General Transformation Commands

- `glMatrixMode()`
  - Mode/view
  - Projection
  - Texture
  - Which matrix will be modified
  - Subsequent transformation commands affect the specified matrix.

- `void glLoadIdentity()`;
  - Sets the currently modifiable matrix to the 4 x 4 identity matrix.
  - Usually done when you first switch matrix mode.

Object Coordinate System

- Used to place objects in scene
  - Draw at origin of WCS
  - Scale and Rotate
  - Translate to final position

- `glMatrixMode(GL_MODELVIEW)`
  - `glScalef(x, y, z)`
  - `glRotatef(angle, x, y, z)`
  - `glTranslatef(x, y, z)`
  - `gluLookAt(eyex, eyey, eyez, x, y, z, upx, upy, upz)`

Transformations

- In OpenGL, transformation are performed in the opposite order they are called

- `glTranslatef(1.0, 1.0, 0.0);`
- `glRotatef(45.0, 0.0, 0.0, 1.0);`
- `glScalef(2.0, 2.0, 0.0);`
- `DrawSquare(0.0, 0.0, 1.0);`

Rotation and Scaling

- Rotation and Scaling is done about origin
  - You always get what you expect
  - Correct on all parts of model

- `glTranslatef(-0.5, -0.5, 0.0);`
- `glScalef(2.0, 2.0, 0.0);`
- `glRotatef(45.0, 0.0, 0.0, 1.0);`
- `DrawSquare(0.0, 0.0, 1.0);`

Load and Mult Matrices

- `void glLoadMatrix(fd)(const TYPE *m);`
  - Sets the sixteen values of the current matrix to those specified by m.

- `void glMultMatrix(fd)(const TYPE *m);`
  - Multiplies the matrix specified by the sixteen values pointed to by m by the current matrix and stores the result as the current matrix.

- OpenGL uses column instead of row vectors
  - Let C be the current matrix and call `glMultMatrix(M)`. After multiplication, the final matrix is always CM.

- Matrices are defined like this (use float m[16]):

```
M = 
  | m0 m1 m2 m3 |
  | m4 m5 m6 m7 |
  | m8 m9 m10 m11|
  | m12 m13 m14 m15|
```
Stack Operations

- `glPushMatrix`
- `glPopMatrix`

Transformations

- Two ways to specify transformations
  - (1) Each part of the object is transformed independently relative to the origin
    - Not the best way!
    - Translate the base by (5,0,0);
    - Translate the lower arm by (5,0,0);
    - Translate the upper arm by (5,0,0);

Relative Transformation

A better (and easier) way:

(2) Relative transformation: Specify the transformation for each object relative to its parent

Object Dependency

- A graphical scene often consists of many small objects
- The attributes of an object (positions, orientations) can depend on others

Hierarchical Representation - Scene Graph

- We can describe the object dependency using a tree structure
  - Root node
    - Base
    - Lower arm
    - Upper arm
    - Hammer
  - Leaf node
  - The position and orientation of an object can be affected by its parent, grand-parent, grand-grand-parent ...
  - This hierarchical representation is referred to as Scene Graph

Relative Transformation

Relative transformation: Specify the transformation for each object relative to its parent

- Step 1: Translate base and its descendants by (5,0,0)
Relative Transformation (2)

Step 2: Rotate the lower arm and all its descendants relative to its local y axis by -90 degree

Relative Transformation (3)

- Represent relative transformations using scene graph

Do it in OpenGL

- Translate base and all its descendants by (5,0,0)
- Rotate the lower arm and its descendants by -90 degree about the locally

A more complicated example

- How about this model?

Depth-first traversal

- Program this transformation by depth-first traversal
How about this?

- Translated (5,0,0)
- Draw base
- Rotated (right hammer)
- Draw right hammer

Something is wrong...

- What’s wrong? We want to transform the right hammer relative to the base, not to the left hammer

Do
- Translate (5,0,0)
- Draw base
- Rotate (75,0,1,0)
- Draw left hammer
- Rotate (-75,0,1,0)
- Draw right hammer

What’s wrong?!

- We should undo the left hammer transformation before we transform the right hammer
- Need to undo this first

Undo the previous transformation(s)

- Need to save the modelview matrix right after we draw base

Initial modelView M
- Translated (5,0,0) -> M = M x T
- Draw base
- Rotated (right hammer)
- Draw right hammer

OpenGL Matrix Stack

- We can use OpenGL Matrix Stack to perform matrix save and restore

Initial modelView M
- Translated (5,0,0) -> M = M x T
- Draw base
- Rotated (right hammer)
- Draw right hammer

Push and Pop Matrix Stack

- A simple OpenGL routine:

  push
  - @Translate(5,0,0)
  - Draw_base();
  - @PushMatrix();
  - @Rotate(75,0,1,0);
  - Draw_left_hammer();
  - @PopMatrix();
  - @Rotate(-75,0,1,0);
  - Draw_right_hammer();

  pop
Hierarchical Transformations

- For geometries with an implicit hierarchy we wish to associate local frames with sub-objects in the assembly.
- Parent-child frames are related via a transformation.
- Transformation linkage is described by a tree.
- Each node has its own local coordinate system.

Opengl® Implementation

```c
#pragma mark - GL_MODELVIEW
glLoadIdentity();
glTranslatef(tx, ty, tz);
create_base();
glTranslatef(0, j1y, 0);
create_joint1();
glTranslatef(0, mag, 0);
create_upward();
glTranslatef(i2y, 0);
create_joint2();
glTranslatef(0, lay, 0);
create_lowerarm();
glTranslatef(0, pp, 0);
create_pointer();
glTranslatef(0, pointer, orientation);
create_pointer();
```

Hierarchical Transformations

- The previous example had simple one-to-one parent-child linkages.
- In general there may be many child frames derived from a single parent frame.
- We need some mechanism to remember the parent frame and return to it when creating new children.
- OpenGL provides a matrix stack for just this purpose:
  - `glPushMatrix()` saves the CTM
  - `glPopMatrix()` returns to the last saved CTM

Each finger is a child of the parent (wrist)
⇒ independent control over the orientation of the fingers relative to the wrist
Hierarchical Transformations

```cpp
// do finger 1
glTranslatef(0, -fy3, 0);
glRotatef(-upperfinger1_orientation);
glTranslatef(0, -fy2, 0);
glTranslatef(0, -fy1, 0);
glRotatef(-lowerfinger1_orientation);
glTranslatef(xf, -fy0, 0);

// do finger 2
Save the matrix state
... // code...

// Restore the matrix state
... // code...
```