Readings for Jan 15-22

- FCG Chap 6 Transformation Matrices
  - except 6.1.6, 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 Appendix Homogeneous Coordinates and Transformation Matrices
  - until Perspective Projection
- RB Chap Display Lists

Matrix Stacks

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

Demo: Brown Applets

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

Modularization

- 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)

Correction/More: Arbitrary Rotation

- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems $XYZ$ and $ABC$
    - it's location in the $XYZ$ coordinate system is $(x_1, y_1, z_1, 1)$...
    - transformation from one to the other is matrix $R$ whose columns are $A, B, C$. C:

$$R(X) = \begin{bmatrix}
    a_1 & a_2 & a_3 & 0 \\
    b_1 & b_2 & b_3 & 0 \\
    c_1 & c_2 & c_3 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} = (x_1, y_1, z_1, 1)$$

Transformation Hierarchy Example 3

- matrix stacks
  - draw same 3D data with different transformations: instancing

Transformation Hierarchy Example 4

- matrix stacks
  - $D = C$ scale($2.2, 2$) translate($1, 0$)
  - $\mathbf{g}(\mathbf{PushMatrix})$
  - $\mathbf{g}(\mathbf{PopMatrix})$
  - $\mathbf{g}(\mathbf{Translate}(\mathbf{3}, 0, 0, 0))$
  - $\mathbf{g}(\mathbf{Rotate}(\mathbf{45}, 0, 0, 1))$
  - $\mathbf{g}(\mathbf{Scale}(\mathbf{2}, 1, 1))$
  - $\mathbf{g}(\mathbf{Identity})$
  - $\mathbf{g}(\mathbf{PopMatrix})$

Demo: Brown Applets

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

### 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](http://www.lighthouse3d.com/opengl/displaylists)

### Making Display Lists
```c
GLint createDL() {
  GLList snowManDL;
  // Create the id for the list
  snowManDL = glGenLists(1);
  glEndList(snowManDL, GL_COMPILE);
  drawSnowMan();
  glEndList();
  return(snowManDL);
}
```

### 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 good for?
  - specify orientation of polygonal face
  - used when computing lighting
- if so points transformed by matrix M, can we just transform normal vector by M too?

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

### Computing Normals
- **polygon:**
  - 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 Geometric Objects
- **lines, polygons made up of vertices**
  - just transform the vertices, interpolate between
  - does this work for everything? no!

### Transforming Normals
- **nonuniform scaling does not work**
  - line \( x' \parallel y' \)
  - 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 \( 2x = -y \)

### 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](http://www.lighthouse3d.com/opengl/displaylists)
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)

\[
N = \begin{bmatrix} a & b & c & d \end{bmatrix} \quad P = \begin{bmatrix} x & y & z \end{bmatrix}
\]

• explicit form: plane = \( ax + by + cz + d \)

Finding Correct Normal Transform
• transform a plane

\[
P \quad \rightarrow \quad P' = MP
\]

\[
N^T P = 0 \quad \rightarrow \quad N^T = QN
\]

thus the normal to any surface can be transformed by the inverse transpose of the modelling transformation

Assignments
• project 1
  • out today, due 5:59pm Fri Feb 2
  • you should start very soon!
  • build armadillo out of cubes and 4x4 matrices
    • think cartoon, not beauty
  • template code gives you program shell, Makefile
    • http://www.ugrad.cs.ubc.ca/~cs314/ijan2007/p1.tar.gz
  • written homework 1
    • out today, due 3pm Fri Feb 2
    • theoretical side of material

Real Armadillos

Articulated Armadillo

Articulated Armadillo

More Fun With Boxes and Matrices:

Lemurs!

Giraffes!

Giraffes!

Kangaroos!

Demo

Project 1 Advice
• do not model everything first and only then worry about animating
• interleave modelling, animation
• add body part, then animate it
• discover if on wrong track sooner
• dependencies: can’t get anim credit if no model
• use middle body as scene graph root
• check from all camera angles

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
Graphical File Comparison
- installed on lab machines
  - xdiff4 (side by side comparison)
  - xwdiff (in-place, with crossouts)
- Windows: windiff
- Macs: FileMerge
  - in /Developer/Applications/Utilities

Style
- you can lose up to 15% for poor style
- most critical: reasonable structure
  - yes: parametrized functions
  - no: cut-and-paste with slight changes
- reasonable names (variables, functions)
- adequate commenting
  - rule of thumb: what if you had to fix a bug two years from now?
- global variables are indeed acceptable

Version Control
- bad idea: just keep changing same file
- save off versions often
  - after got one thing to work, before you try starting something else
  - just before you do something drastic
- how?
  - not good: commenting out big blocks of code
  - a little better: save off file under new name
    - p1.almostworks.cpp, p1.fixedbug.cpp
  - much better: use version control software
    - strongly recommended

Version Control Software
- easy to browse previous work
- easy to revert if needed
- for maximum benefit, use meaningful comments to describe what you did
  - "started on tail", "fixed head breakoff bug", "leg code compiles but doesn't run"
- useful when you're working alone
- critical when you're working together
- many choices: RCS, CVS, subversion
  - RCS is a good place to start
    - easy to use, installed on lab machines

RCS Basics
- setup, just do once in a directory
  - mkdir RCS
- checkin
  - ci –u p1.cpp
- checkout
  - co –l p1.cpp
- see history
  - rcs log p1.cpp
- compare to previous version
  - rcsdiff p1.cpp
- checkout old version to stdout
  - co –p1.5 p1.cpp > p1.cpp.5

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

Tail Wag Frame 0
- first: jump cut from old to new position
- all change happens in single frame
- do last: add smooth transition
- change happens gradually over 30 frames
- key click triggers animation loop
  - explicitly redraw 30 times
  - linear interpolation:
    - each time, param += (new-old)/30
  - example: 5-frame transition

Tail Wag Frame 1
- transitions
- safe to linearly interpolate parameters for glRotate/glTranslate/glScale
- do not interpolate individual elements of 4x4 matrix!

Tail Wag Frame 2
- transitions
- safe to linearly interpolate parameters for glRotate/glTranslate/glScale
- do not interpolate individual elements of 4x4 matrix!

Tail Wag Frame 3
- decide what to do in your next transition
- before we did it, we tested

Tail Wag Frame 4
- decide what to do in your next transition
- before we did it, we tested

Tail Wag Frame 5
- transitions
- safe to linearly interpolate parameters for glRotate/glTranslate/glScale
- do not interpolate individual elements of 4x4 matrix!