Hierarchical Modelling

- advantages
  - define object once, instantiate multiple copies
  - maintain structural constraints if well-designed

- limitations
  - expressivity: not always the best controls
  - can’t do closed kinematic chains
  - can’t do other constraints
  - collision detection
  - walk through walls

Modularization

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

Matrix Stacks

- advantages
  - no need to compute inverse matrices all the time
  - modularizes 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)

Transformations

- all scene graph parts would be on top of each other if translation set to 0 everywhere
- composition of transformations can be surprising and tricky even with just a few simple building blocks
- negative scale is a reflection

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

University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2008
Tamara Munzner

Transformations IV
Week 3, Wed Jan 23

Readings for Jan 16-25

- FCG Chap 6 Transformation Matrices
  - except 6.5, 6.3.1
- FCG Sect 13.3 Scene Graphs
- RB Chap Viewing
  - Viewing and Modeling Transforms until Viewing Transformations
  - Examples of Composing Several Transformations through Building an Articulated Robot Arm
- RB Chap Homogeneous Coordinates and Transformation Matrices
  - until Perspective Projection
- RB Chap Display Lists

Review: 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

Review: Arbitrary Rotation

- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems XYZ and ABC
  - transformation from one to the other is matrix R whose columns are ABC

Matrix Stacks

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

Matrix Stacks

- drawBlock
  - push/pop ensures no coord system change

Matrix Stacks

- glPushMatrix()
- glPopMatrix()

Example:

```c
void drawSnowMan()
{
    glPushMatrix();
    glScalef(2,2,2);
    glTranslatef(-1,-1,-1);
    // Draw Snowman
    glPopMatrix();
}
```
Planes and Normals
- plane is all points perpendicular to normal $N \cdot P = 0$ (with dot product)
- $N^T \cdot P = 0$ (matrix multiply requires transpose)

Explicit form: plane $= ax + by + cz + d$

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);
}
```

Computing Normals
- normal
  - direction specifying orientation of polygon
  - $w=0$ means direction with homogeneous coords
  - vs. $w=1$ for points/vectors of object vertices
  - used for lighting
  - must be normalized to unit length
  - can compute if not supplied with object

Transforming Normals
- so if points transformed by matrix $M$, can we just transform normal vector by $M$ too?
  - translations OK: $w=0$ means unaffected
  - rotations OK
  - uniform scaling OK
  - these all maintain direction

Transforming Geometric Objects
- lines, polygons made up of vertices
- transform the vertices
- interpolate between
- does this work for everything? no!
  - normals are trickier

Planes and Normals
- plane is all points perpendicular to normal
  - $N \cdot P = 0$ (with dot product)
  - $N^T \cdot P = 0$ (matrix multiply requires transpose)

Finding Correct Normal Transform
- transform a plane
  - $P$
  - $N = \begin{bmatrix} a & b & c & d \end{bmatrix}$
  - $N^T \cdot P = 0$
  - $(QN)^T(MP) = 0$
  - $N^T Q^T M P = 0$
  - $Q^T M = I$

Thus the normal to any surface can be transformed by the inverse transpose of the modeling transformation

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 $2x - y$

Transforming Geometric Objects
- lines, polygons made up of vertices
- transform the vertices
- interpolate between
- does this work for everything? no!
  - normals are trickier