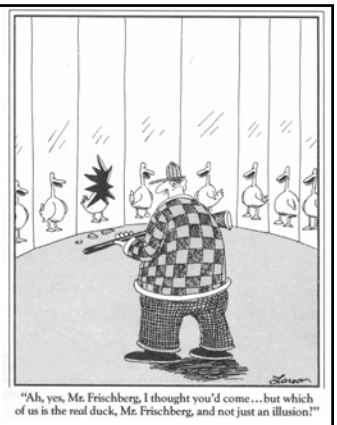
Transformations



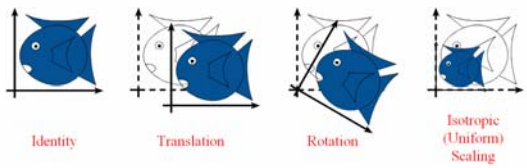
What is a Transformation?

- Maps points (x, y) in one coordinate system to points (x', y') in another coordinate system

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

- For example, IFS:

Simple Transformations

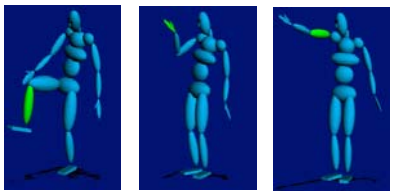


- Can be combined
- Are these operations invertible?

  Yes, except scale = 0

Transformations are used to:

- Position objects in a scene
- Change the shape of objects
- Create multiple copies of objects
- Projection for virtual cameras
- Describe animations

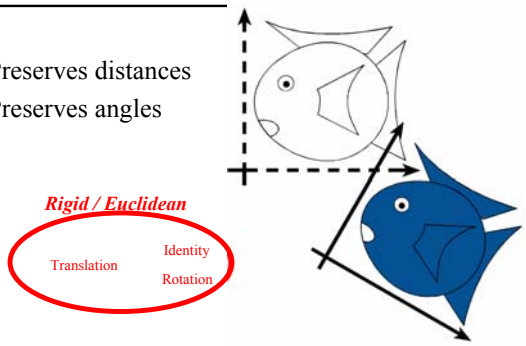


Outline

- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modelling
- Orthographic & Perspective Projections
- OpenGL Basics
- Color/Normal Interpolation

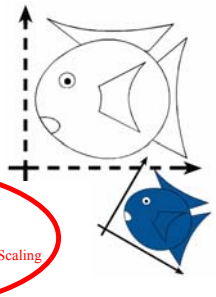
Rigid-Body / Euclidean Transforms

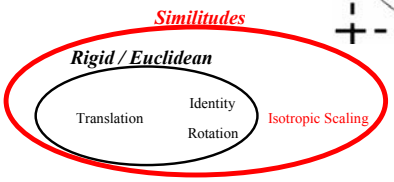
- Preserves distances
- Preserves angles



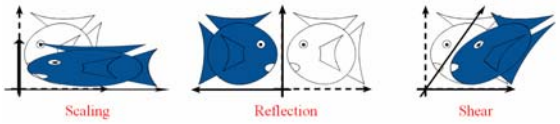
Similitudes / Similarity Transforms

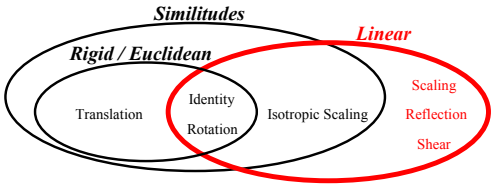
- Preserves angles





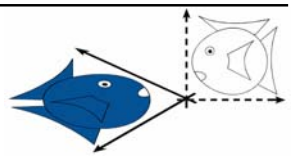
Linear Transformations

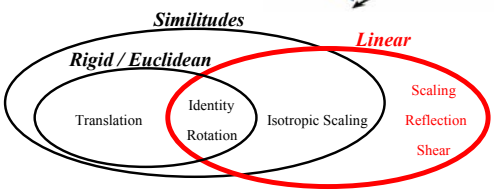




Linear Transformations

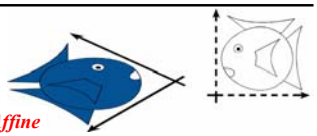
- $L(p + q) = L(p) + L(q)$
- $L(ap) = a\,L(p)$

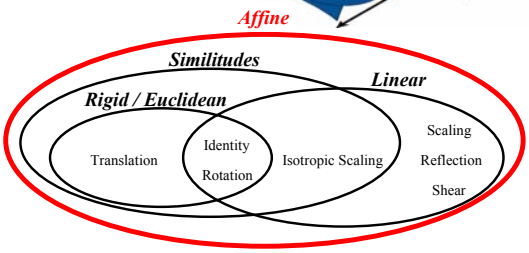




Affine Transformations

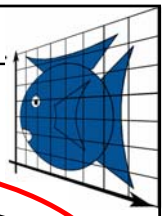
- preserves parallel lines

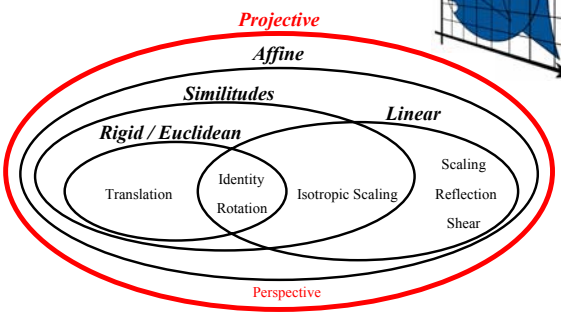




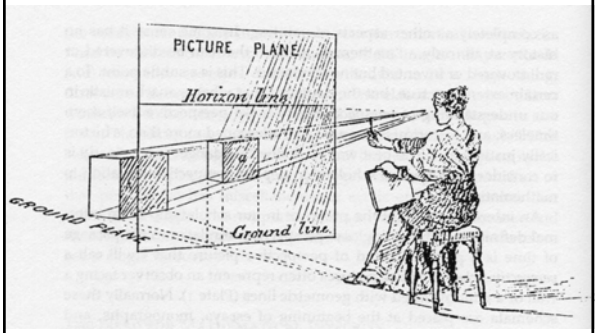
Projective Transformations

- preserves lines





Perspective Projection



General (free-form) transformation

- Does not preserve lines
- Not as pervasive, computationally more involved

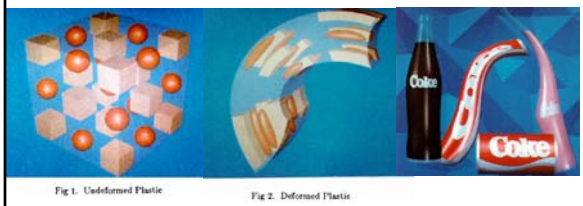


Fig 1. Undeformed Plastic

Fig 2. Deformed Plastic

From Sederberg and Parry, Siggraph 1986

Outline

- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modelling
- Orthographic & Perspective Projections
- OpenGL Basics
- Color/Normal Interpolation

How are Transforms Represented?

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

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

$$p' = Mp + t$$

Homogeneous Coordinates

- Add an extra dimension
  - in 2D, we use 3 x 3 matrices
  - In 3D, we use 4 x 4 matrices
- Each point has an extra value, w

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$p' = Mp$$

Translation in homogenous coordinates

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

Affine formulation

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

$$p' = Mp + t$$

Homogeneous formulation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$p' = Mp$$

Homogeneous Coordinates

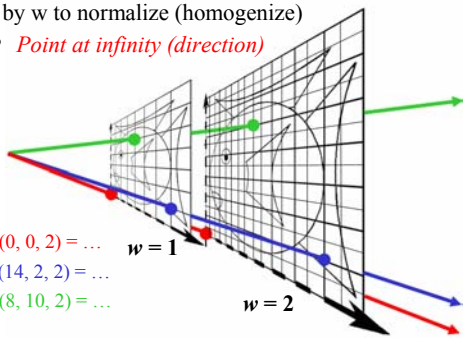
- Most of the time w = 1, and we can ignore it

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- If we multiply a homogeneous coordinate by an *affine matrix*, w is unchanged

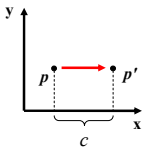
Homogeneous Visualization

- Divide by w to normalize (homogenize)
- W = 0? *Point at infinity (direction)*



Translate (t_x, t_y, t_z)

Translate(c,0,0)

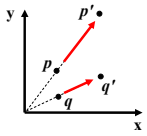


- Why bother with the extra dimension?
  Because now translations can be encoded in the matrix!

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

Scale (s_x, s_y, s_z)

Scale(s,s,s)

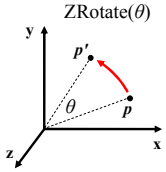


- Isotropic (uniform) scaling: $s_x = s_y = s_z$

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

Rotation

ZRotate(θ)

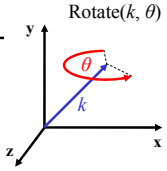


- About z axis

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

Rotation

Rotate(k, θ)



- About (k_x, k_y, k_z), a unit vector on an arbitrary axis (Rodrigues Formula)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} k_xk_x(1-c)+c & k_zk_x(1-c)-k_zs & k_xk_z(1-c)+k_ys & 0 \\ k_yk_x(1-c)+k_zs & k_zk_x(1-c)+c & k_yk_z(1-c)-k_xs & 0 \\ k_zk_x(1-c)-k_ys & k_zk_x(1-c)-k_xs & k_zk_z(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

where $c = \cos\theta$ & $s = \sin\theta$

Storage

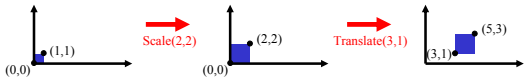
- Often, w is not stored (always 1)
- Needs careful handling of direction vs. point
  - Mathematically, the simplest is to encode directions with w=0
  - In terms of storage, using a 3 component array for both direction and points is more efficient
  - Which requires to have special operation routines for points vs. directions

Outline

- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modelling
- Orthographic & Perspective Projections
- OpenGL Basics
- Color/Normal Interpolation

How are transforms combined?

Scale then Translate



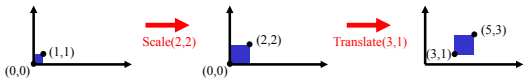
Use matrix multiplication: $p' = T(Sp) = TSp$

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

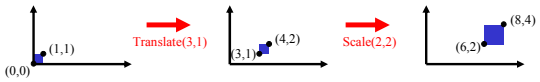
Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

Scale then Translate: $p' = T(Sp) = TSp$



Translate then Scale: $p' = S(Tp) = STp$



Non-commutative Composition

Scale then Translate: $p' = T(Sp) = TSp$

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Translate then Scale: $p' = S(Tp) = STp$

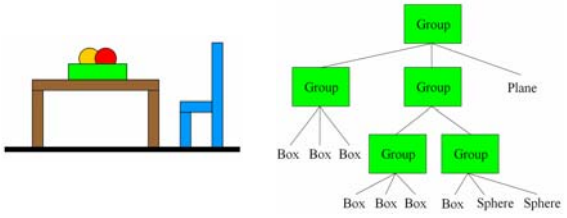
$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Today

- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modelling
- Orthographic & Perspective Projections
- OpenGL Basics
- Color/Normal Interpolation

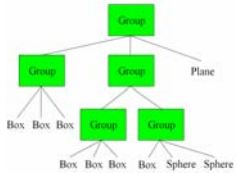
Scene Hierarchy

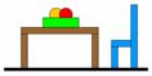
- Logical organization of scene



Simple Example with Groups

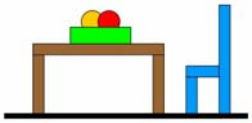
```
Group {
    numObjects 3
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } }
        Group {
            Box { <BOX PARAMS> }
            Sphere { <SPHERE PARAMS> }
            Sphere { <SPHERE PARAMS> } } }
    Plane { <PLANE PARAMS> } }
```





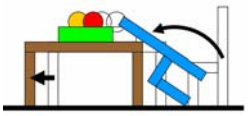
Adding Materials

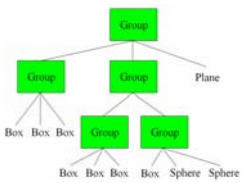
```
Group {
    numObjects 3
    Material { <BLUE> }
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> } }
    Group {
        numObjects 2
        Material { <BROWN> }
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } }
        Group {
            Material { <GREEN> }
            Box { <BOX PARAMS> }
            Material { <RED> }
            Sphere { <SPHERE PARAMS> }
            Material { <ORANGE> }
            Sphere { <SPHERE PARAMS> } } }
            Material { <BLACK> }
    Plane { <PLANE PARAMS> } }
```

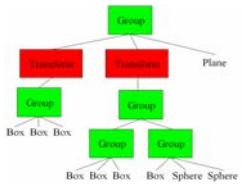


Adding Transformations

- To position the logical groupings of objects within the scene

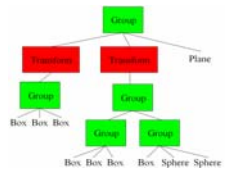






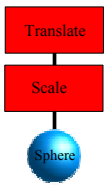
Simple Example with Transforms

```
Group {
    numObjects 3
    Transform {
        ZRotate { 45 }
        Group {
            numObjects 3
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } } }
    Transform {
        Translate { -2 0 0 }
        Group {
            numObjects 2
            Group {
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> } }
            Group {
                Box { <BOX PARAMS> }
                Sphere { <SPHERE PARAMS> }
                Sphere { <SPHERE PARAMS> } } } }
    Plane { <PLANE PARAMS> } }
```

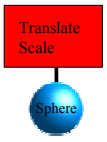


Nested Transforms

$$p' = T(Sp) = TSp$$



same as




```
Transform {
    Translate { 1 0.5 0 }
    Transform {
        Scale { 2 2 2 }
        Sphere {
            center 0 0 0
            radius 1 } } }
```

```
Transform {
    Translate { 1 0.5 0 }
    Scale { 2 2 2 }
    Sphere {
        center 0 0 0
        radius 1 } }
```

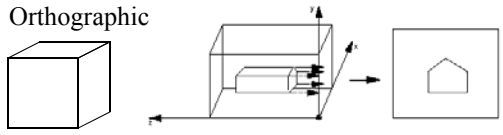
Questions?

Today

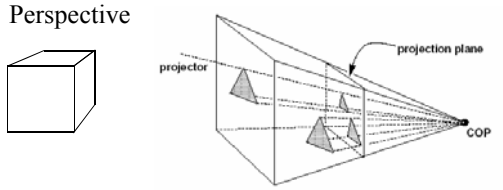
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modelling
- Orthographic & Perspective Projections
- OpenGL Basics
- Color/Normal Interpolation

Orthographic vs. Perspective

- Orthographic

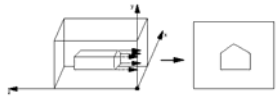


- Perspective



Simple Orthographic Projection

- Project all points along the z axis to the $z = 0$ plane

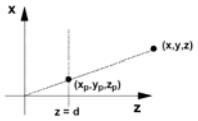


$$\begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Simple Perspective Projection

- Project all points along the z axis to the $z = d$ plane, eyepoint at the origin:

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}$$
$$y_p = \frac{d \cdot y}{z} = \frac{y}{z/d}$$
$$z_p = d$$



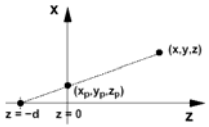
homogenize

$$\begin{pmatrix} x * d / z \\ y * d / z \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z / d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Alternate Perspective Projection

- Project all points along the z axis to the $z = 0$ plane, eyepoint at the $(0,0,-d)$:

$$x_p = \frac{d \cdot x}{z + d} = \frac{x}{(z/d) + 1}$$
$$y_p = \frac{d \cdot y}{z + d} = \frac{y}{(z/d) + 1}$$



homogenize

$$\begin{pmatrix} x * d / (z + d) \\ y * d / (z + d) \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ (z + d)/ d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

In the limit, as $d \rightarrow \infty$

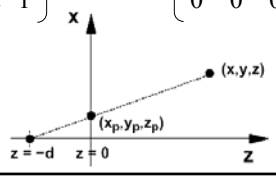
this perspective projection matrix...

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}$$

$\rightarrow$

...is simply an orthographic projection

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

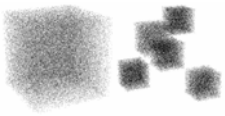


Today

- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modelling
- Orthographic & Perspective Projections
- OpenGL Basics
- Color/Normal Interpolation

OpenGL Basics: GL_POINTS

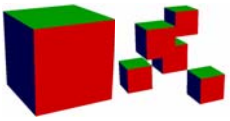
```
glDisable(GL_LIGHTING);
glBegin(GL_POINTS);
glColor3f(0.0,0.0,0.0);
glVertex3f(…);
glEnd();
```



- lighting should be *disabled*...

OpenGL Basics: GL_QUADS

```
glEnable(GL_LIGHTING);
glBegin(GL_QUADS);
glNormal3f(…);
glColor3f(1.0,0.0,0.0);
glVertex3f(…);
glVertex3f(…);
glVertex3f(…);
glVertex3f(…);
glEnd();
```

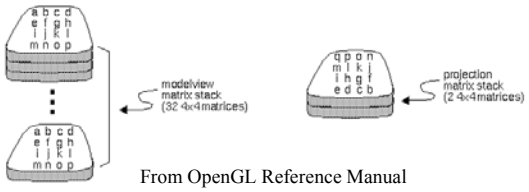


- lighting should be *enabled*...
- an appropriate normal should be specified

OpenGL Basics: Transformations

- Useful commands:

```
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glPopMatrix();
glMultMatrixf(…);
```



From OpenGL Reference Manual

Questions?

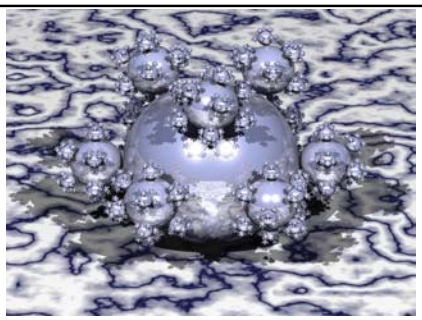


Image by Henrik Wann Jensen

Today

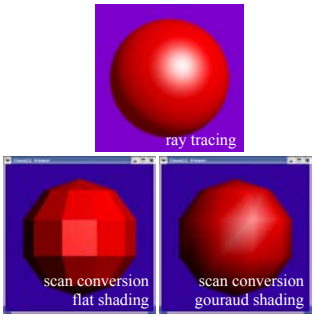
- Classes of Transformations
- Representing Transformations
- Combining Transformations
- Transformations in Modelling
- Orthographic & Perspective Projections
- OpenGL Basics
- Color/Normal Interpolation

Color Interpolation

- Interpolate colors of the 3 vertices
- Linear interpolation, barycentric coordinates

```
glBegin(GL_TRIANGLES);
glColor3f(1.0,0.0,0.0);
glVertex3f(…);
glColor3f(0.0,1.0,0.0);
glVertex3f(…);
glColor3f(0.0,0.0,1.0);
glVertex3f(…);
glEnd();
```

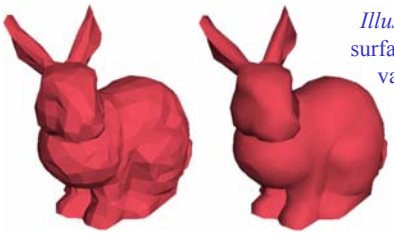

Normal Interpolation




```
glBegin(GL_TRIANGLES);
glNormal3f(…);
glVertex3f(…);
glNormal3f(…);
glVertex3f(…);
glNormal3f(…);
glVertex3f(…);
glEnd();
```


Gouraud Shading

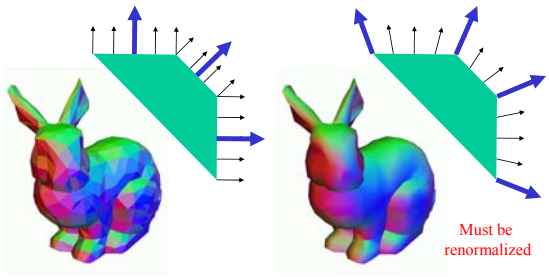
- Instead of shading with the normal of the triangle, shade the vertices with the *average normal* and *interpolate the shaded color* across each face



Illusion of a smooth surface with smoothly varying normals

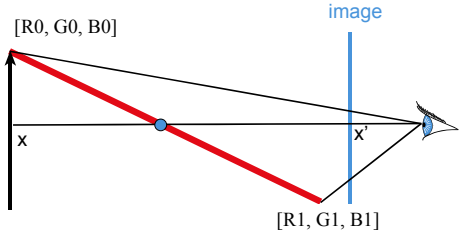
Phong Normal Interpolation (Not Phong *Shading*)

- Interpolate the average vertex normals across the face and compute *per-pixel shading*



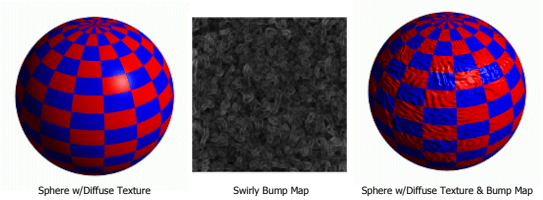
Gouraud Interpolation

- Gouraud: interpolate color linearly in screen space. Is it correct?
- No, we should use hyperbolic interpolation It's costly (division) but can now be done on modern hardware



Bump Mapping

- Use textures to alter the surface normal
  - Does not change the actual shape of the surface
  - Just shaded as if it were a different shape

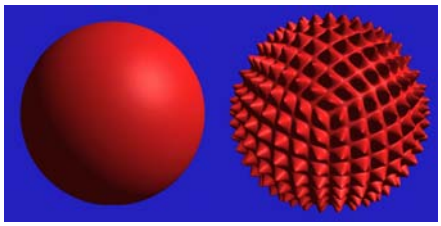


What's Missing?

- There are no bumps on the silhouette of a bump-mapped object
- Bump maps don't allow self-occlusion or self-shadowing

Displacement Mapping

- Use the texture map to actually move the surface point
- The geometry must be displaced before visibility is determined



Displacement Mapping

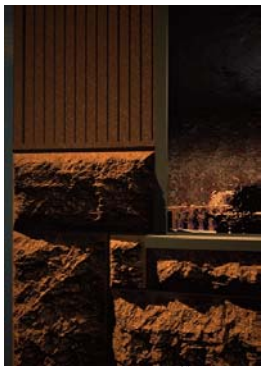


Image from:

Geometry Caching for Ray-Tracing Displacement Maps

by Matt Pharr and Pat Hanrahan.

note the detailed shadows cast by the stones

Displacement Mapping

Ken Musgrave

Questions?