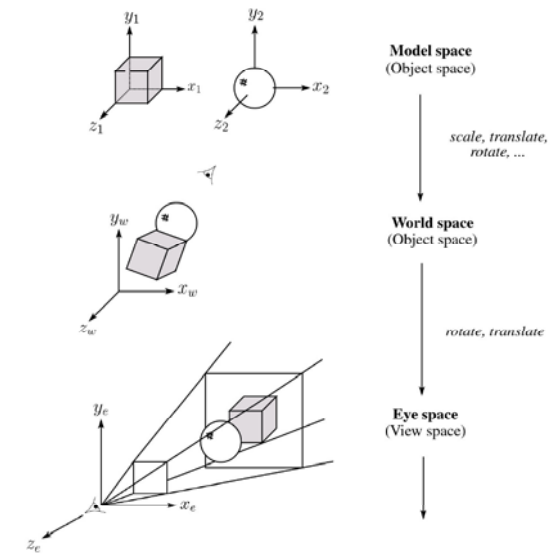
face attributes

corner attributes

Composition of a scene



Graphics pipeline



Transformations

Representation

We can represent a **point**, $\mathbf{p} = (x, y)$ in the plane

- as a column vector $\begin{bmatrix} x \\ y \end{bmatrix}$
- as a row vector $\begin{bmatrix} x & y \end{bmatrix}$

Representation



We can represent a **2-D transformation** M by a matrix

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

If \mathbf{p} is a column vector, M goes on the left:

$$\mathbf{p}' = M\mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D transformations



Here's all you get with a 2 x 2 transformation matrix M :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

So:

$$x' = ax + by$$

$$y' = cx + dy$$

We will develop some intimacy with the elements $a, b, c, d \dots$

Identity



Suppose we choose $a=d=1, b=c=0$:

- Gives the **identity** matrix:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Doesn't move the points at all

Scaling



Suppose we set $b=c=0$, but let a and d take on any *positive* value:

- Gives a **scaling** matrix:

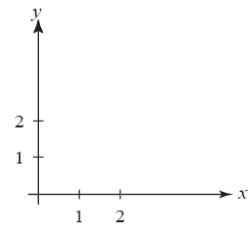
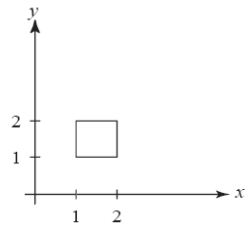
$$\begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$$

- Provides **differential scaling** in x and y :

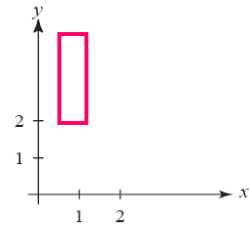
$$x' = ax$$

$$y' = dy$$

Scaling



$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$



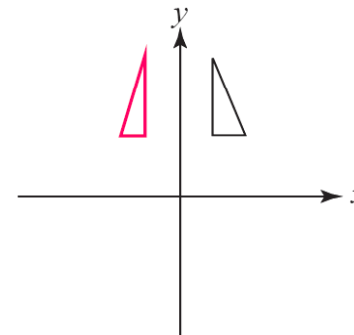
$$\begin{bmatrix} 1/2 & 0 \\ 0 & 2 \end{bmatrix}$$

Reflection

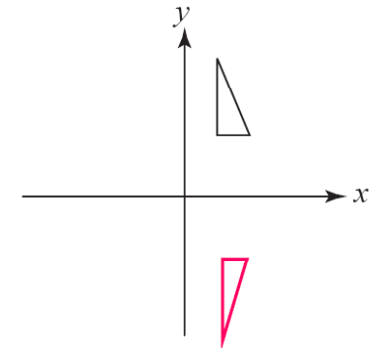


Examples:

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

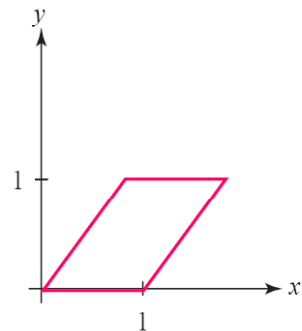
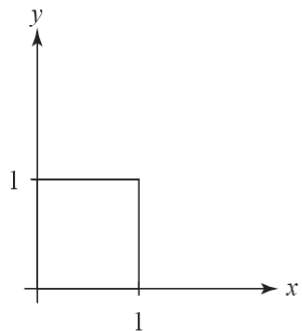


Shearing



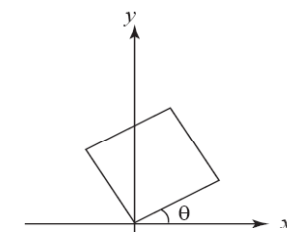
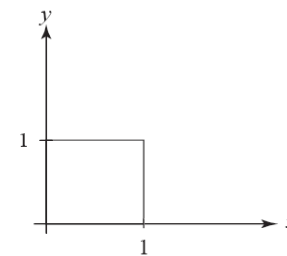
The matrix $\begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix}$

gives:
 $x' = x + by$
 $y' = y$



$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Rotation



$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix}$$

$$M = R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Limitations of a 2X2 matrix



- Scaling
- Rotation
- Reflection
- Shearing
- What do we miss?

Homogeneous coordinate



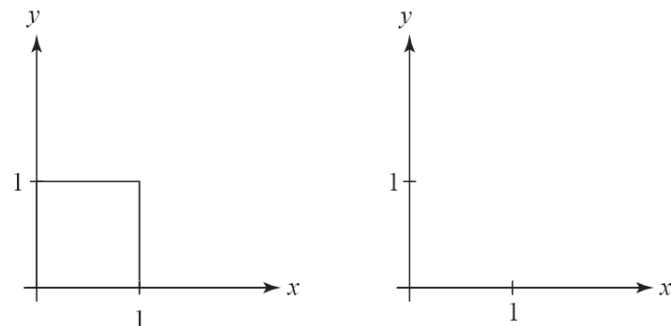
Idea is to loft the problem up into 3-space, adding a third component to every point:

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

And then transform with a 3 x 3 matrix:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = T(\mathbf{t}) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation



$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

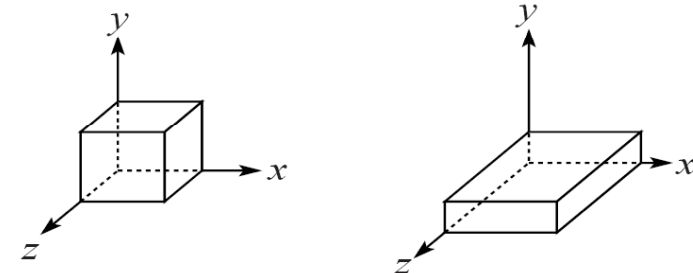
3D scaling



Some of the 3-D transformations are just like the 2-D ones.

For example, scaling:

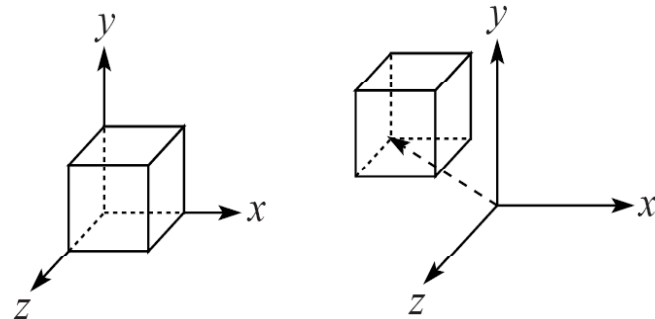
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



3D translation



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



3D rotation

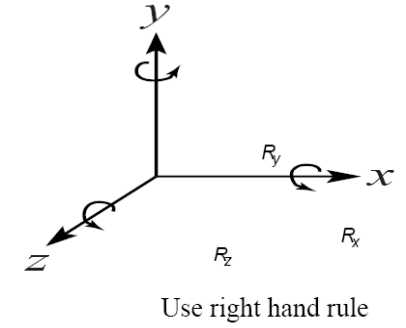


Rotation now has more possibilities in 3D:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

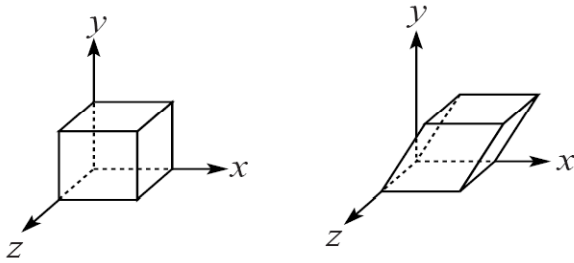


3D shearing

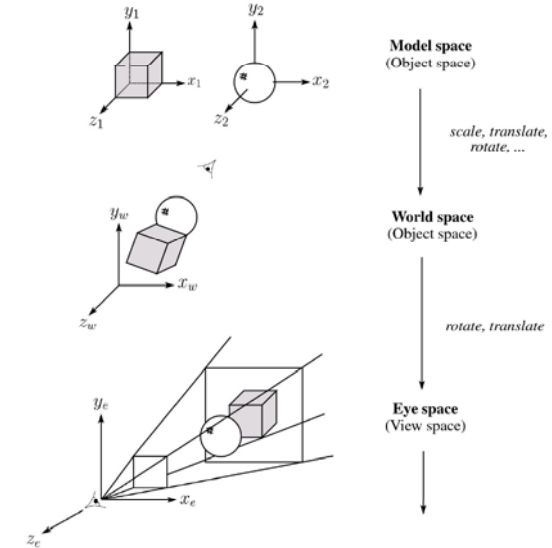


Shearing is also more complicated. Here is one example:

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

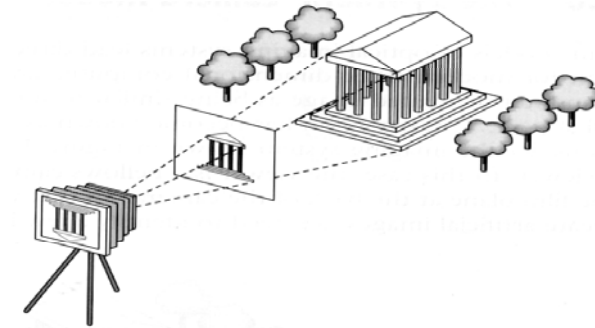


Graphics pipeline



Projections

Imaging with the synthetic camera

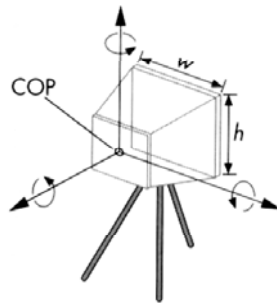


The image is rendered onto an **image plane** or **projection plane** (usually in front of the camera).

Projectors emanate from the **center of projection** (COP) at the center of the lens (or pinhole).

The image of an object point P is at the intersection of the projector through P and the image plane.

Specifying a viewer



Camera specification requires four kinds of parameters:

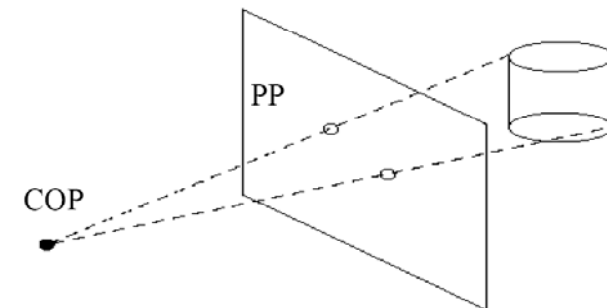
- ♦ *Position*: the COP.
- ♦ *Orientation*: rotations about axes with origin at the COP.
- ♦ *Focal length*: determines the size of the image on the film plane, or the **field of view**.
- ♦ *Film plane*: its width and height, and possibly orientation.

Projections



Projections transform points in n -space to m -space, where $m < n$.

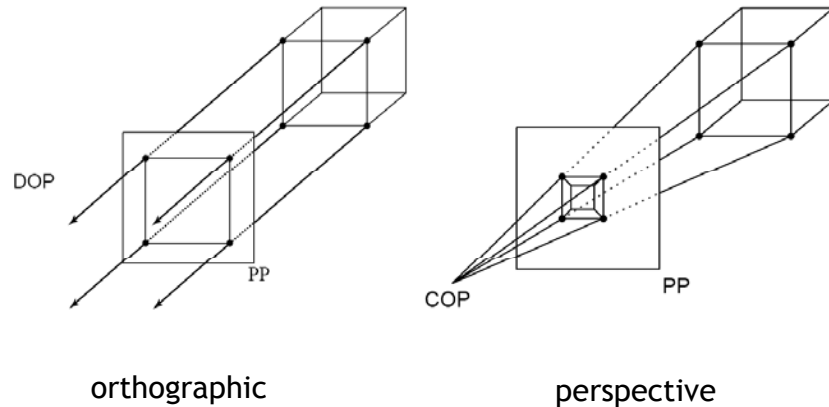
In 3D, we map points from 3-space to the **projection plane** (PP) along **projectors** emanating from the **center of projection** (COP).



There are two basic types of projections:

- ♦ **Perspective** - distance from COP to PP finite
- ♦ **Parallel** - distance from COP to PP infinite

Parallel and perspective projections



Orthographic transformation



For parallel projections, we specify a **direction of projection** (DOP) instead of a COP.

We can write orthographic projection onto the $z=0$ plane with a simple matrix.

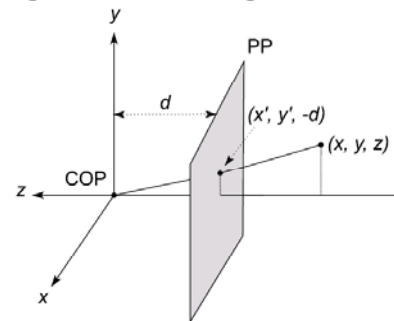
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Normally, we do not drop the z value right away. Why not?

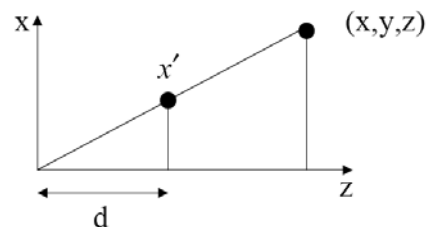
Perspective projection



Q: How do we perform the perspective projection from eye space into screen space?



Using similar triangles gives:



Perspective transform



We can write this transformation in matrix form:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = MP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Perspective divide:

$$\begin{bmatrix} X/W \\ Y/W \\ Z/W \\ W/W \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ \frac{z}{z/d} \\ \frac{d}{1} \end{bmatrix}$$

Graphics pipeline review

Triangle meshes



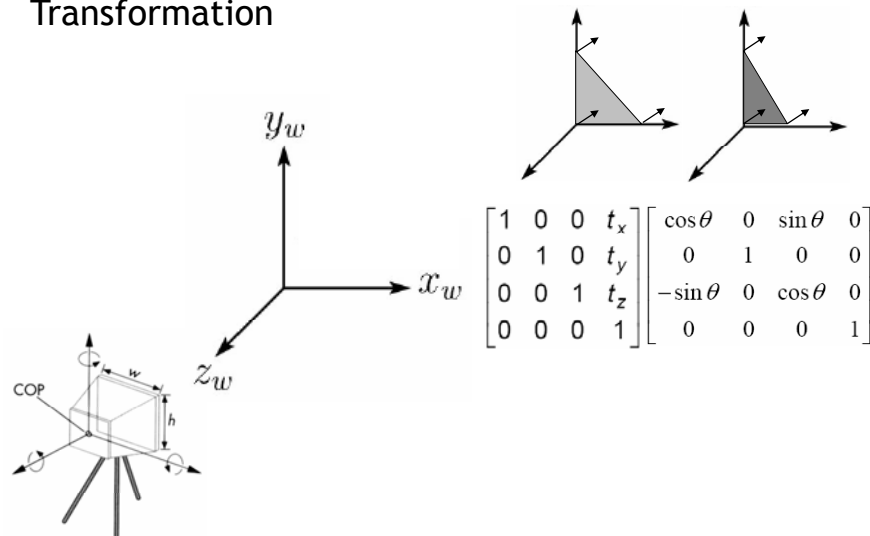
Copyright © 1998, Microsoft

- $\{f_1\} : \{v_1, v_2, v_3\}$ connectivity
- $\{f_2\} : \{v_3, v_2, v_4\}$
- ...
- $\{v_1\} : (x, y, z)$ geometry
- $\{v_2\} : (x, y, z)$
- ...
- $\{f_1\} : \text{"skin material"}$ face attributes
- $\{f_2\} : \text{"brown hair"}$
- ...
- $\{v_2, f_1\} : (n_x, n_y, n_z) (u, v)$ corner attributes
- $\{v_2, f_2\} : (n_x, n_y, n_z) (u, v)$
- ...

Review of graphics pipeline



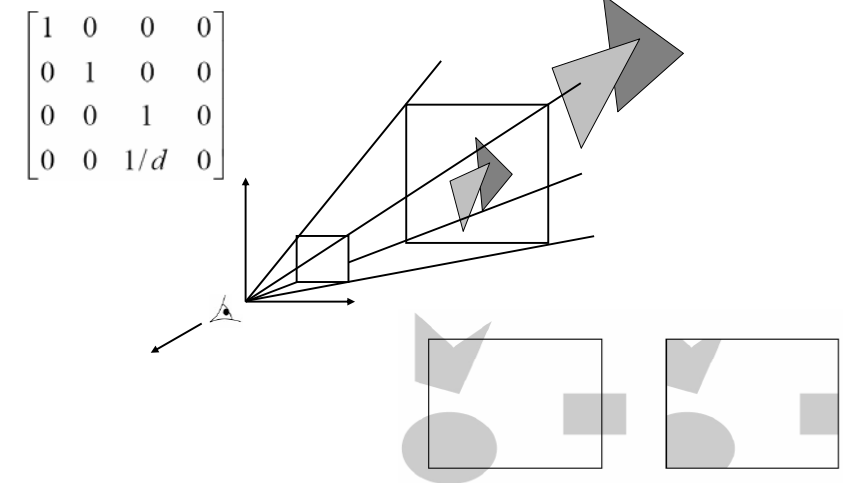
Transformation



Review of graphics pipeline



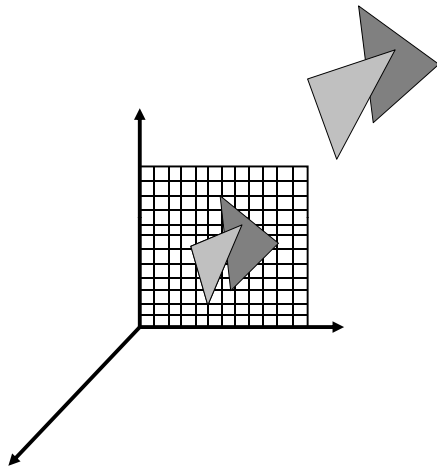
Projection & clipping



Review of graphics pipeline



- Rasterization
- Visibility

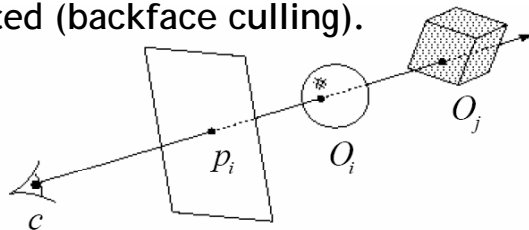


Visibility (Hidden surface removal)

Hidden surface removal



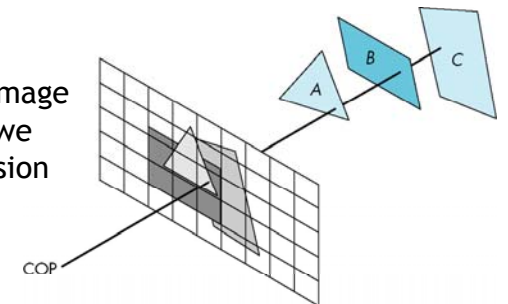
- Determining what to render at each pixel.
- A point is visible if there exists a direct line-of-sight to it, unobstructed by another other objects (visible surface determination).
- Moreover, some objects may be invisible because there are behind the camera, outside of the field-of-view, too far away (clipping) or back faced (backface culling).



Hidden surfaces: why care?



- Occlusion: Closer (opaque) objects along same viewing ray obscure more distant ones
- Reasons for removal
 - Efficiency: As with clipping, avoid wasting work on invisible objects
 - Correctness: The image will look wrong if we don't model occlusion properly



Hidden surface removal algorithms

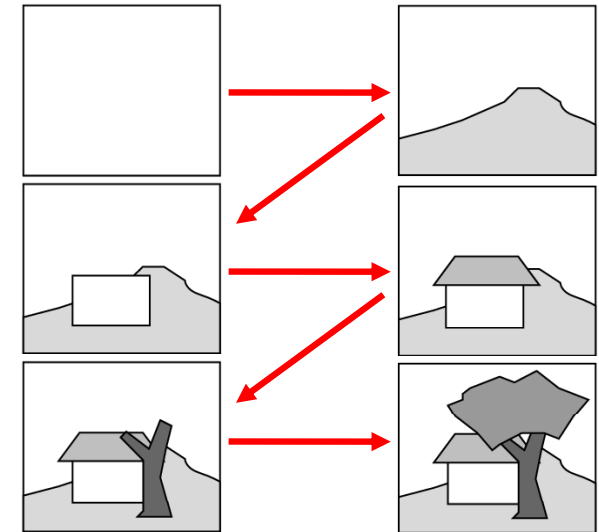


- Painter's algorithm
- Binary space partitioning
- Z-buffer
- Ray casting
- And many others

Painter's algorithm



Draw primitives
from back to
front to avoid
need for depth
comparisons

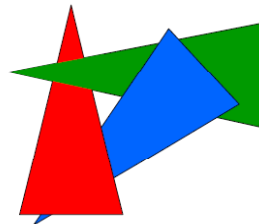


Painter's algorithm



- Idea: Sort primitives by minimum depth, then rasterize from furthest to nearest
- When there are depth overlaps, do more tests of bounding areas, etc. to see one actually occludes the other

- Cyclical overlaps are a problem



Z-buffer algorithm



- Resolve depths at the pixel level
- Idea: add Z to frame buffer, when a pixel is drawn, check whether it is closer than what's already in the framebuffer
- Proposed by Ed Catmull in 1975, widely used today, especially in hardware.

- Z-buffer, texture, subdivision surface, RenderMan
- Co-founder of Pixar
- 3 Oscars (1993, 1996, 2001), SIGGRAPH Steven Coons Award (1993)

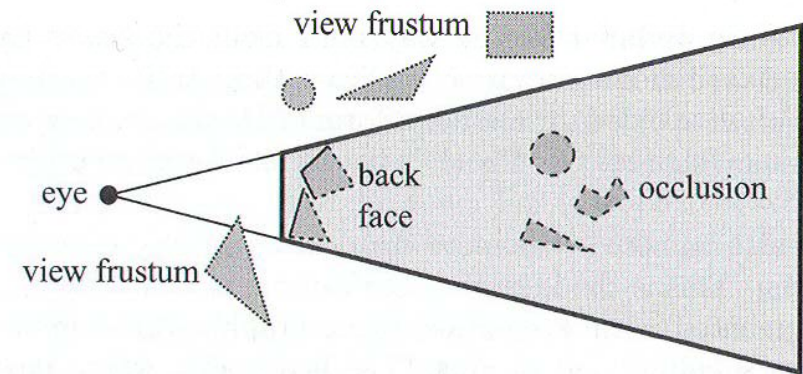


Z-Buffer



- Benefits
 - Easy to implement
 - Works for any geometric primitive
 - Parallel operation in hardware (independent of order of polygon drawn)
- Limitations
 - Memory required for depth buffer
 - Quantization and aliasing artifacts
 - Overfill
 - Transparency does not work well

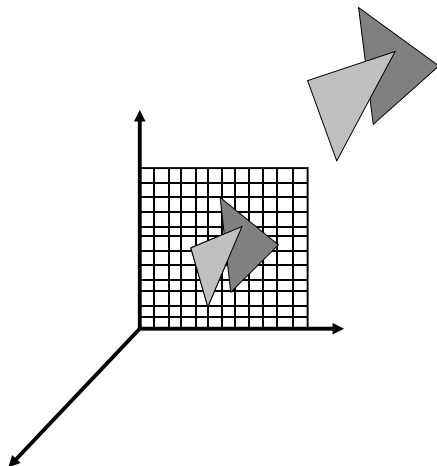
Clipping (view frustum culling)



Review of graphics pipeline



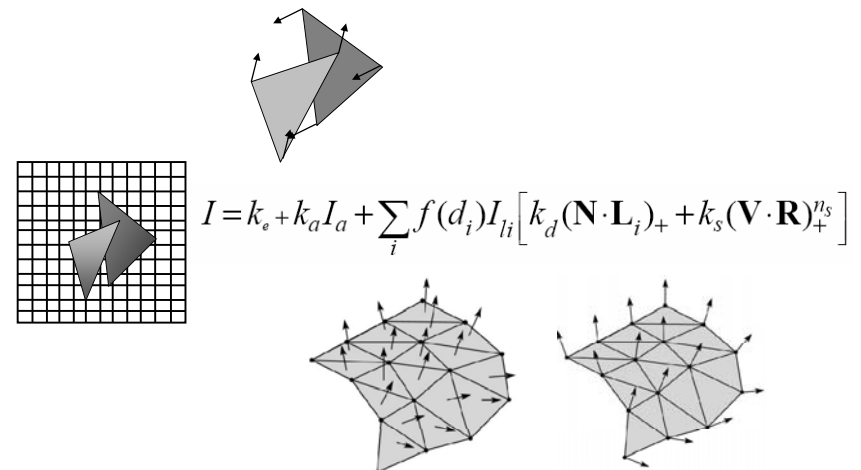
- Rasterization
- Visibility



Review of graphics pipeline



- Shading



Shading

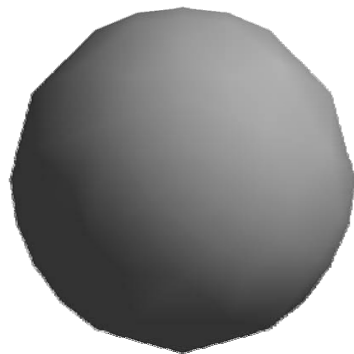
Z-buffer algorithm



```
for each pixel  $p_i$ 
{
    Z-buffer[  $p_i$  ] = FAR
    Fb[  $p_i$  ] = BACKGROUND_COLOR
}

for each polygon P
{
    for each pixel  $p_i$  in the projection of P
    {
        Compute depth  $z$  and shade  $s$  of P at  $p_i$ 
        if  $z < \text{Z-buffer}[ p_i ]$ 
        {
            Z-buffer[  $p_i$  ] =  $z$ 
            Fb[  $p_i$  ] =  $s$ 
        }
    }
}
```

What is normal?



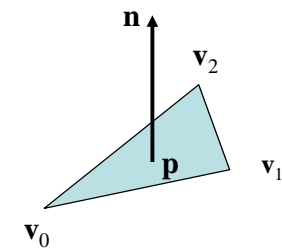
Normal for a triangle



$$\text{plane } \mathbf{n} \cdot (\mathbf{p} - \mathbf{v}_0) = 0$$

$$\mathbf{n} = (\mathbf{v}_2 - \mathbf{v}_0) \times (\mathbf{v}_1 - \mathbf{v}_0)$$

$$\text{normalize } \mathbf{n} \leftarrow \mathbf{n} / |\mathbf{n}|$$

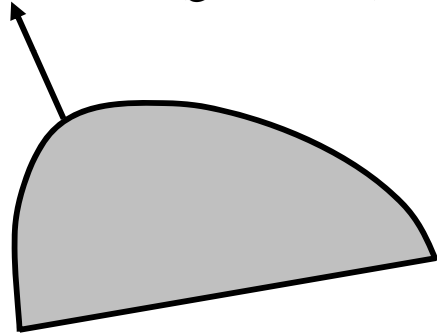


Note that right-hand rule determines outward face

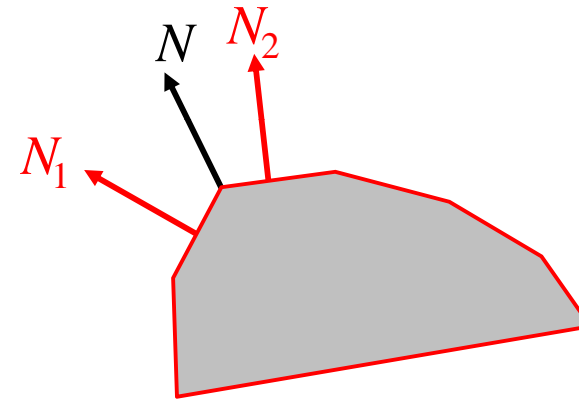
Using average normals



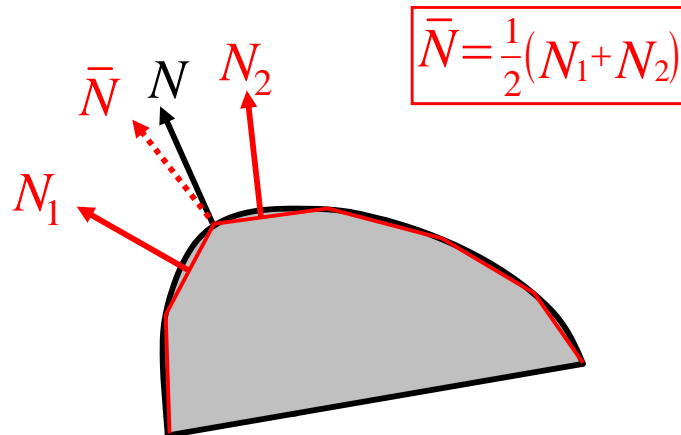
N = true (geometric) normal



Using average normals



Using average normals



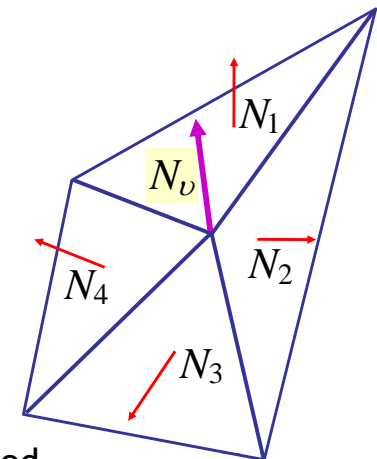
Using average normals



$$N_v = \frac{(N_1 + N_2 + N_3 + N_4)}{\|N_1 + N_2 + N_3 + N_4\|}$$

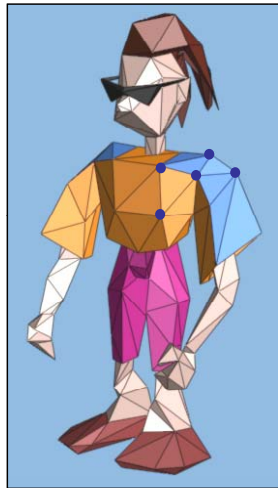
More generally,

$$N_v = \frac{\sum_{i=1}^n N_i}{\left| \sum_{i=1}^n N_i \right|}$$



It can also be area-weighted.

Triangle meshes



Copyright © 1998, Microsoft

$\{f_1\} : \{v_1, v_2, v_3\}$

$\{f_2\} : \{v_3, v_2, v_4\}$

...

$\{v_1\} : (x, y, z)$

$\{v_2\} : (x, y, z)$

...

$\{f_1\} : \text{"skin material"}$

$\{f_2\} : \text{"brown hair"}$

...

$\{v_2, f_1\} : (n_x, n_y, n_z) (u, v)$

$\{v_2, f_2\} : (n_x, n_y, n_z) (u, v)$

...

connectivity

geometry

face attributes

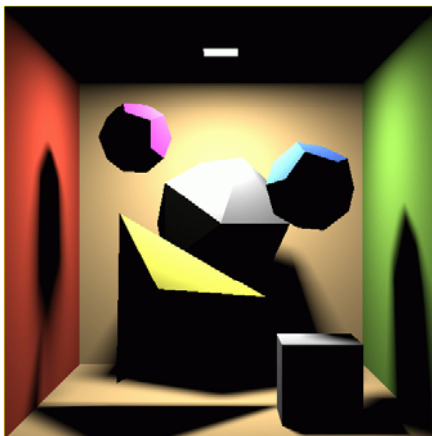
corner attributes

Illumination (shading) models

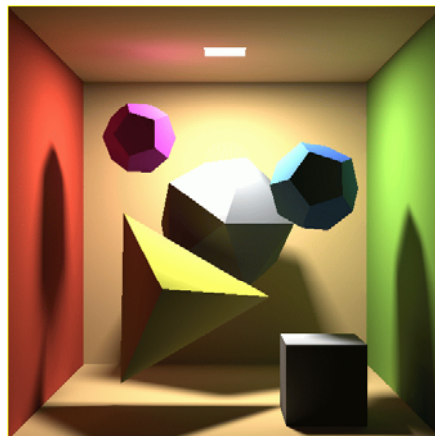


- Interaction between light sources and objects in scene that results in perception of intensity and color at eye
- Local** vs. **global** models
 - Local: perception of a particular primitive only depends on light sources directly affecting that one primitive
 - Geometry
 - Material properties
 - Shadows cast (global?)
 - Global: also take into account indirect effects on light of other objects in the scene
 - Light reflected/refracted
 - Indirect lighting

Local vs. global models

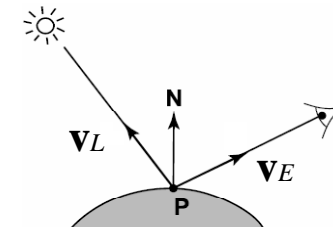


Direct lighting



Indirect lighting

Setup

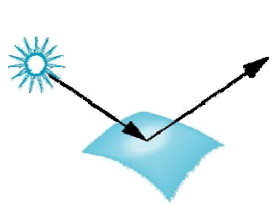


- Point **P** on a surface through a pixel **p**
- Normal **N** at **P**
- Lighting direction \mathbf{v}_L
- Viewing direction \mathbf{v}_E
- Compute color **L** for pixel **p**

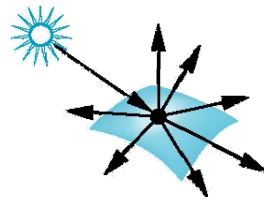
Surface types



- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
- A very rough surface scatters light in all directions



smooth surface

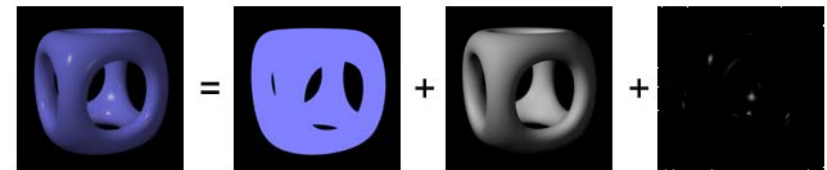


rough surface

Basics of local shading



- Diffuse reflection
 - light goes everywhere; colored by object color
- Specular reflection
 - happens only near mirror configuration; usually white
- Ambient reflection
 - constant accounted for other source of illumination



color and ambient

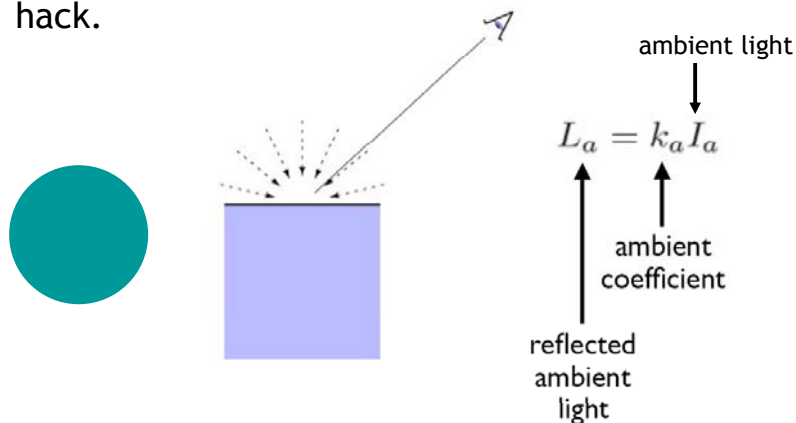
diffuse

specularity

Ambient shading



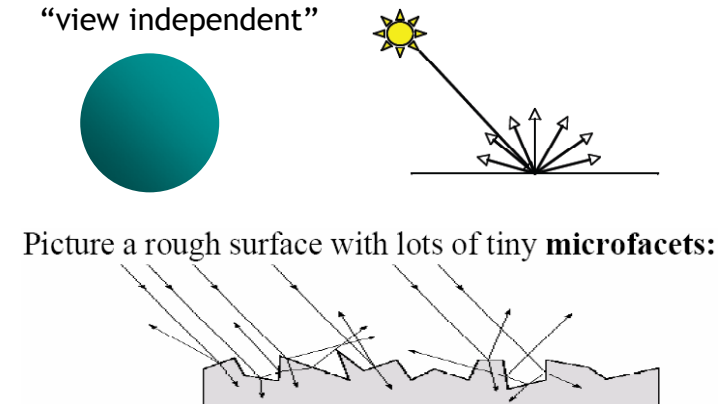
- add constant color to account for disregarded illumination and fill in black shadows; a cheap hack.



Diffuse shading



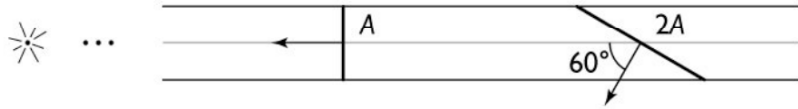
- Assume light reflects equally in all directions
 - Therefore surface looks same color from all views; “view independent”



Diffuse shading



- Illumination on an oblique surface is less than on a normal one (Lambertian cosine law)

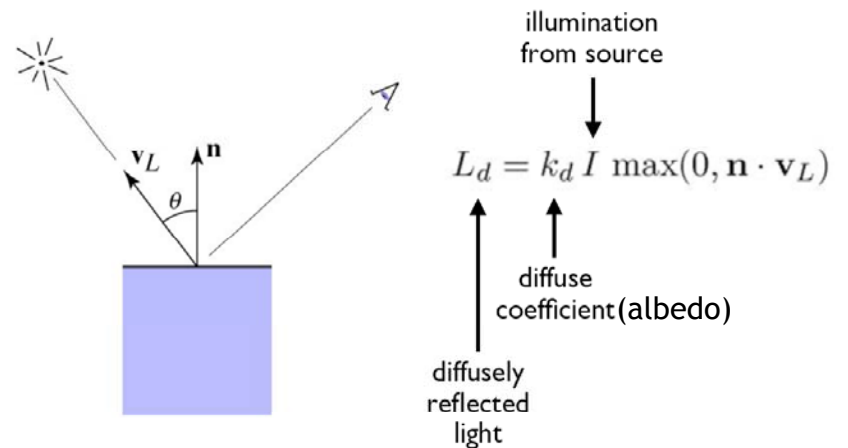


- Generally, illumination falls off as $\cos\theta$

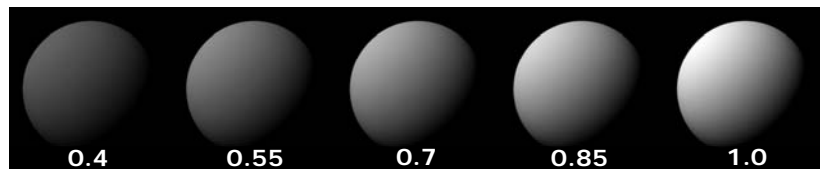
Diffuse shading (Gouraud 1971)



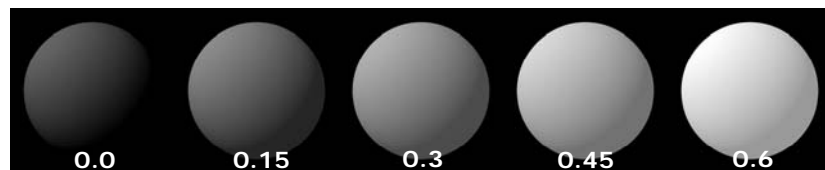
- Applies to *diffuse*, *Lambertian* or *matte* surfaces



Diffuse shading



diffuse-reflection model with different k_d



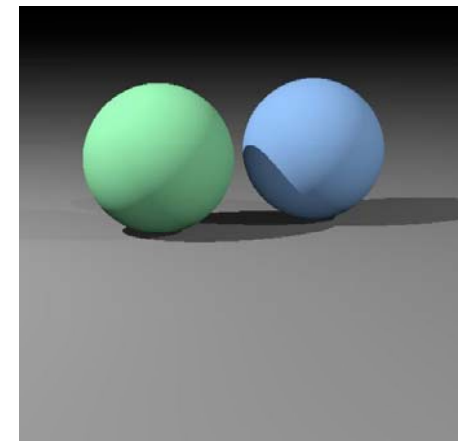
ambient and diffuse-reflection model with different k_a

and $I_a = I_p = 1.0, k_d = 0.4$

Diffuse shading



For color objects, apply the formula for each color channel separately



Specular shading



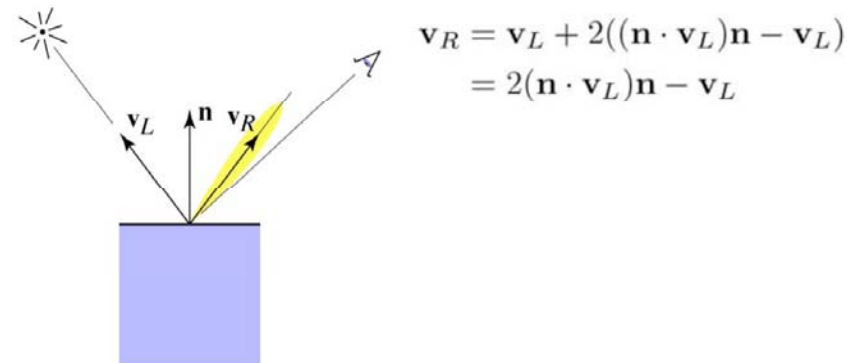
- Some surfaces have highlights, mirror like reflection; view direction dependent; especially for smooth shiny surfaces



Specular shading (Phong 1975)



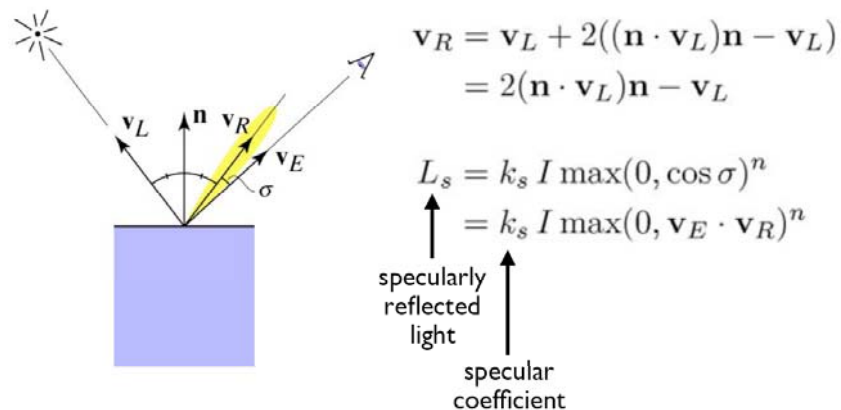
- Also known as *glossy*, *rough specular* and *directional diffuse* reflection



Specular shading



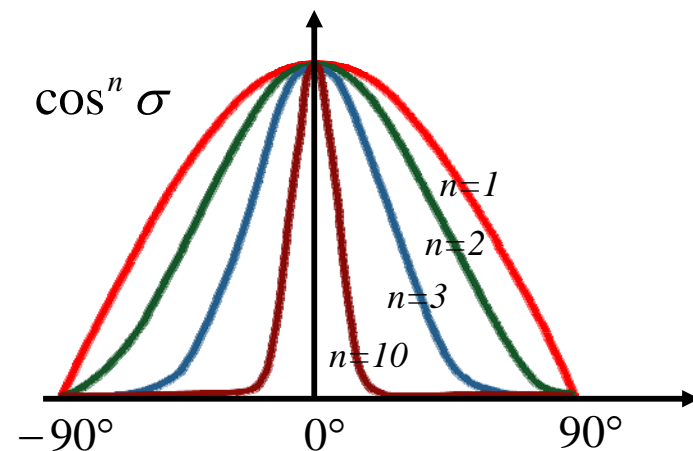
- Fall off gradually from the perfect reflection direction



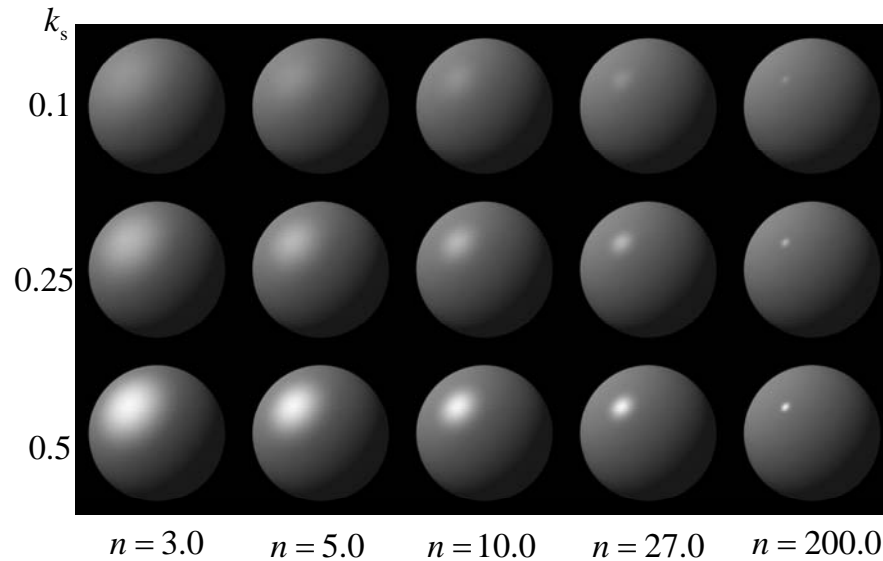
Specular shading



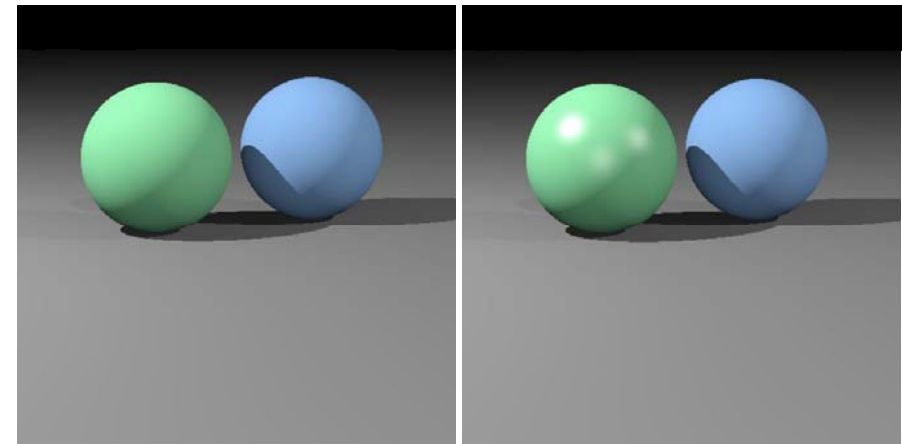
- Increasing n narrows the lobe



Specular shading



Specular shading



diffuse

diffuse + specular

Put it all together



- Include ambient, diffuse and specular

$$L = L_a + L_d + L_s$$

$$= k_a I_a + I (k_d \max(0, \mathbf{n} \cdot \mathbf{v}_L) + k_s \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n)$$

- Sum over many lights

$$L = L_a + \sum_i (L_d)_i + (L_s)_i$$

$$= k_a I_a + \sum_i I_i (k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n)$$

Choosing the parameters



n_s in the range $[0, 100]$

Try $k_a + k_d + k_s \leq 1$

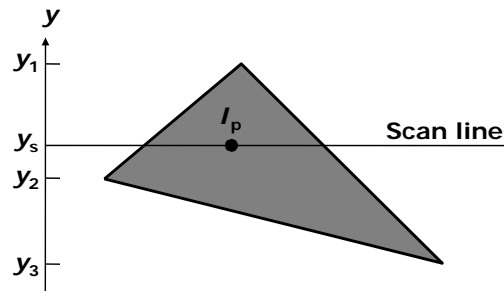
Use a small k_a (~ 0.1)

	n_s	k_d	k_s
Metal	Large	Small, color of metal	Large, color of metal
Plastic	Medium	Medium, color of plastic	Medium, white
Planet	0	Varying	0

Computing lighting at each pixel



- Most accurate approach: Compute component illumination at each pixel with individual positions, light directions, and viewing directions
- But this could be expensive...



Shading models for polygons



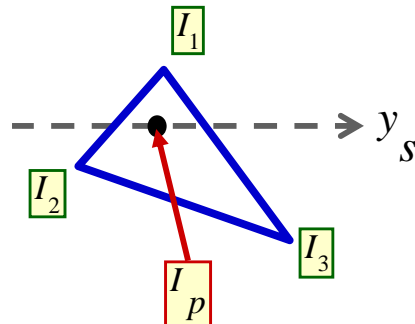
- Flat Shading
 - Faceted Shading
 - Constant Shading
- Gouraud Shading
 - Intensity Interpolation Shading
 - Color Interpolation Shading
- Phong Shading
 - Normal-Vector Interpolation Shading

Flat Shading



- Compute constant shading function, over each polygon
- Same normal and light vector across whole polygon
- Constant shading for polygon

$$I_p = I$$



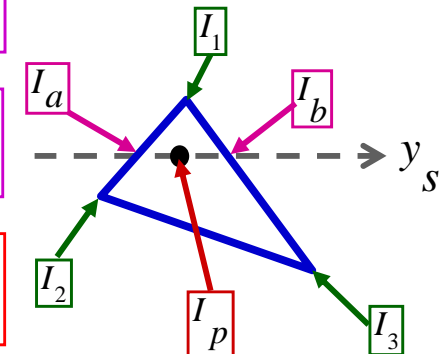
Intensity Interpolation (Gouraud)



$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$$

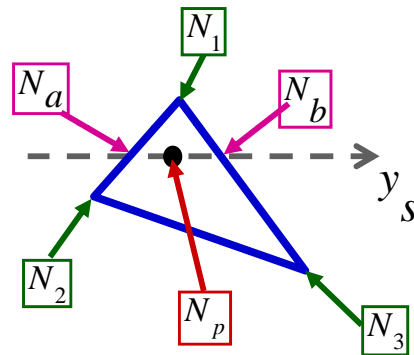


Normal Interpolation (Phong)



$$N_a = N_1 \frac{y_s - y_2}{y_1 - y_2} + N_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$N_b = N_1 \frac{y_s - y_3}{y_1 - y_3} + N_3 \frac{y_1 - y_s}{y_1 - y_3}$$



Normal Interpolation (Phong)



$$\tilde{N}_p = \frac{N_a}{\|N_a\|} \begin{bmatrix} x_b - x_p \\ x_b - x_a \end{bmatrix} + \frac{N_b}{\|N_b\|} \begin{bmatrix} x_p - x_a \\ x_b - x_a \end{bmatrix}$$

$$N_p = \frac{\tilde{N}_p}{\|\tilde{N}_p\|}$$

Normalizing makes
this a unit vector

Gouraud v.s. Phong Shading



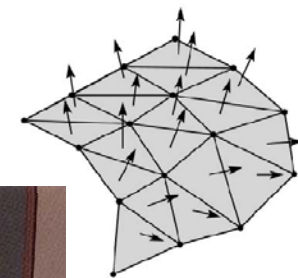
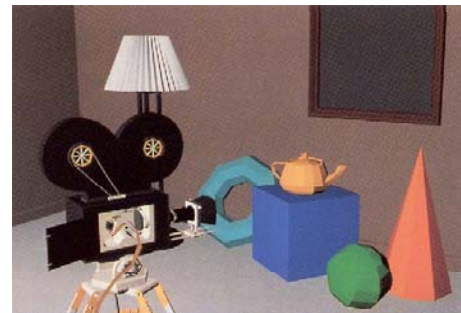
Gouraud

Phong

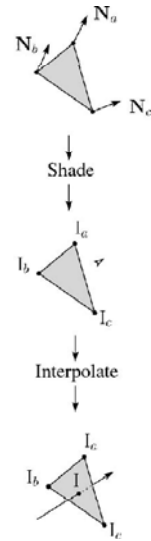
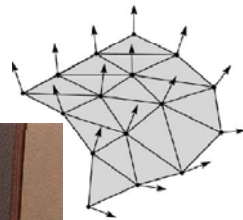
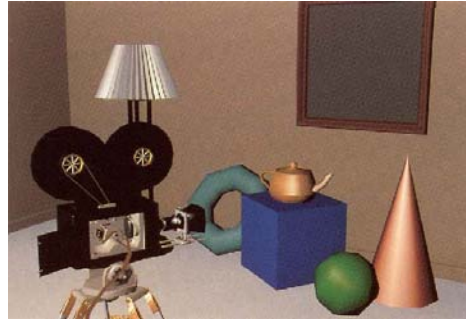
Gouraud

Phong

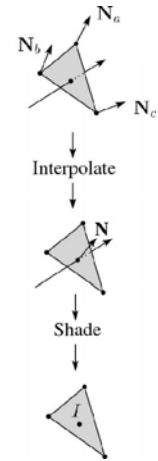
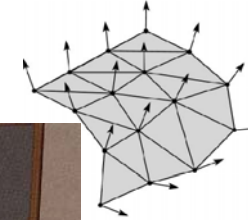
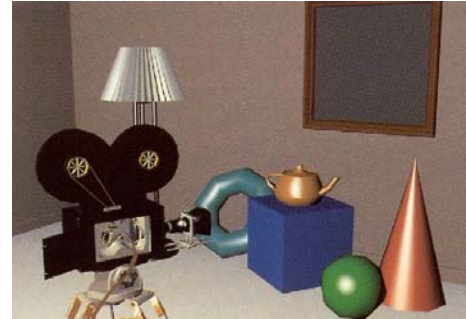
Flat shading



Gouraud shading

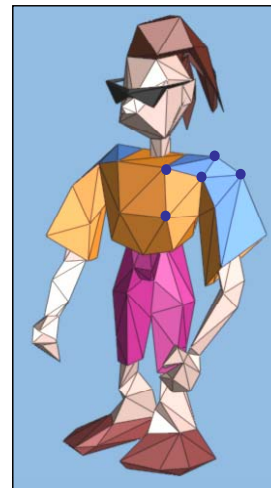


Phong shading



Graphics Pipeline

Triangle meshes



Copyright © 1998, Microsoft

$\{f_1\} : \{v_1, v_2, v_3\}$

$\{f_2\} : \{v_3, v_2, v_4\}$

...

$\{v_1\} : (x, y, z)$

$\{v_2\} : (x, y, z)$

...

$\{f_1\} : \text{"skin material"}$

$\{f_2\} : \text{"brown hair"}$

...

$\{v_2, f_1\} : (n_x, n_y, n_z) (u, v)$

$\{v_2, f_2\} : (n_x, n_y, n_z) (u, v)$

...

connectivity

geometry

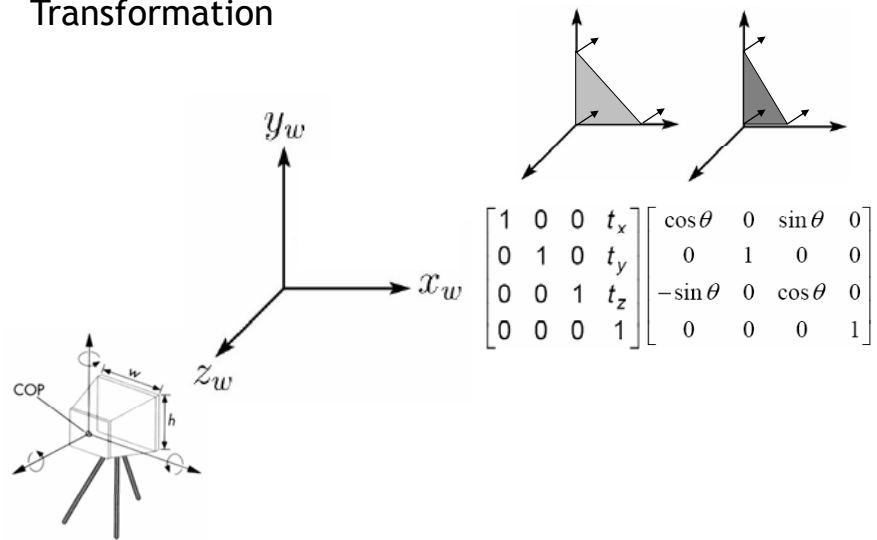
face attributes

corner attributes

Review of graphics pipeline



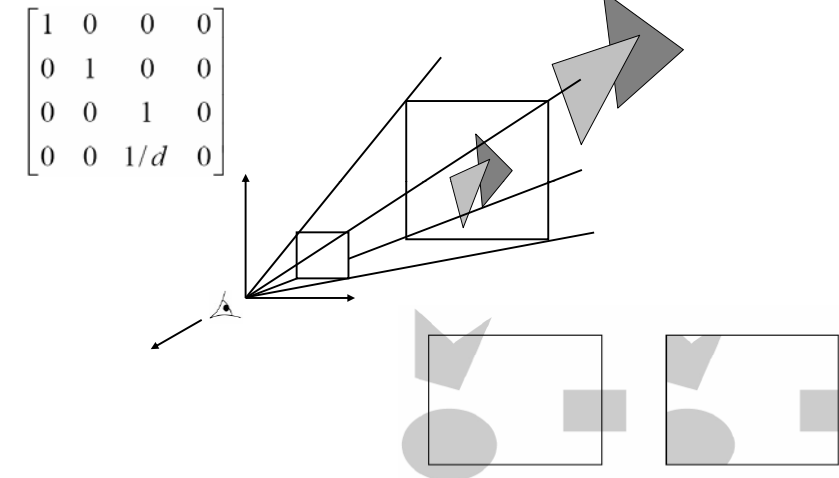
Transformation



Review of graphics pipeline



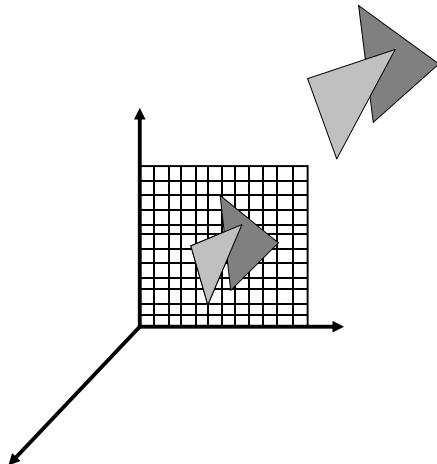
Projection & clipping



Review of graphics pipeline



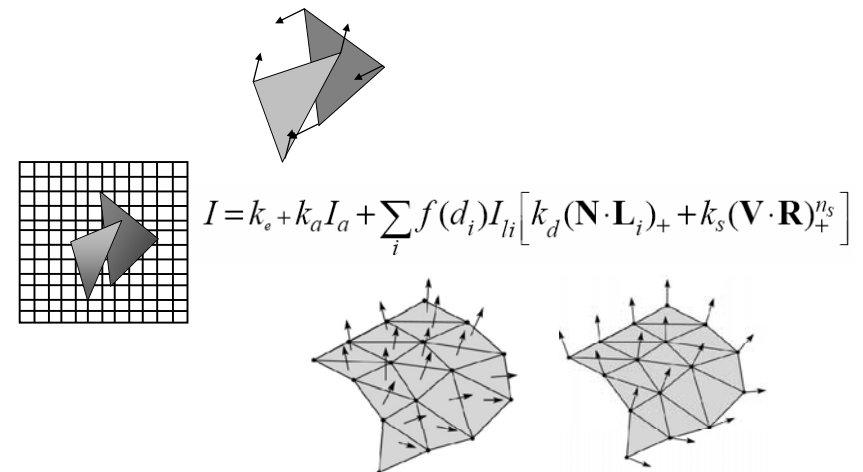
- Rasterization
- Visibility



Review of graphics pipeline



- Shading



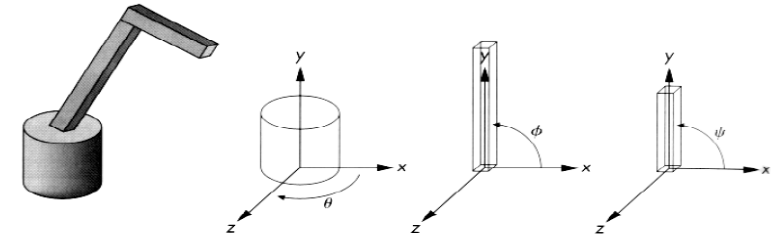
Animation

Hierarchical modeling: a robot arm



Consider this robot arm with 3 degrees of freedom:

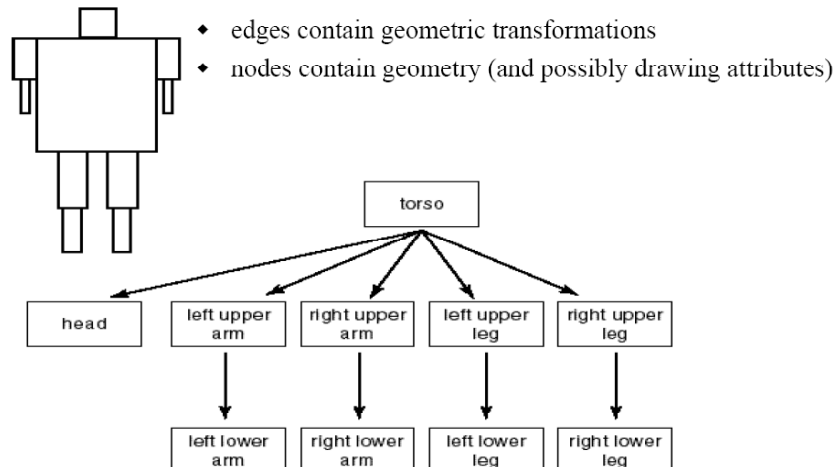
- Base rotates about its vertical axis by θ
- Lower arm rotates in its xy -plane by ϕ
- Upper arm rotates in its xy -plane by ψ



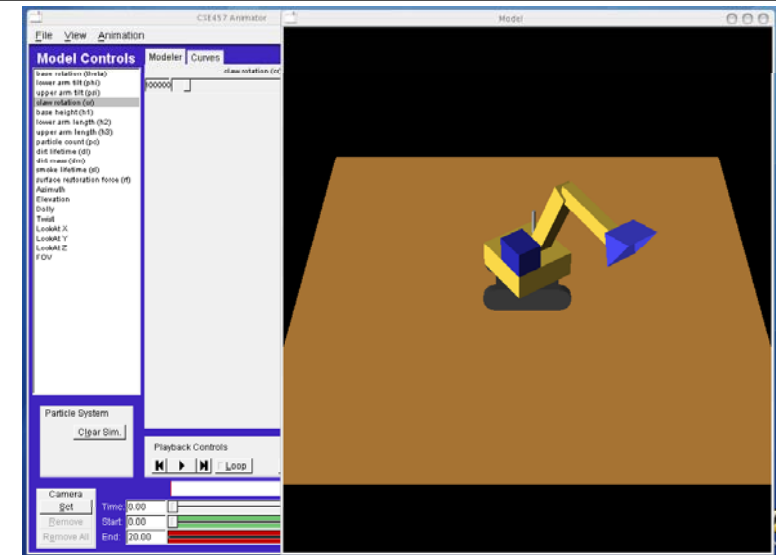
Hierarchical modeling



Hierarchical models can be composed of instances using trees or DAGs:



Animator demos



Videos



- [TigerWang](#)
- [Racing](#)

Advanced topics

Global illumination



$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t) (-\omega' \cdot \mathbf{n}) d\omega'$$

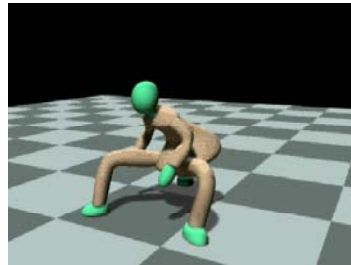
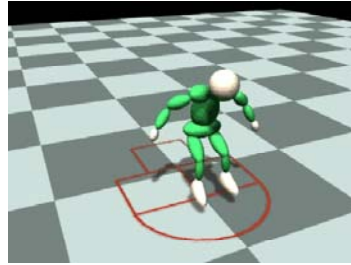
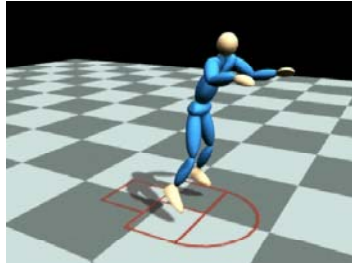


(c) 2004 VISUAL-PAATHOUSE - KRAUSELUTER

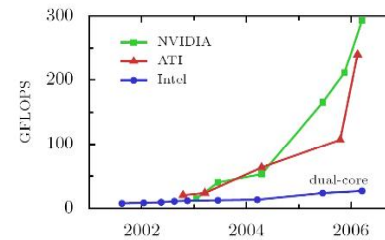
Complex materials



Realistic motion



Graphics hardware



Animation production

Animation production pipeline



story



text treatment



storyboard



voice



storyreal



look and feel

Animation production pipeline



modeling/articulation



layout



animation



shading/lighting



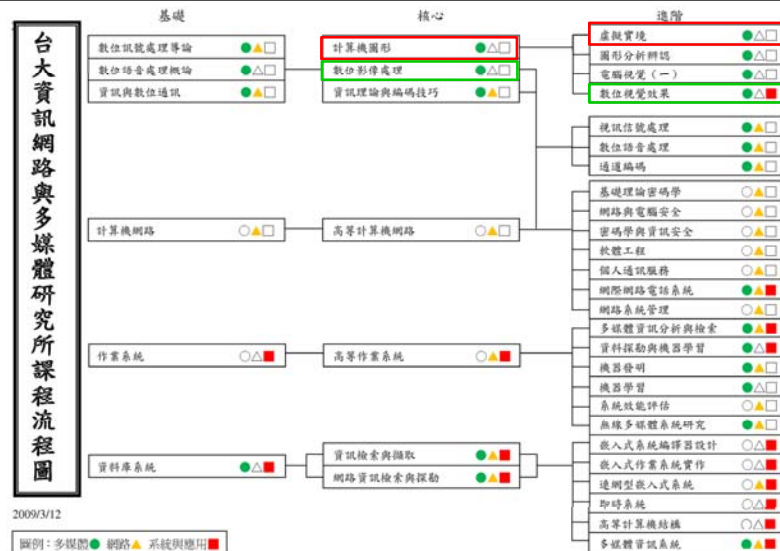
rendering



final touch

What's next?

Related courses



Related courses

