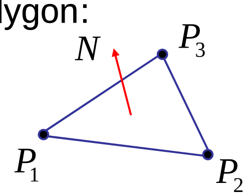
Chapter 4: Transformations- Transforming Normals, Hierarchies and OpenGL, Assignment 2

Transforming Normals

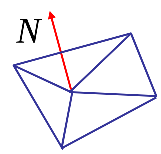
Computing Normals

- polygon:



$$N = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\left\|(P_2 - P_1) \times (P_3 - P_1)\right\|}$$

- assume vertices ordered CCW when viewed from visible side of polygon
- normal for a vertex
  - used for lighting
  - supplied by model (i.e., sphere), or computed from neighboring polygons



Transforming Normals

- What is a normal?
  - **Vector**
    - Orthogonal (perpendicular) to plane/surface
  - Do standard transformations preserve orthogonality?
    - Or angles in general?

Planes and Normals

- Plane - all points where $N \cdot P = 0$

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, N = \begin{bmatrix} A \\ B \\ C \\ 0 \end{bmatrix}$$

- Implicit form

$$Plane = A \cdot x + B \cdot y + C \cdot z + D$$

Finding Correct Normal Transform

- transform a plane

$$P \rightarrow P' = MP$$
$$N \rightarrow N' = QN$$

Given M, find Q

$N'^T P' = 0$ **stay perpendicular**

$(QN)^T (MP) = 0$ **substitute from above**

$N^T Q^T M P = 0$ **$(AB)^T = B^T A^T$**

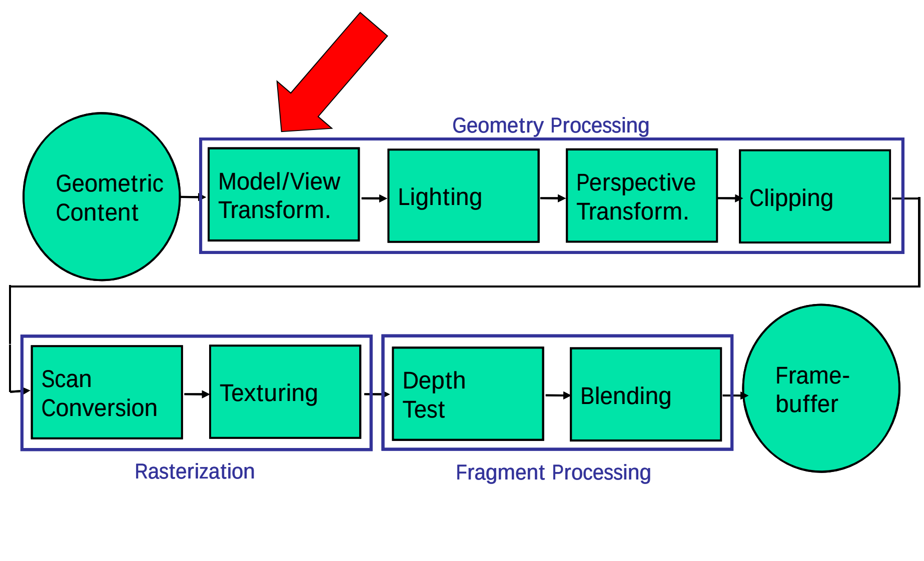
$Q^T M = I$ $N^T P = 0$

$$Q = \left(M^{-1}\right)^T$$

Normal transformed by *transpose* of *the inverse* of the modeling transformation

Transformations in OpenGL

The Rendering Pipeline

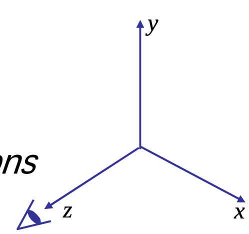


Modeling Transformation

- Purpose:
  - Map geometry from local object coordinate system into a global world coordinate system
  - Same as placing objects
- Transformations:
  - Arbitrary affine transformations are possible
    - More complex transformations may be desirable
      - Freeform deformations
    - Not available in hardware

Viewing Transformation

- Purpose:
  - Map geometry from *world coordinate system* into *camera coordinate system*
  - Camera coordinate system is ***right-handed***, viewing direction is *negative* z-axis
  - Same as placing camera
- Transformations:
  - Usually only *rigid body transformations*
    - Rotations and translations
  - Objects have same size and shape in camera and world coordinates



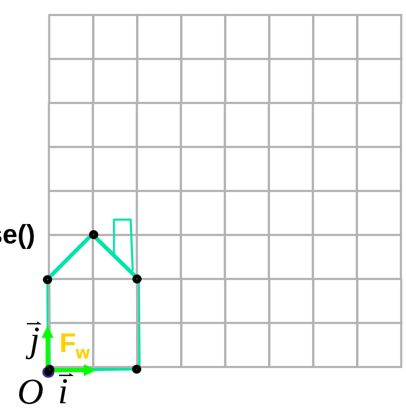
Model/View Transformation

- Combine modeling and viewing transform
  - Combine into single matrix
  - Saves computation time
    - if many points are to be transformed
  - Possible because viewing transformation directly follows modeling transformation without intermediate operations

Transformations in OpenGL

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glBegin(GL_LINE_LOOP);
glVertex2f(0,0);
glVertex2f(2,0);
glVertex2f(2,2);
glVertex2f(1,3);
glVertex2f(0,2);
glEnd();
```

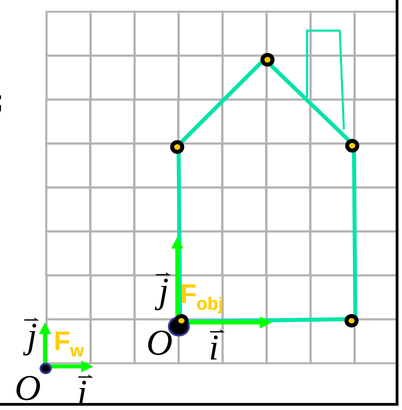
DrawHouse()



Transformations in OpenGL

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_w = \begin{bmatrix} 2 & 0 & 0 & 3 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{obj}$$

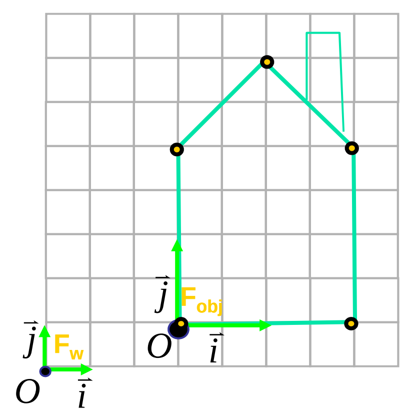
```
GLfloat  T[16] = { 2,0,0,0,  0,2,0,0,
                   0,0,2,0  3,1,0,1};

glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(T);

DrawHouse();
```



Transformations in OpenGL

- An easier way to do the same thing….

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glTranslatef(3,1,0);
glScale(2,2,2);

DrawHouse();
```

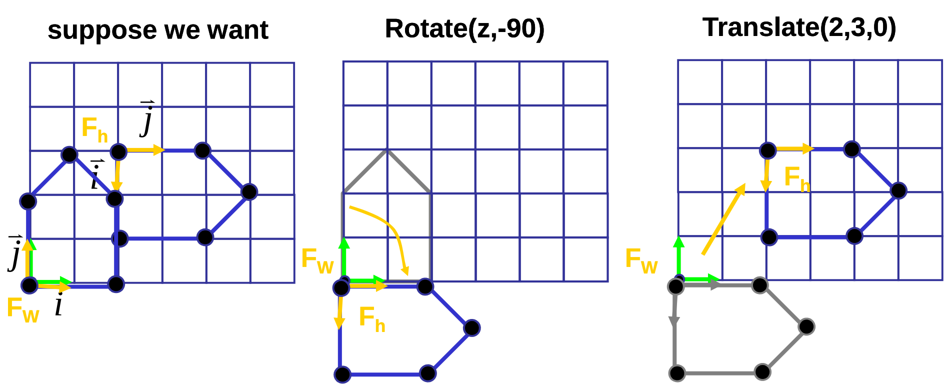


Matrix Operations in OpenGL

- 2 Matrices:
  - Model/view matrix M
  - Projective matrix P
- Example:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity(); // M=Id
glRotatef( angle, x, y, z ); // M= R(α)*Id
glTranslatef( x, y, z ); // M= T(x,y,z)*R(α)*Id
glMatrixMode( GL_PROJECTION );
glRotatef( … ); // P= …
```

Composing Transformations



$P_A = Rot(z,-90)\,P_h$ $\quad P_W = Trans(2,3,0)\,P_A$

$$P_W = Trans(2,3,0)\,Rot(z,-90)\,P_h$$

Composing Transformations

$$P_W = Trans(2,3,0)Rot(z,-90)P_h$$

- R-to-L: interpret operations wrt fixed coords
  - moving object
- L-to-R: interpret operations wrt local coords
  - changing coordinate system
- OpenGL (L-to-R, local coords)

```
glTranslatef(2,3,0);
glRotatef(-90,0,0,1);
DrawHouse();
```

$$M_{MV} = Trans(2,3,0) \cdot M_{MV}$$

$$M_{MV} = Rot(z,-90)M_{MV}$$

updates current transformation matrix by postmultiplying

Post Multiplication

- Composite transformation is now just the product of a few matrixes
- Rather than multiply each point sequentially with 3 matrices, first multiply the matrices, then multiply each point with only one (composite) matrix
  - Much faster for large # of points!
  - Same reason to use homogeneous coordinates

Interpreting Composite OpenGL Transformations

- Example from earlier lectures:
  - Rotation around arbitrary center
  - In OpenGL:

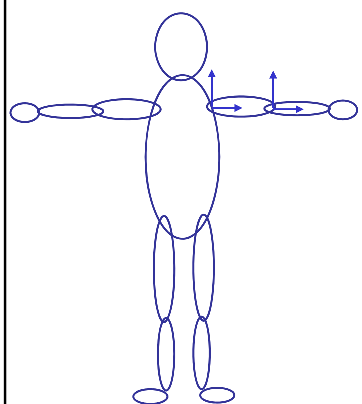
```
// initialization of matrix
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();

glTranslatef( 4, 3 );
glRotatef( 30, 0.0, 0.0, 1.0 );
glTranslatef( -4, -3 );

glBegin( GL_TRIANGLES );
// specify object geometry...
```

Top-to-bottom: transf. of coordinate frame

Bottom-to-top: transf. of object

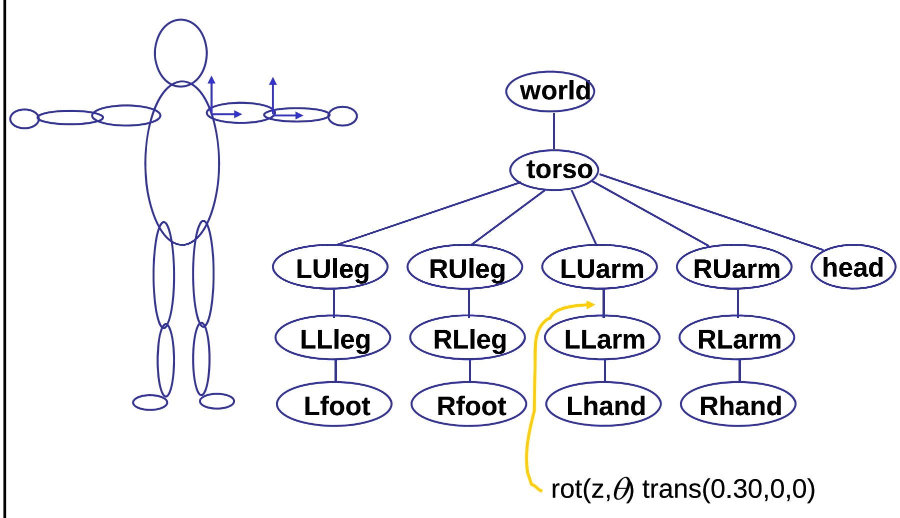
Transformation Hierarchies

Transformation Hierarchies



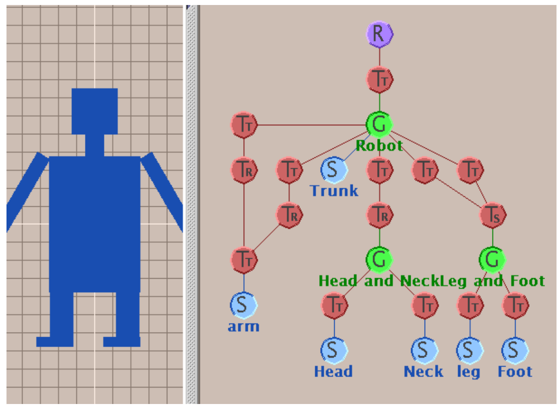
- scene may have a hierarchy of coordinate systems
  - Multiple objects, multiple joint links, ...
  - stores matrix at each level with incremental transform from parent's coordinate system

Transformation Hierarchies



Check out: Brown Applets

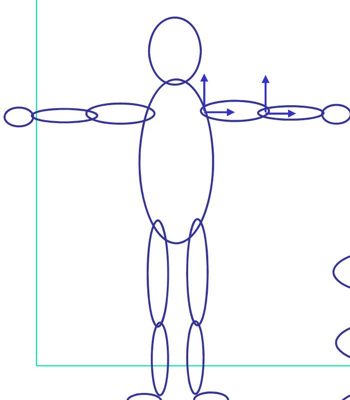
http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html

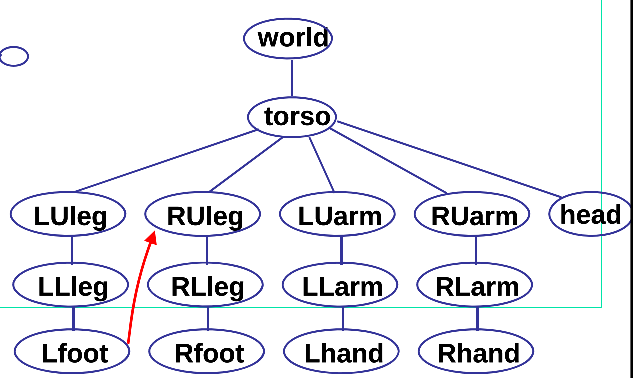


Have a look later

Matrix Stacks

- Avoiding unnecessary computation when incremental processing makes no sense
  - Using inverse to return to origin costs..

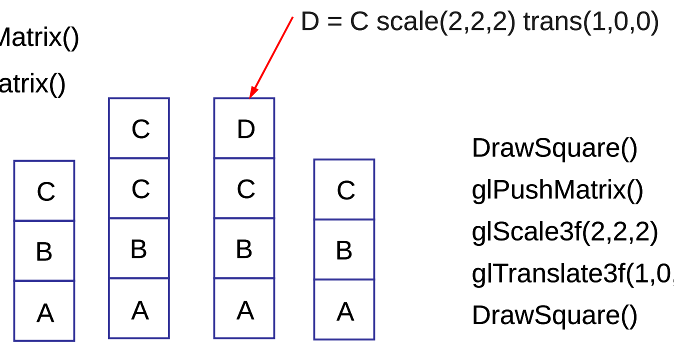




Matrix Stacks

glPushMatrix()

glPopMatrix()

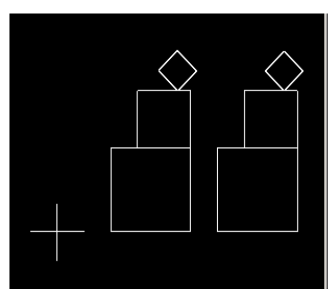


DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()

Modularization

- Drawing a scaled square
  - Push/pop ensures no coord system change

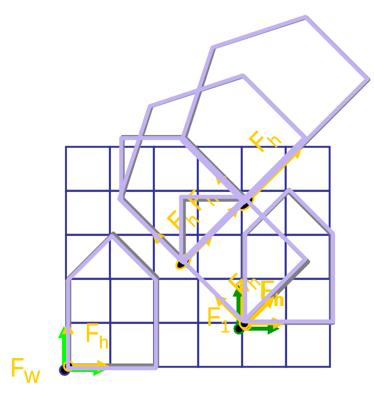
```
void drawBlock(float k) {
  glPushMatrix();

  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

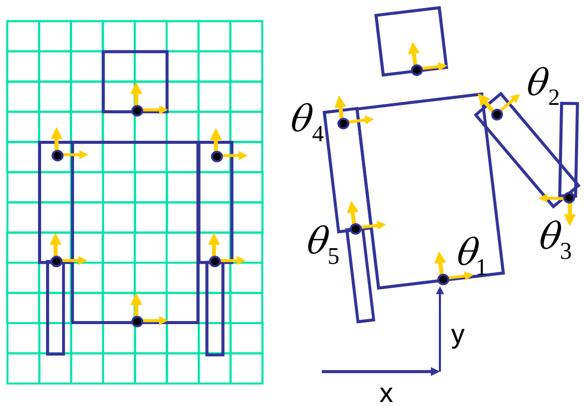
- Advantages
  - No need to compute inverse matrices all the time
  - Modularize changes to pipeline state
  - Avoids incremental changes to coordinate systems
    - Accumulation of numerical errors
- Practical issues
  - In graphics hardware, depth of matrix stacks is limited
    - Typically 16 for model/view and ~4 for projective matrix

Transformation Hierarchy Examples




```
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslate(1,0,0);
glPopMatrix();
```

Transformation HierarchyExamples




```
glTranslate3f(x,y,0);
glRotatef( θ1,0,0,1);
DrawBody();
glPushMatrix();
  glTranslate3f(0,7,0);
  DrawHead();
glPopMatrix();
glPushMatrix();
  glTranslate(2.5,5.5,0);
  glRotatef( θ2,0,0,1);
  DrawUArm();
  glTranslate(0,-3.5,0);
  glRotatef( θ3,0,0,1);
  DrawLArm();
glPopMatrix();
... (draw other arm)
```

Hierarchical Modeling

- 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

Assignment 2

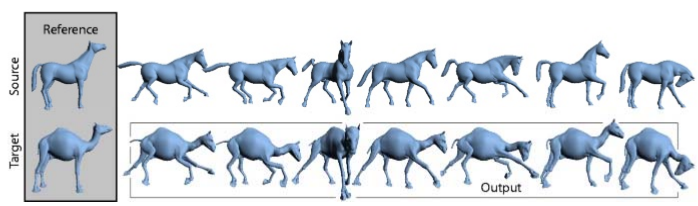
Assignment 2

- Out this week, due **4pm Fri Oct 12, 2012**
  - http://www.ugrad.cs.ubc.ca/~cs314/Vsep2012/a2/programming_a2.pdf
  - Start very soon!
  - Build and animate a robot made out of cubes and 4x4 matrices
    - think cartoon, not beauty
  - Template code - program shell, Makefile
    - http://www.ugrad.cs.ubc.ca/~cs314/Vsep2012/a2/a2.tar.gz

Advice

- **Draw one section at a time**
  - Ensure you're constructing hierarchy correctly
  - Use body as scene graph root
  - Continue with attached parts
- Finish all required parts before...
  - ...Adding extra links or DOFs
  - ...Going for extra credit
- Visual debugging
  - Draw the current coord system

Advanced transformations example



- Deformation Transfer [Sumner'05]
  - Use transformation gradients (transformation without translation) as per-triangle encoding of motion

Advanced transformations example

- Deformation Transfer