



University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2007

Tamara Munzner

Transformations V

Week 3, Wed Jan 24

<http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007>

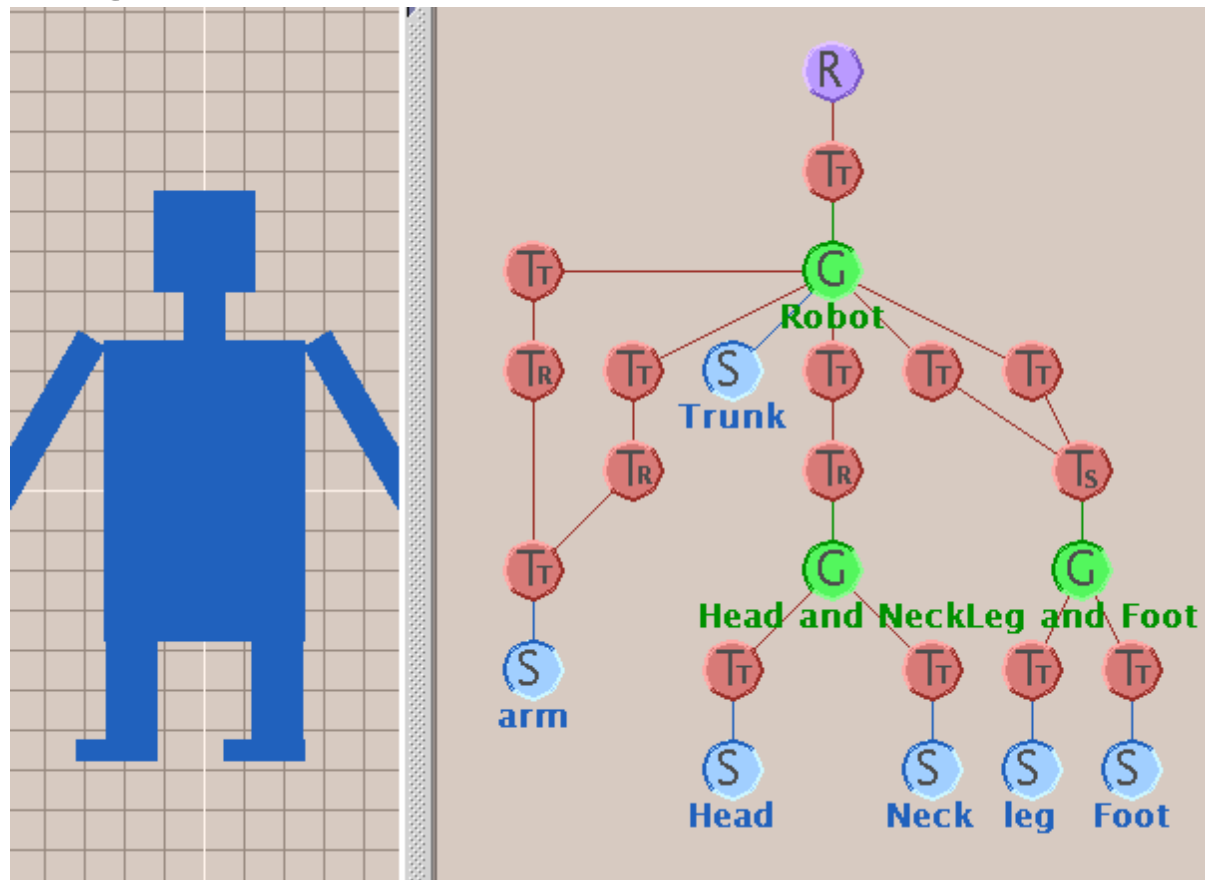
Reading for Next 3 Lectures

- FCG Chapter 7 Viewing
- FCG Section 6.3.1 Windowing Transforms

- RB rest of Chap Viewing
- RB rest of App Homogeneous Coords

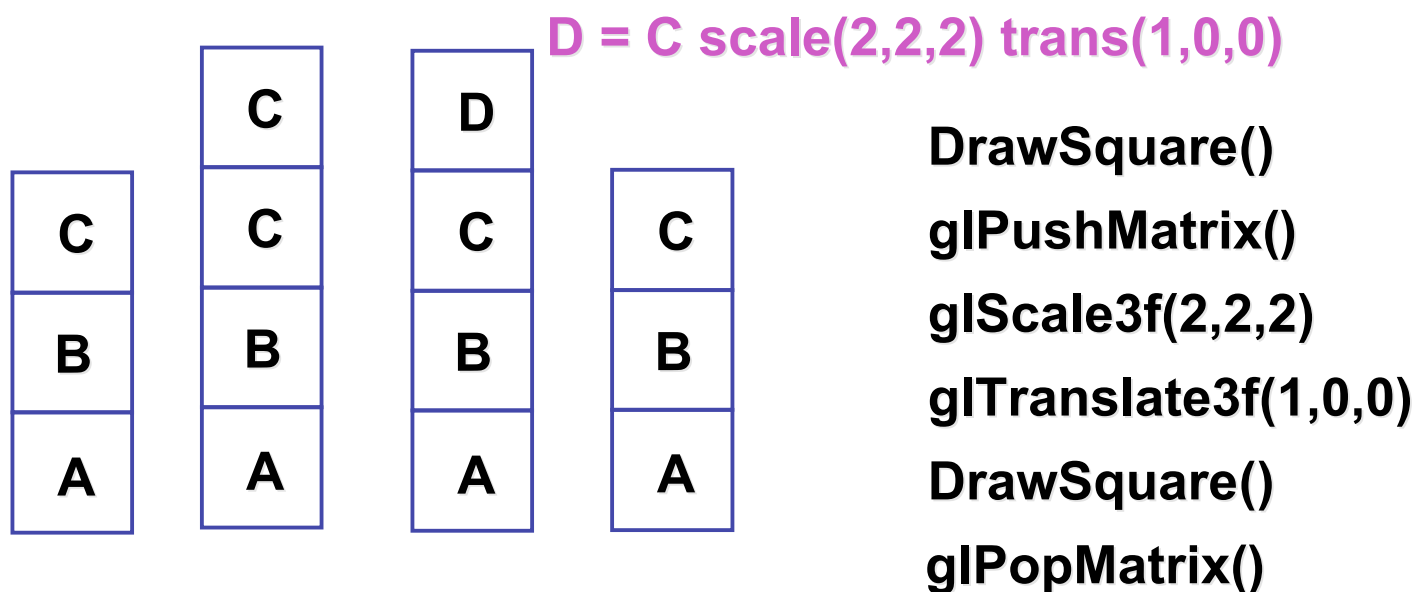
Review: Transformation Hierarchies

- transforms apply to graph nodes beneath them
- design structure so that object doesn't fall apart
- instancing



Review: Matrix Stacks

- OpenGL matrix calls postmultiply matrix M onto current matrix P, overwrite it to be PM
 - or can save intermediate states with stack
 - no need to compute inverse matrices all the time
 - modularize changes to pipeline state
 - avoids accumulation of numerical errors



Assignments

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/Vjan2007/p1.tar.gz>
- written homework 1
 - out today, due 3pm Fri Feb 2
 - theoretical side of material

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

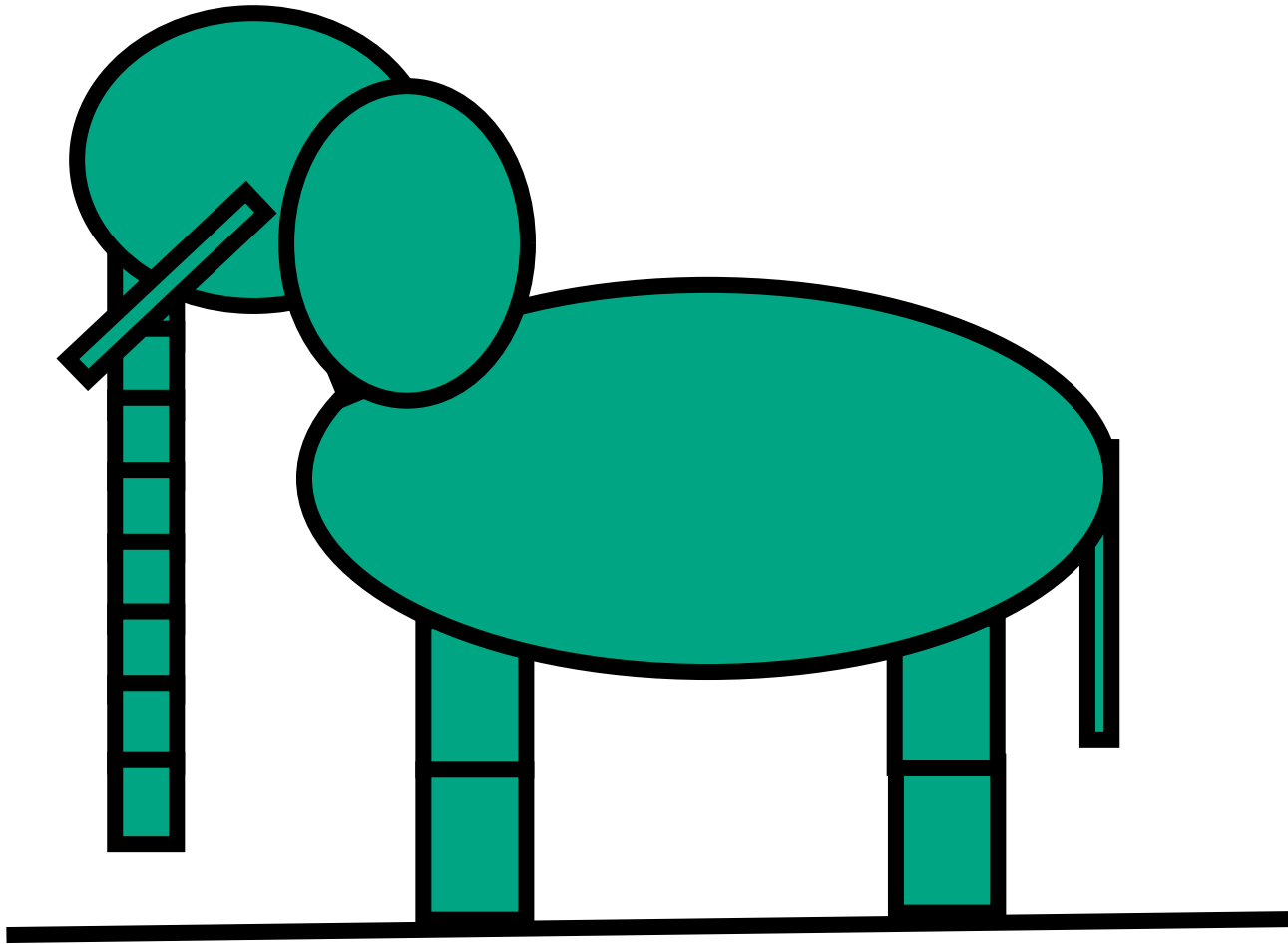
Project 1 Advice

- 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

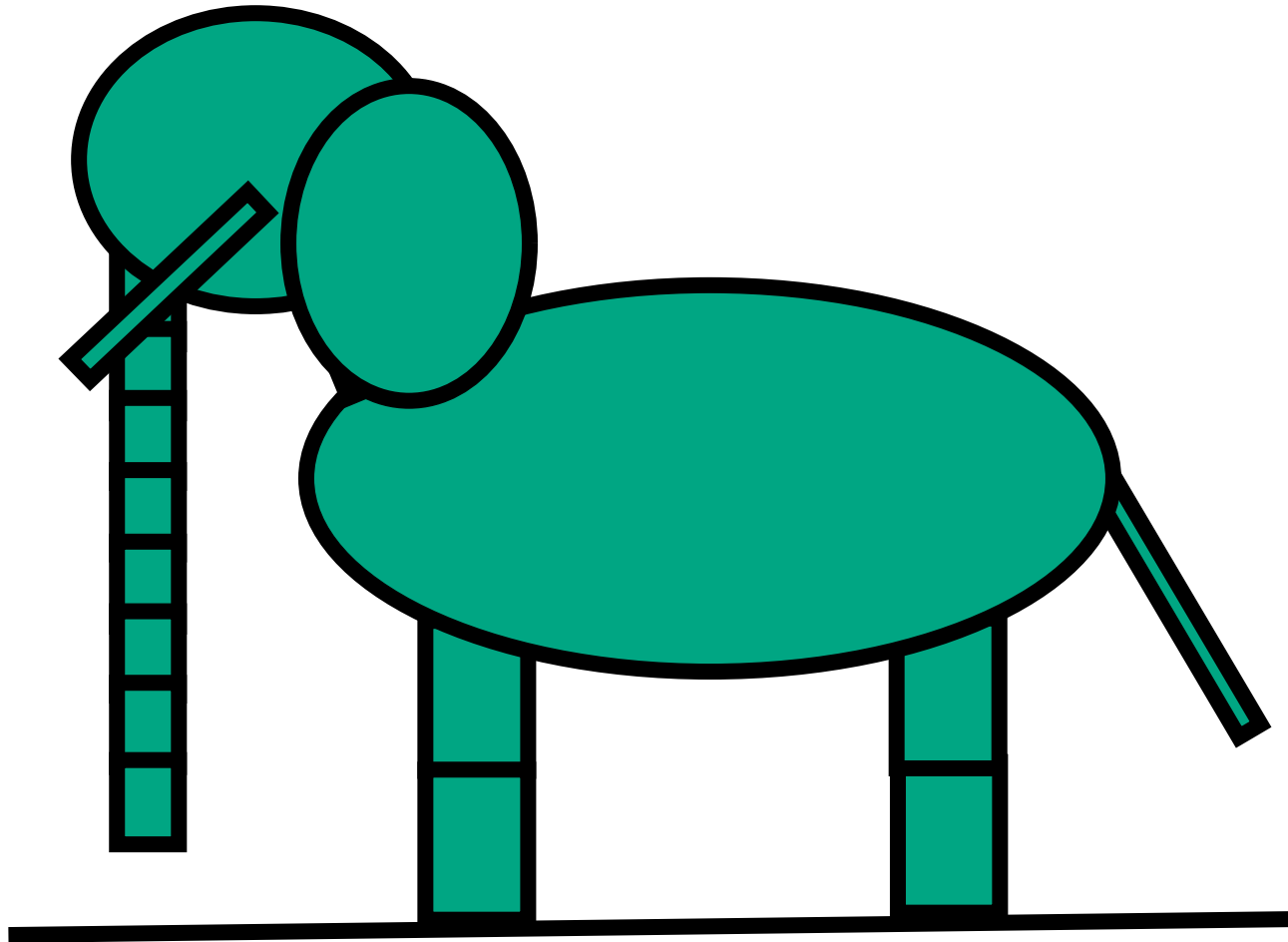
Project 1 Advice

- 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, $\text{param} += (\text{new-old})/30$
 - example: 5-frame transition

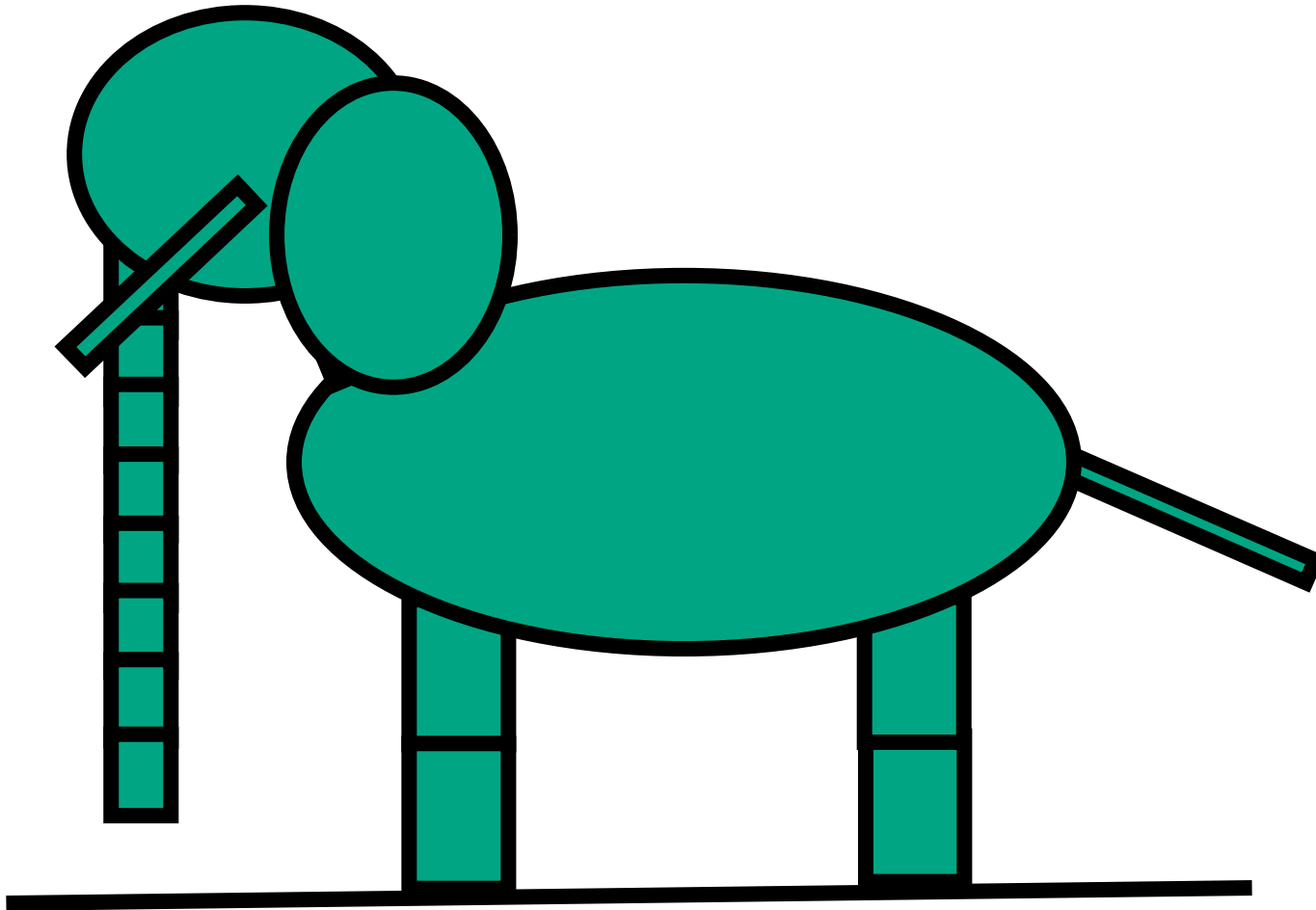
Tail Wag Frame 0



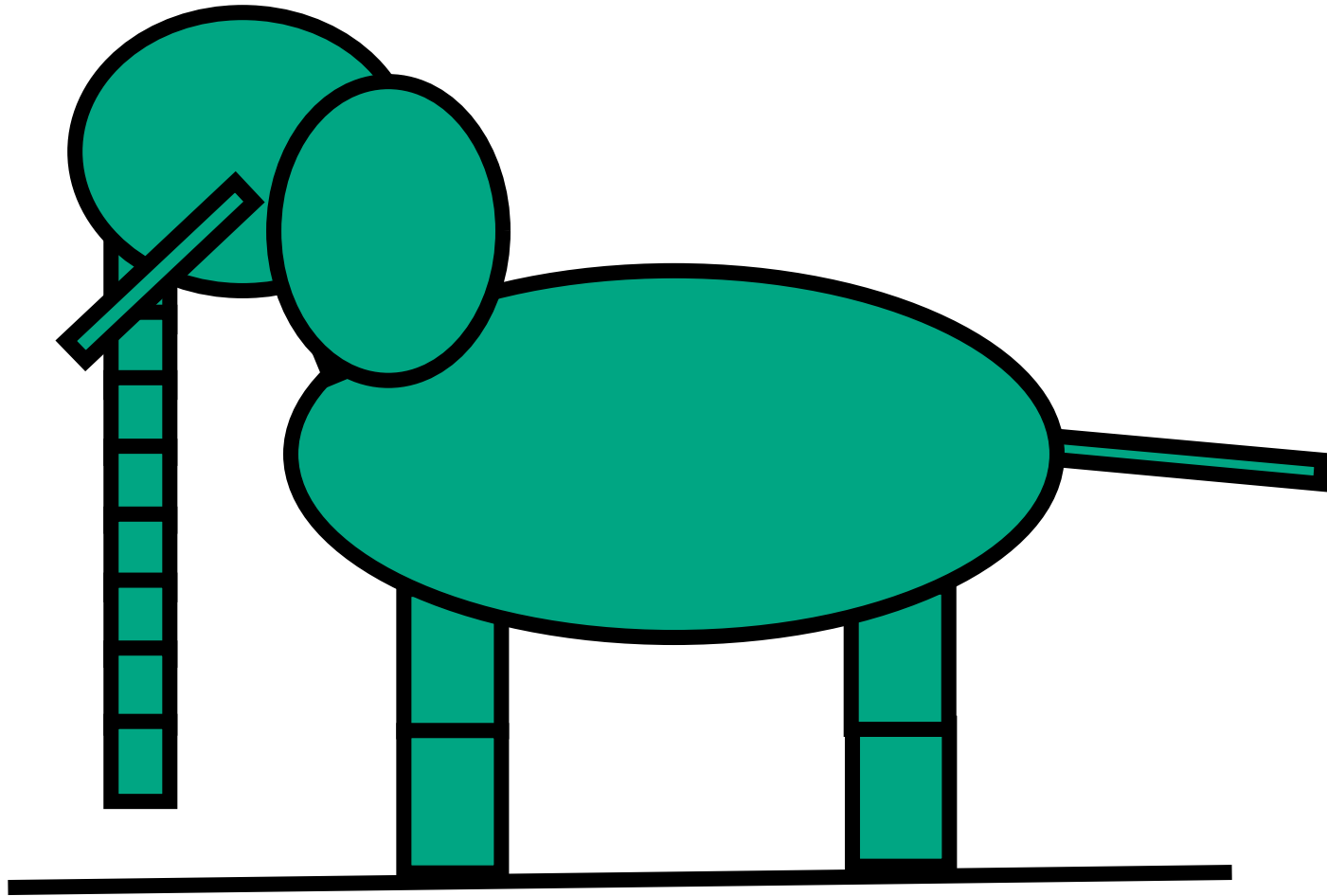
Tail Wag Frame 1



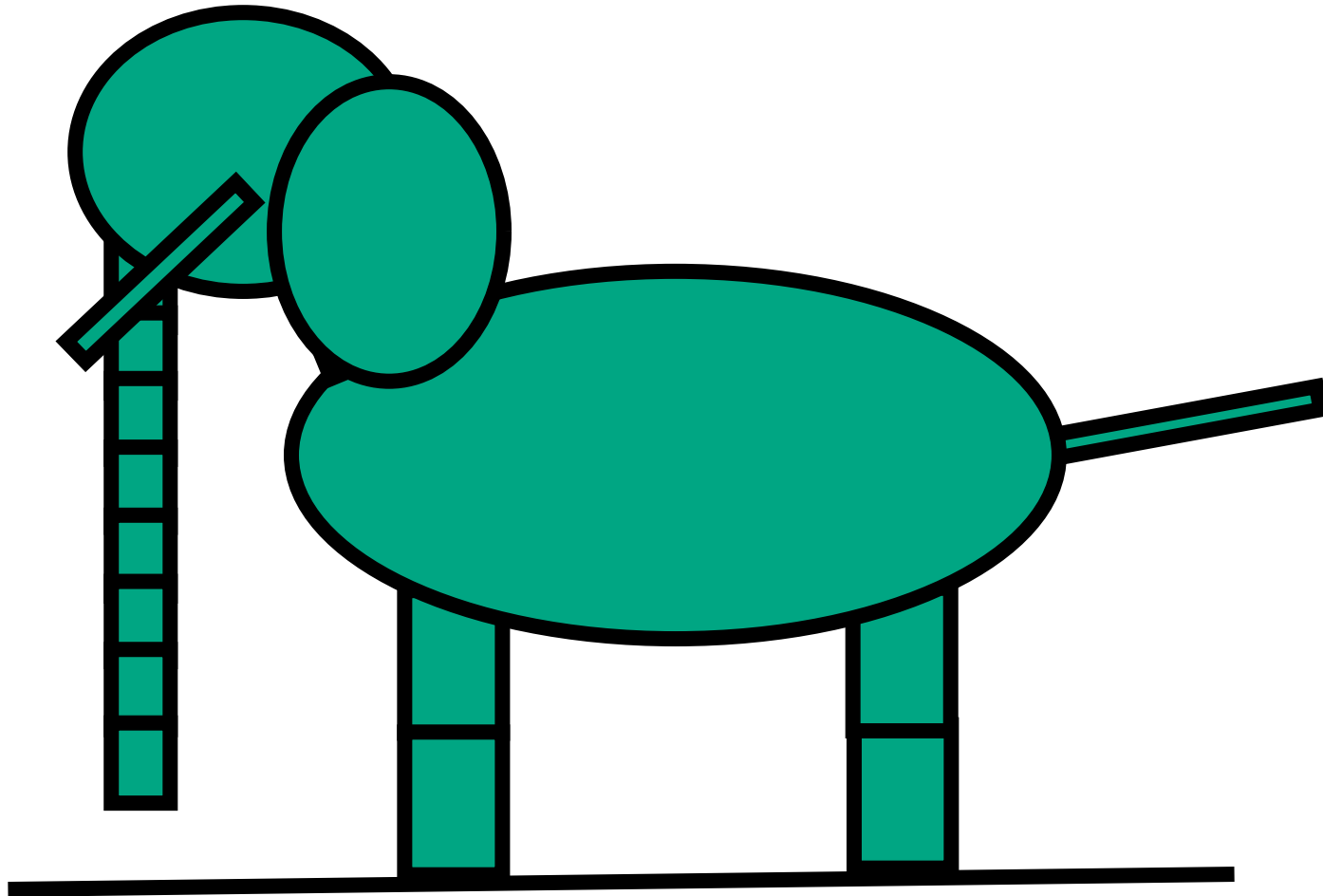
Tail Wag Frame 2



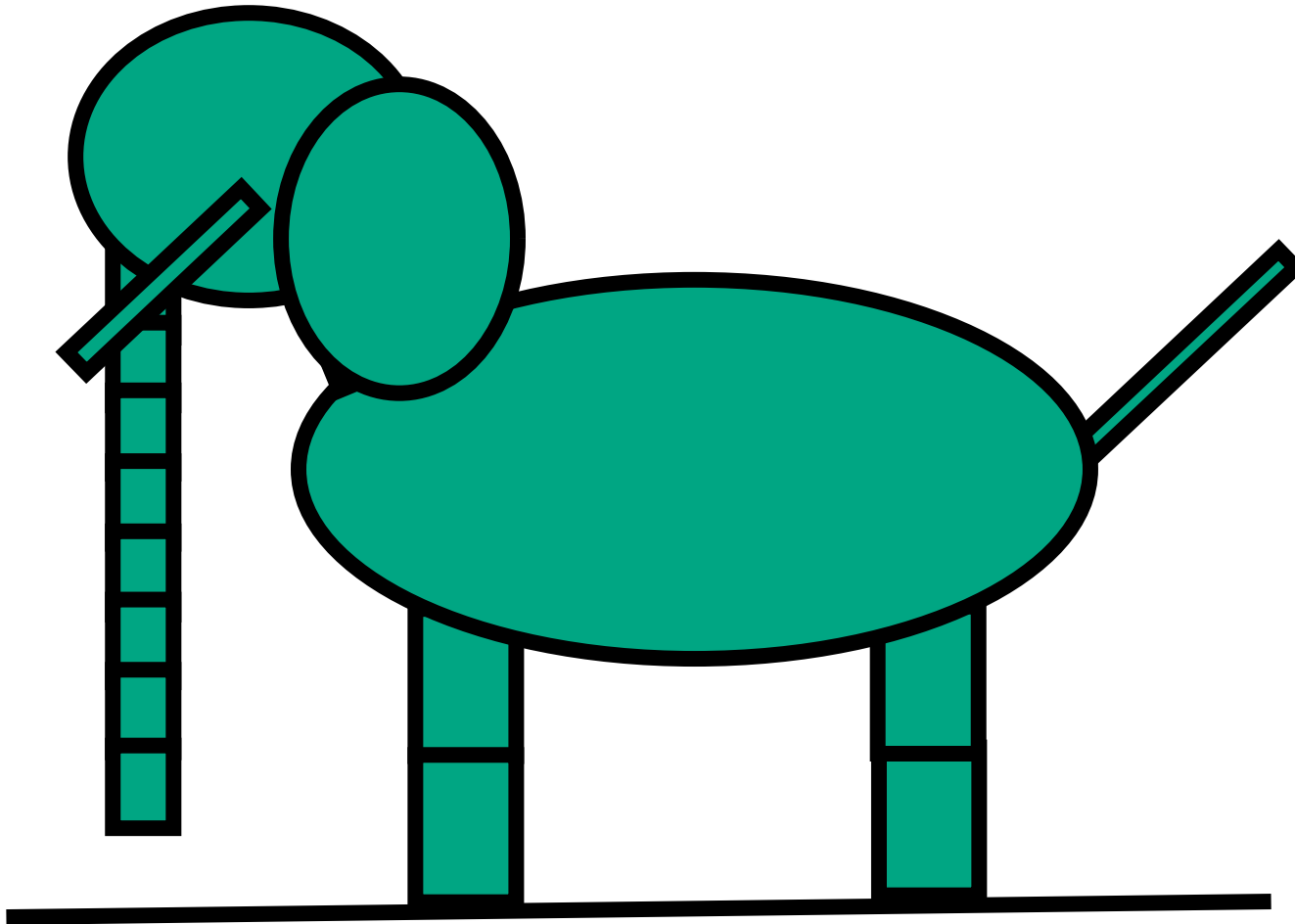
Tail Wag Frame 3



Tail Wag Frame 4



Tail Wag Frame 5



Project 1 Advice

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

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

Graphical File Comparison

- installed on lab machines
 - xfdiff4 (side by side comparison)
 - xwdiff (in-place, with crossouts)
- Windows: windiff
 - <http://keithdevens.com/files/windiff>
- Macs: FileMerge
 - in /Developer/Applications/Utilities

Display Lists

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>

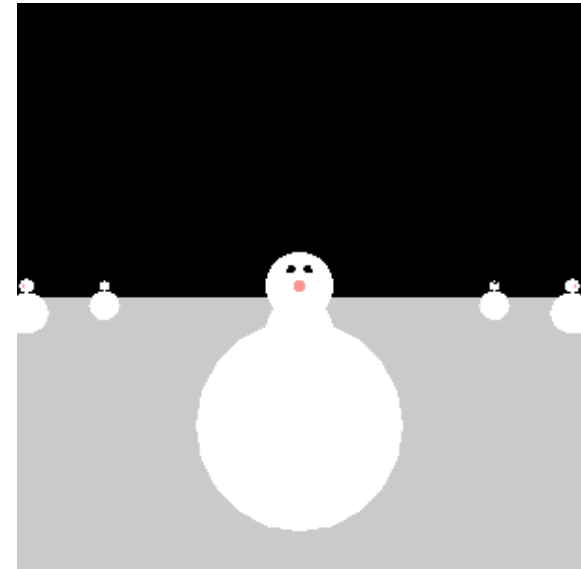
One Snowman



```
void drawSnowMan() {  
  
glColor3f(1.0f, 1.0f, 1.0f);  
  
// Draw Body  
glTranslatef(0.0f, 0.75f, 0.0f);  
glutSolidSphere(0.75f, 20, 20);  
  
// Draw Head  
glTranslatef(0.0f, 1.0f, 0.0f);  
glutSolidSphere(0.25f, 20, 20);  
  
// Draw Eyes  
glPushMatrix();  
glColor3f(0.0f, 0.0f, 0.0f);  
glTranslatef(0.05f, 0.10f, 0.18f);  
glutSolidSphere(0.05f, 10, 10);  
glTranslatef(-0.1f, 0.0f, 0.0f);  
glutSolidSphere(0.05f, 10, 10);  
glPopMatrix();  
  
// Draw Nose  
glColor3f(1.0f, 0.5f, 0.5f);  
glRotatef(0.0f, 1.0f, 0.0f, 0.0f);  
glutSolidCone(0.08f, 0.5f, 10, 2);  
}
```

Instantiate Many Snowmen

```
// 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);  
        // Call the function to draw a snowman  
        drawSnowMan();  
        glPopMatrix();  
    }
```



36K polygons, 55 FPS

Making Display Lists

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

```
snowmanDL = createDL();  
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);  
        glPopMatrix(); }
```

36K polygons, 153 FPS ²⁷

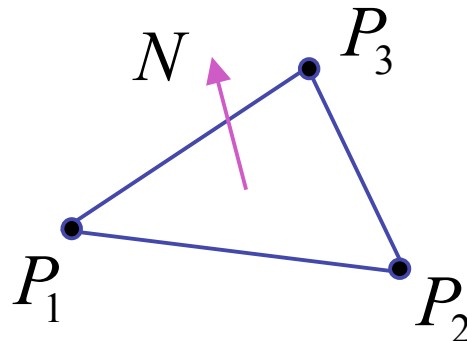
Transforming Normals

Transforming Geometric Objects

- lines, polygons made up of vertices
- just transform the vertices, interpolate between
- does this work for everything? no!

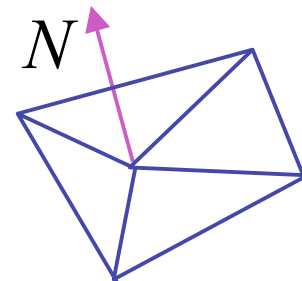
Computing Normals

- polygon:



$$N = (P_2 - P_1) \times (P_3 - P_1)$$

- 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 **M**, can we just transform normal vector by **M** too?

Transforming Normals

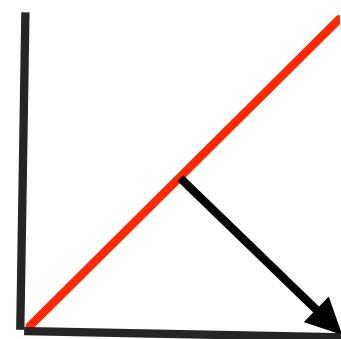
$$\begin{bmatrix} x' \\ y' \\ z' \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

- translations OK: $w=0$ means unaffected
- rotations OK
- uniform scaling OK

- these all maintain direction

Transforming Normals

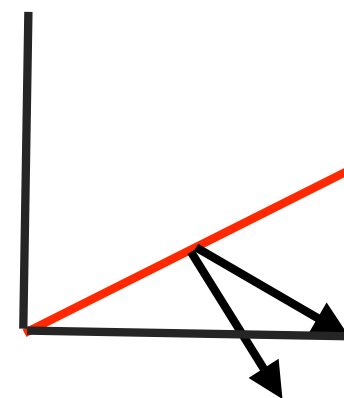
- nonuniform scaling does not work
- $x-y=0$ plane
 - line $x=y$
 - normal: $[1,-1,0]$
 - direction of line $x=-y$
 - (ignore normalization for now)



Transforming Normals

- apply nonuniform scale: stretch along x by 2
 - new plane $x = 2y$
- transformed normal: $[2, -1, 0]$

$$\begin{bmatrix} 2 \\ -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$



- 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)

$$N = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, P = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- explicit form: plane = $ax + by + cz + d$

Finding Correct Normal Transform

- transform a plane

$$\begin{array}{c} P \\ N \end{array} \longrightarrow \begin{array}{c} P' = MP \\ N' = QN \end{array}$$

given M ,
what should Q be?

$$N'^T P' = 0$$

stay perpendicular

$$(QN)^T (MP) = 0$$

substitute from above

$$N^T \underbrace{Q^T MP}_{MP} = 0$$

$$(AB)^T = B^T A^T$$

$$Q^T M = I$$

$$N^T P = 0 \text{ if } Q^T M = I$$

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

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