Textbook Errata
- list at http://www.cs.utah.edu/~shirley/fcg/errata
  * math review: also p 48
  * a x (b x c) = (a x b) x c
  * transforms: p 91
    * should halve x (not y) in Fig 5.10
  * transforms: p 106
    * second line of matrices: [x, y, 1]

Correction: Vector-Vector Subtraction
- subtract: vector - vector = vector
  \[ \mathbf{u} - \mathbf{v} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \\ u_3 - v_3 \end{bmatrix} \]

Correction: Vector-Vector Multiplication
- multiply: vector * vector = scalar
- dot product, aka inner product
  \[ \mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + u_3 v_3 \]
- geometric interpretation
  * lengths, angles
  * can find angle between two vectors
  \[ \mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos \theta \]

Correction: Matrix Multiplication
- can only multiply (n,k) by (k,m)
- number of left cols = number of right rows
  * legal
    \[ \begin{bmatrix} a & b & c \\ e & f & g \end{bmatrix} \begin{bmatrix} h & i \\ j & k \end{bmatrix} = \begin{bmatrix} \_ & \_ \\ \_ & \_ \end{bmatrix} \]
  * undefined
    \[ \begin{bmatrix} a & b & c \\ e & f & g \end{bmatrix} \begin{bmatrix} h & i \\ j & k \end{bmatrix} = \begin{bmatrix} \_ & \_ \\ \_ & \_ \end{bmatrix} \]
**Correction: Matrices and Linear Systems**

- linear system of n equations, n unknowns
  
  \[ \begin{align*}
  3x + 7y + 2z &= 4 \\
  2x - 4y - 3z &= -1 \\
  5x + 2y + z &= 1
  \end{align*} \]

- matrix form \( Ax = b \)

\[
\begin{pmatrix}
3 & 7 & 2 \\
2 & -4 & -3 \\
5 & 2 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
=
\begin{pmatrix}
4 \\
-1 \\
1
\end{pmatrix}
\]

---

**Review: Rendering Pipeline**

- **Geometry Database**
- **Model/View Transform**
- **Lighting**
- **Perspective Transform**
- **Clipping**

1. **Scan Conversion**
   - scan conversion
     - turn 2D drawing primitives (lines, polygons etc.) into individual pixels (discretizing/sampling)
     - interpolate color across primitive
     - generate discrete fragments

2. **Blending**
   - blending
     - final image: write fragments to pixels
     - draw from farthest to nearest
     - no blending – replace previous color
     - blending: combine new & old values with arithmetic operations

3. **Framebuffer**
   - framebuffer
     - video memory on graphics board that holds image
     - double-buffering: two separate buffers
     - draw into one while displaying other, then swap
     - allows smooth animation, instead of flickering

---

**Correction: Scan Conversion**

**Correction: Blending**

**Correction: Framebuffer**

**Review: OpenGL**

- pipeline processing, set state as needed

```c
void display()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, -0.5);
    glVertex3f(0.75, 0.25, -0.5);
    glVertex3f(0.75, 0.75, -0.5);
    glVertex3f(0.25, 0.75, -0.5);
    glEnd();
    glFlush();
}
```
Review: Event-Driven Programming

- main loop not under your control
- vs. procedural
- control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
- callback functions called from main loop when events occur
  - mouse/keyboard state setting vs. redrawing

Overview

- 2D Transformations
- Homogeneous Coordinates
- 3D Transformations
- Composing Transformations
- Transformation Hierarchies
- Display Lists
- Transforming Normals
- Assignments

Transformations

- transforming an object = transforming all its points
- transforming a polygon = transforming its vertices

Matrix Representation

- represent 2D transformation with matrix
- multiply matrix by column vector
- apply transformation to point

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

- transformations combined by multiplication

\[
\begin{bmatrix}
  x'' \\
  y''
\end{bmatrix} =
\begin{bmatrix}
  e & f & h & i \\
  g & j & k & l
\end{bmatrix}
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
\]

- matrices are efficient, convenient way to represent sequence of transformations!

Scaling

- scaling a coordinate means multiplying each of its components by a scalar
- uniform scaling means this scalar is the same for all components:

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} \rightarrow
\begin{bmatrix}
  2x \\
  2y
\end{bmatrix}
\]
Scaling

- non-uniform scaling: different scalars per component:

![Scaling Diagram]

- how can we represent this in matrix form?

Scaling

- scaling operation:

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  ax \\
  by
\end{bmatrix}
\]

- or, in matrix form:

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  a & 0 \\
  0 & b
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

2D Rotation

- counterclockwise
- RHS

\[
\begin{align*}
  x' &= x \cos(\theta) - y \sin(\theta) \\
  y' &= x \sin(\theta) + y \cos(\theta)
\end{align*}
\]

2D Rotation From Trig Identities

\[
\begin{align*}
  x &= r \cos(\phi) \\
  y &= r \sin(\phi) \\
  x' &= r \cos(\phi + \theta) \\
  y' &= r \sin(\phi + \theta)
\end{align*}
\]

Substitute…

\[
\begin{align*}
  x' &= x \cos(\theta) - y \sin(\theta) \\
  y' &= x \sin(\theta) + y \cos(\theta)
\end{align*}
\]

2D Rotation Matrix

- easy to capture in matrix form:

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  \cos(\theta) & -\sin(\theta) \\
  \sin(\theta) & \cos(\theta)
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

- even though \(\sin(q)\) and \(\cos(q)\) are nonlinear functions of \(q\),
- \(x'\) is a linear combination of \(x\) and \(y\)
- \(y'\) is a linear combination of \(x\) and \(y\)

2D Rotation: Another Derivation

\[
\begin{align*}
  x' &= x \cos(\theta) - y \sin(\theta) \\
  y' &= x \sin(\theta) + y \cos(\theta)
\end{align*}
\]
2D Rotation: Another Derivation

\[ x' = x \cos \theta - y \sin \theta \]
\[ y' = x \sin \theta + y \cos \theta \]

Shear

- shear along x axis
- push points to right in proportion to height

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} =
\begin{bmatrix}
  1 & \theta \\
  0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]
Shear
- shear along x axis
- push points to right in proportion to height

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix} 1 & sh \ x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]

Reflection
- reflect across x axis
- mirror

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]

2D Translation
- vector addition

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix} x + a \\ y + b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}
\]

- matrix multiplication

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \]

scaling matrix
rotation matrix
matrix multiplication
matrix multiplication
2D Translation

- Matrix multiplication
  - \( \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} \)

- Vector addition
  - \( \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \end{bmatrix} \)

- Translation multiplication matrix

Linear Transformations

- Linear transformations are combinations of:
  - Shear
  - Scale
  - Rotate
  - Reflect

- Properties of linear transformations:
  - Satisfies \( T(ax + by) = a T(x) + b T(y) \)
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

Challenge

- Matrix multiplication
  - For everything except translation
  - How to do everything with multiplication?
  - Then just do composition, no special cases

- Homogeneous coordinates trick
  - Represent 2D coordinates \((x, y)\) with 3-vector \((x, y, 1)\)

Homogeneous Coordinates

- Our 2D transformation matrices are now 3x3:
  - \( \begin{bmatrix} a & b & x' \\ c & d & y' \\ 0 & 0 & 1 \end{bmatrix} \)
  - Use rightmost column

- Homogeneous coordinates geometrically:
  - Point in 2D Cartesian

- Cartesian to homogeneous:
  - \( (x, y, w) \)

- Homogeneous coordinates geometrically:
  - Point in 2D Cartesian + weight \( w \) = point \( P \) in 3D homog. coords
  - Muliplies of \((x, y, w)\)
    - Form a line \( L \) in 3D
    - All homogeneous points on \( L \) represent same 2D cartesian point
    - Example: \((2, 2, 1) \rightarrow (4, 4, 2) \rightarrow (1, 1, 0.5)\)
Homogeneous Coordinates Geometrically

- **Homogenize** to convert homog. 3D point to cartesian 2D point:
  - divide by w to get \((x/w, y/w, 1)\)
  - projects line to point onto \(w=1\) plane
- when \(w=0\), consider it as direction
- points at infinity: these points cannot be homogenized
- lies on \(x-y\) plane
- \((0,0,0)\) is undefined

Homogeneous Coordinates Summary

- may seem unintuitive, but they make graphics operations much easier
- allow all linear transformations to be expressed through matrix multiplication
- use 4x4 matrices for 3D transformations

Affine Transformations

- affine transforms are combinations of:
  - linear transformations
  - translations
- properties of affine transformations:
  - origin does not necessarily map to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

3D Rotation About Z Axis

\[
\begin{align*}
    x' &= x \cos \theta - y \sin \theta \\
    y' &= x \sin \theta + y \cos \theta \\
    z' &= z
\end{align*}
\]

- general OpenGL command
  - `glRotatef(angle,0,0,1);`

3D Rotation in X, Y

- around \(x\) axis: `glRotatef(angle,1,0,0);`
- \[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & x \\
    0 & \cos \theta & \sin \theta & 0 \\
    0 & -\sin \theta & \cos \theta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]
- around \(y\) axis: `glRotatef(angle,0,1,0);`
- \[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & 0 & \sin \theta & 0 \\
    0 & 1 & 0 & 0 \\
    -\sin \theta & 0 & \cos \theta & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

3D Scaling

- \[
\begin{bmatrix}
    x' \\
    y' \\
    z'
\end{bmatrix} =
\begin{bmatrix}
    a & 0 & 0 & x \\
    0 & b & 0 & y \\
    0 & 0 & c & z \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]
- `glScalef(a,b,c);`
3D Translation

\[ \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a & 0 \\ 0 & 1 & 0 & b & 0 \\ 0 & 0 & 1 & c & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \]

gTranslate(a,b,c);

3D Shear

- Shear in x
  \[ \begin{bmatrix} 1 & sy & sz & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

- Shear in y
  \[ \begin{bmatrix} 1 & 0 & sz & 0 \\ 0 & 1 & sy & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

- Shear in z
  \[ \begin{bmatrix} 1 & 0 & 0 & sz \\ 0 & 1 & 0 & sy \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Summary: Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translate(a,b,c)</td>
<td>[ \begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; a &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; b &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; c &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 &amp; 1 \end{bmatrix} ]</td>
</tr>
<tr>
<td>Scale(a,b,c)</td>
<td>[ \begin{bmatrix} s_x &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; s_y &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; s_z &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 &amp; 1 \end{bmatrix} ]</td>
</tr>
<tr>
<td>Rotate(x,θ)</td>
<td>[ \begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; c &amp; s &amp; 0 &amp; -s \ 0 &amp; -s &amp; c &amp; 0 &amp; s \ 0 &amp; 0 &amp; 0 &amp; 1 &amp; 1 \end{bmatrix} ]</td>
</tr>
<tr>
<td>Rotate(y,θ)</td>
<td>[ \begin{bmatrix} c &amp; 0 &amp; -s &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 0 &amp; 0 \ s &amp; 0 &amp; c &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 &amp; 1 \end{bmatrix} ]</td>
</tr>
<tr>
<td>Rotate(z,θ)</td>
<td>[ \begin{bmatrix} c &amp; s &amp; 0 &amp; 0 &amp; 0 \ -s &amp; c &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 &amp; 1 \end{bmatrix} ]</td>
</tr>
</tbody>
</table>

Undoing Transformations: Inverses

\[ T(x,y,z)^{-1} = T(-x,-y,-z) \]
\[ R(\alpha,\beta,\gamma)^{-1} = R(-\alpha,-\beta,-\gamma) \] (R is orthogonal)

Composing Transformations

- Translation

\[ T_1 = T(dx,dy) \]
\[ T_2 = T(dx,dy) \]

\[ P' = T_2 \circ T_1 \]

so translations add
Composing Transformations

- scaling

\[
S \times S' = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

so scales multiply

- rotation

\[
R \times R' = \begin{bmatrix}
\cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) \\
\sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2)
\end{bmatrix}
\]

so rotations add

Ta Tb = Tb Ta, but Ra \neq Rb

suppose we want

\[
p' = R(z, -90)p
\]

\[
p'' = T(2, 3, 0)p'
\]

\[
p''' = T(2, 3, 0)R(z, -90)p = TRp
\]
Composing Transformations

\[ p' = TRp \]

- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
    - moving object
  - left to right
    - interpret operations wrt local coordinates
    - changing coordinate system

OpenGL pipeline ordering!

Interpreting Transformations

\[ p' = TRp \]

- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
  - left to right
    - OpenGL pipeline ordering!

- moving object
- changing coordinate system

translate by \((-1,0)\)

Intuitive?

Matrix Composition

- matrices are convenient, efficient way to represent series of transformations
  - general purpose representation
  - hardware matrix multiply
  - matrix multiplication is associative
    - \[ p' = (T^R)(S^R)p \]
  - \[ p' = (T^R)(S^R)p \]

procedure
  - correctly order your matrices!
  - multiply matrices together
  - result is one matrix, multiply vertices by this matrix
  - all vertices easily transformed with one matrix multiply

Rotation About a Point: Moving Object

\[ T(x,y,z)R(z,\theta)T(-x,-y,-z) \]
Rotation: Changing Coordinate Systems
- same example: rotation around arbitrary center

Step 1: Translate coordinate system to rotation center

Step 2: Perform rotation

Step 3: Back to original coordinate system

General Transform Composition
- Transformation of geometry into coordinate system where operation becomes simpler
  - Typically translate to origin
- Perform operation
- Transform geometry back to original coordinate system

Rotation About an Arbitrary Axis
- Axis defined by two points
- Translate point to the origin
- Rotate to align axis with z-axis (or x or y)
- Perform rotation
- Undo aligning rotations
- Undo translation
Arbitrary Rotation

problem:
- given two orthonormal coordinate systems XYZ and UVW
- find transformation from one to the other

answer:
- transformation matrix $R$ whose columns are $U, V, W$:
$$
R = \begin{bmatrix}
    u_x & u_y & u_z \\
    v_x & v_y & v_z \\
    w_x & w_y & w_z
\end{bmatrix}
$$

why?
- similarly $R(Y) = V$ & $R(Z) = W$

Transformation Hierarchies

- scene may have a hierarchy of coordinate systems
- stores matrix at each level with incremental transform from parent’s coordinate system

Transformation Hierarchy Example 1

Transformation Hierarchies

- hierarchies don’t fall apart when changed
- transforms apply to graph nodes beneath
Demo: Brown Applets

http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html

Transformation Hierarchy Example 2
- draw same 3D data with different transformations: instancing

Matrix Stacks
- challenge of avoiding unnecessary computation
- using inverse to return to origin
- computing incremental $T_1 \rightarrow T_2$

Matrix Stacks
- $T_1(x)$
- $T_2(x)$
- $T_3(x)$

Modularization
- drawing a scaled square
  - push/pop ensures no coord system change

void drawBlock(float k) {
    glPushMatrix();
    glScalef(k, k, k);
    glBegin(GL_LINE_LOOP);
    glVertex3f(0, 0, 0);
    glVertex3f(1, 0, 0);
    glVertex3f(1, 1, 0);
    glVertex3f(0, 1, 0);
    glEnd();
    glPopMatrix();
}

Matrix Stacks
- advantages
  - no need to compute inverse matrices all the time
  - modularize changes to pipeline state
  - avoids incremental changes to coordinate systems
  - accumulation of numerical errors
- practical issues
  - in graphics hardware, depth of matrix stacks is limited
    (typically 16 for model/view and about 4 for projective matrix)
Transformation Hierarchy Example 3

Hierarchical Modelling

- advantages
  - define object once, instantiate multiple copies
  - transformation parameters often good control knobs
  - maintain structural constraints if well-designed

- limitations
  - expressivity: not always the best controls
  - can’t do closed kinematic chains
  - keep hand on hip
  - can’t do other constraints
    - collision detection
    - self-intersection
    - walk through walls

Transformation Hierarchy Example 4

Single Parameter: simple

- parameters as functions of other params
  - clock: control all hands with seconds $s$
    - $m = s/60$, $h = m/60$
    - $\theta_s = (2 \pi s) / 60$
    - $\theta_m = (2 \pi m) / 60$
    - $\theta_h = (2 \pi h) / 60$

Single Parameter: complex

- mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk/mechanisms/pages/irregular.html
Display Lists

- precompile/cache block of OpenGL code for reuse
  - usually more efficient than immediate mode
  - exact optimizations depend on driver
- good for multiple instances of same object
  - but cannot change contents, not parametrizable
- good for static objects redrawn often
  - display lists persist across multiple frames
  - interactive graphics: objects redrawn every frame from new viewpoint from moving camera
- can be nested hierarchically
- snowman example
  - http://www.lighthouse3d.com/opengl/displaylists

One Snowman

```c
void drawSnowMan() {
    glColor3f(1.0f, 1.0f, 1.0f); // Draw Body
    glutSolidSphere(0.75f, 20, 20); // Draw Head
    glutSolidSphere(0.25f, 20, 20); // Draw Nose
    glColor3f(1.0f, 0.5f, 0.5f); // Draw Eyes
    glPopMatrix();
}
```

Instantiate Many Snowmen

```c
// Draw 36 Snowmen
for(int i = -3; i < 3; i++)
    for(int j = -3; j < 3; j++) {
        glPushMatrix();
        glTranslatef(i*10.0, 0, j * 10.0);
        glCallList(snowmanDL); // Call the function to draw a snowman
        glPopMatrix();
    }
```

Making Display Lists

```c
GLuint createDL() {
    GLuint snowManDL;
    // Create the id for the list
    snowManDL = glGenLists(1);
    glNewList(snowManDL, GL_COMPILE);
    drawSnowMan();
    glEndList();
    return(snowManDL); }
```

Transforming Normals

```c
36K polygons, 153 FPS
```

One Snowman

```c
// Draw Body
    glutSolidSphere(0.75f, 20, 20);
```

Instantiate Many Snowmen

```c
36K polygons, 55 FPS
```

Making Display Lists

```c
snowmanDL = createDL();
for(int i = -3; i < 3; i++)
    for(int j = -3; j < 3; j++) {
        // Call the function to draw a snowman
        drawSnowMan();
        glEndList();
    }
```

Transforming Normals

```c
36K polygons, 153 FPS
```
Transforming Geometric Objects
- lines, polygons made up of vertices
- just transform the vertices, interpolate between
- does this work for everything? no!

Computing Normals
- polygon:

![Diagram of a triangle with vertices P1, P2, and P3, and a normal vector N]  
\[ \mathbf{N} = (\mathbf{P}_2 - \mathbf{P}_1) \times (\mathbf{P}_3 - \mathbf{P}_1) \]
- assume vertices ordered CCW when viewed from visible side of polygon
- normal for a vertex
  - specify polygon orientation
  - used for lighting
  - supplied by model (i.e., sphere), or computed from neighboring polygons

Transforming Normals
- what is a normal?
  - a direction
  - homogeneous coordinates: w=0 means direction
  - often normalized to unit length
  - vs. points/vectors that are object vertex locations
- what are normals for?
  - specify orientation of polygonal face
  - used when computing lighting
  - so if points transformed by matrix \( \mathbf{M} \), can we just transform normal vector by \( \mathbf{M} \) too?

Transforming Normals
- translations OK: w=0 means unaffected
- rotations OK
- uniform scaling OK
  - these all maintain direction

Transforming Normals
- nonuniform scaling does not work
- x-y=0 plane
  - line x=y
  - normal: [1, -1, 0]
  - direction of line \( x = y \)
  - (ignore normalization for now)

Transforming Normals
- apply nonuniform scale: stretch along x by 2
  - new plane \( x = 2y \)
  - transformed normal: [2, -1, 0]
  - normal is direction of line \( x = -2y \) or \( x + 2y = 0 \)
  - not perpendicular to plane!
  - should be direction of \( 2x = -y \)
Planes and Normals
- plane is all points perpendicular to normal
  - \( N \cdot P = 0 \) (with dot product)
  - \( N^T P = 0 \) (matrix multiply requires transpose)
- explicit form: plane = \( ax + by + cz + d \)

Finding Correct Normal Transform
- transform a plane
  \[
  P \quad N \quad P' = M P \quad N' = Q N
  \]
  \[
  N^T P' = 0
  \]
  \[
  Q N^T (MP) = 0
  \]
  \[
  Q^T M P = 0
  \]
  \[
  Q^T M = I
  \]
  given \( M \), what should \( Q \) be?
  stay perpendicular
  substitute from above
  \[
  (AB)^T = B^T A^T
  \]
  \[
  N^T P = 0 \iff Q^T M = I
  \]
  thus the normal to any surface can be transformed by the inverse transpose of the modelling transformation.

Assignments
- project 1
  - out today, due 11:59pm Wed May 18
  - you should start very soon!
  - build giraffe out of cubes and 4x4 matrices
    - think cartoon, not beauty
  - template code gives you program shell, Makefile
- written homework 1
  - out today, due 4pm Wed May 18
  - theoretical side of material

Real Giraffes

Articulated Giraffe
Articulated Giraffe

Demo

Project 1 Advice
- build then animate one section at a time
  - ensure you’re constructing hierarchy correctly
  - use body as scene graph root
  - start with an upper leg
- consider using separate transforms for animation and modelling
- make sure you redraw exactly and only when necessary

Project 1 Advice
- finish all required parts before going for extra credit
  - playing with lighting or viewing
- ok to use glRotate, glTranslate, glScale
- ok to use glutSolidCube, or build your own
  - where to put origin? your choice
    - center of object, range -.5 to +.5
    - corner of object, range 0 to 1

Project 1 Advice
- visual debugging
  - color cube faces differently
  - colored lines sticking out of glutSolidCube faces
- thinking about transformations
  - move physical objects around
  - play with demos
  - Brown scenegraph applets

Project 1 Advice
- transitions
  - safe to linearly interpolate parameters for glRotate/glTranslate/glScale
  - do not interpolate individual elements of 4x4 matrix!
Labs Reminder

- in CICSR 011
- today 3-4, 4-5
  - Thu labs are for help with programming projects
  - Thursday 11-12 slot deprecated first four weeks
  - Tue labs are for help with written assignments
  - Tuesday 11-12 slot is fine
- no separate materials to be handed in
- after-hours door code