University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2010

Tamara Munzner

Transformations IV

Week 3, Wed Jan 20

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2010

Assignments

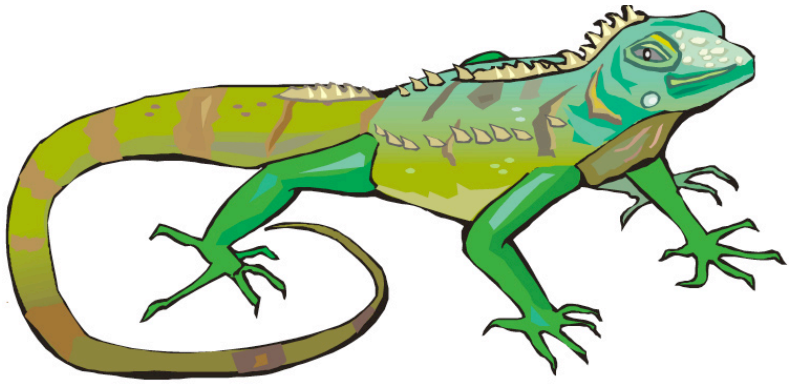
Correction: Assignments

- project 1
  - out today, due 5pm sharp Fri Jan 29
    - projects will go out before we've covered all the material
      - so you can think about it before diving in
  - template code gives you program shell and build tools
    - now out: separate packages for Linux, Mac, Windows
    - see http://www.ugrad.cs.ubc.ca/~cs314/Vjan2010/#assign
      - p1_template_linux.tar.gz
      - p1_template_mac.tar.gz
      - p1_template_win.zip
- written homework 1
  - out today, due 5pm sharp Fri Jan 29
  - theoretical side of material

Demo

- animal out of boxes and matrices

Real Iguanas

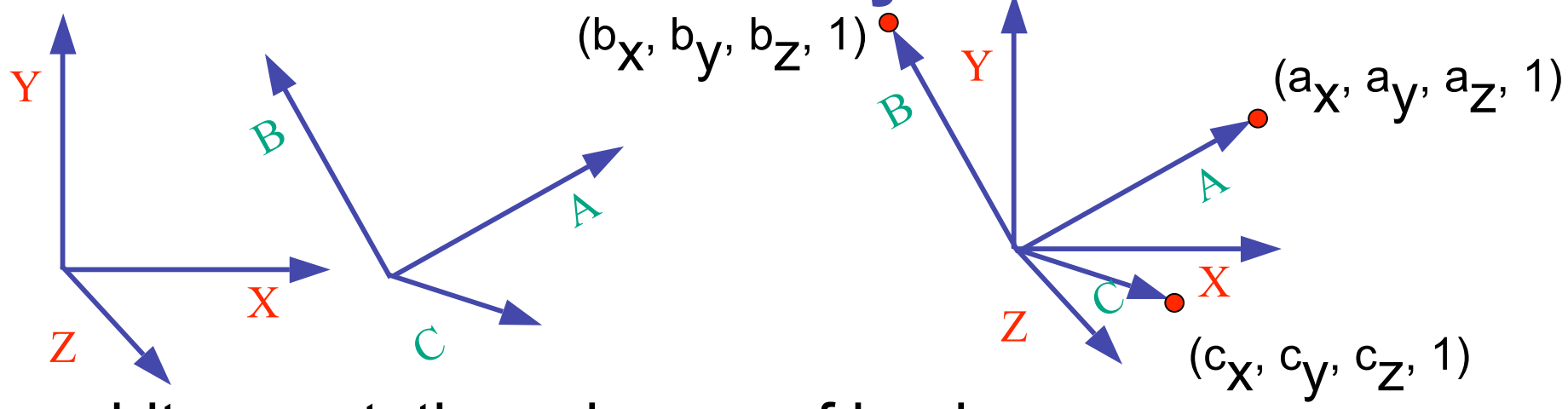


http://funkman.org/animal/reptile/iguana1.jpg

http://www.naturephoto-cz.com/photos/sevcik/green-iguana--iguana-iguana-1.jpg

http://www.mccullagh.org/db9/d30-3/iguana-closeup.jpg

Review: Arbitrary Rotation

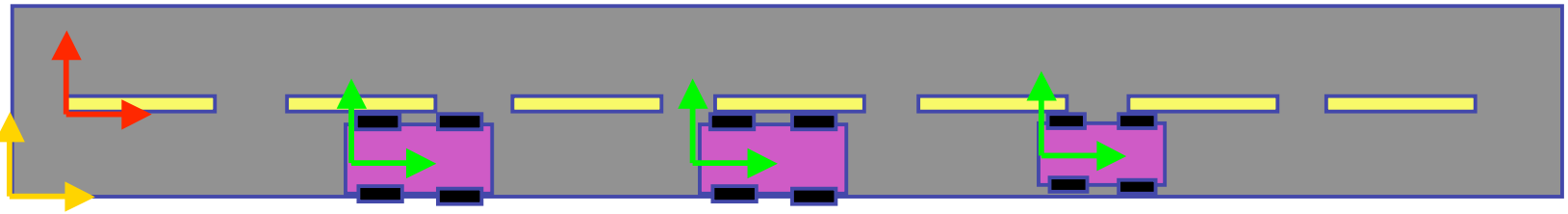


- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems *XYZ* and *ABC*
    - *A*'s location in the XYZ coordinate system is (a_x, a_y, a_z, 1), ...
- transformation from one to the other is matrix R whose columns are *A,B,C:*

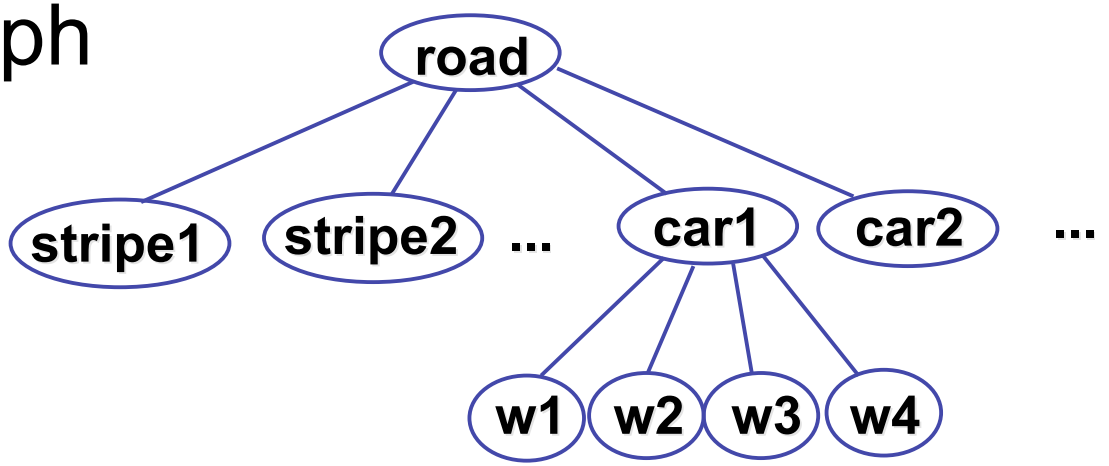
$$R(X) = \begin{bmatrix} a_x & b_x & c_x & 0 \\ a_y & b_y & c_y & 0 \\ a_z & b_z & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = (a_x, a_y, a_z, 1) = A$$

Review: Transformation Hierarchies

- scene may have a hierarchy of coordinate systems
  - stores matrix at each level with incremental transform from parent's coordinate system



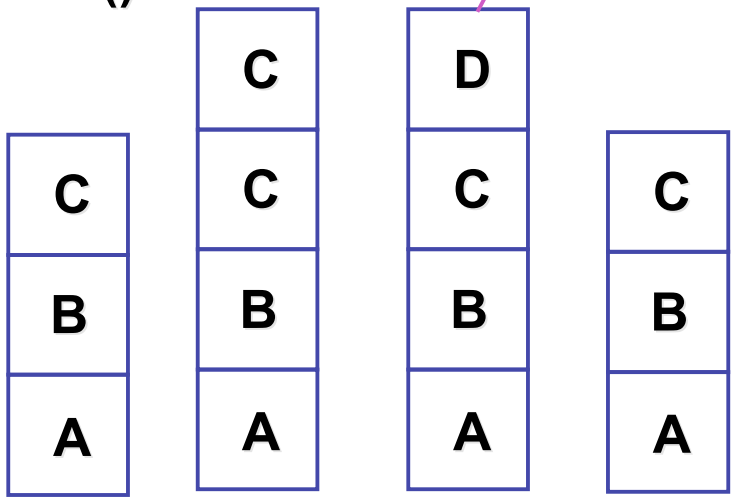
- scene graph



Review: Matrix Stacks

glPushMatrix()

glPopMatrix()



DrawSquare()

glPushMatrix()

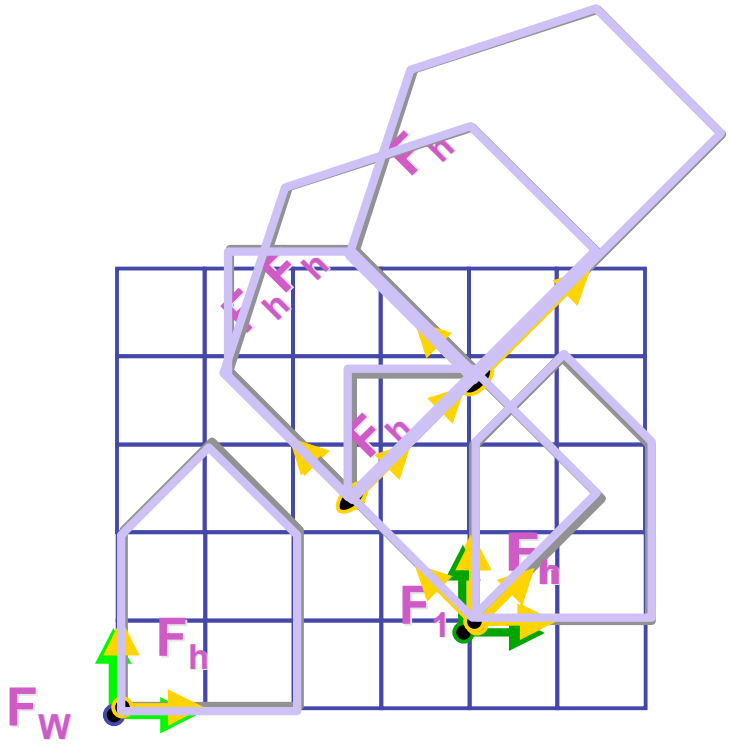
glScale3f(2,2,2)

glTranslate3f(1,0,0)

DrawSquare()

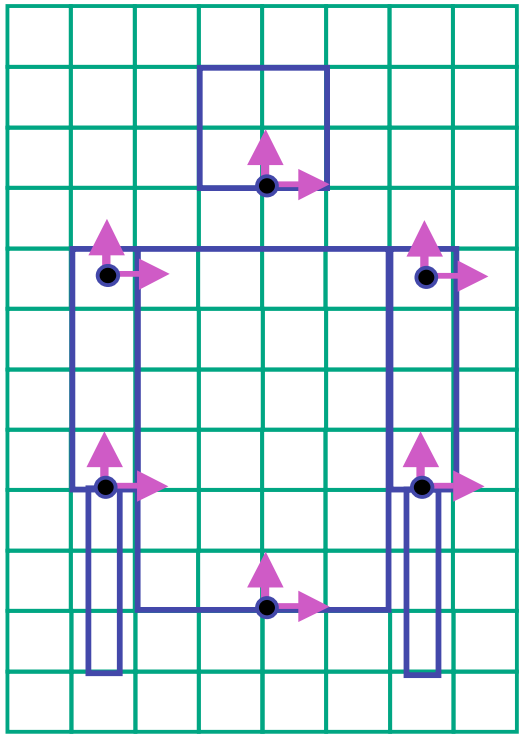
glPopMatrix()

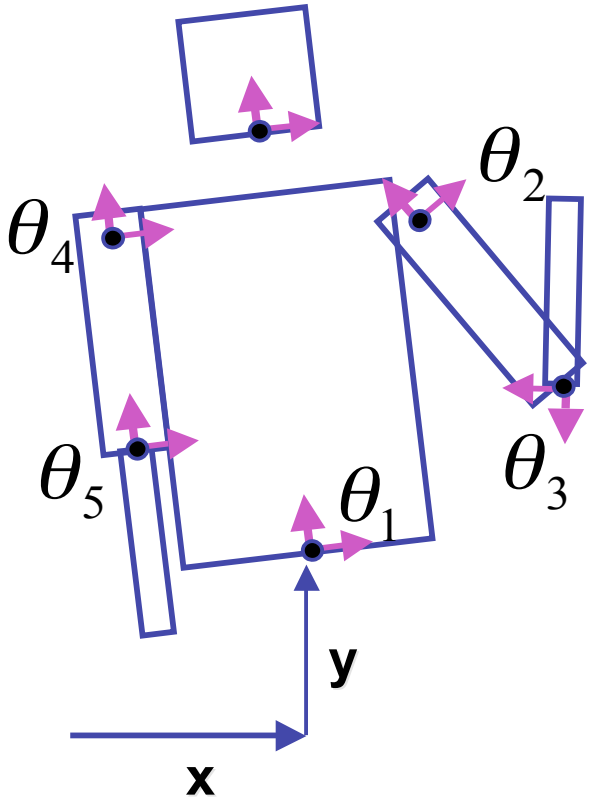
Transformation Hierarchy Example 3




```
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 Hierarchy Example 4






```
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 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

Display Lists

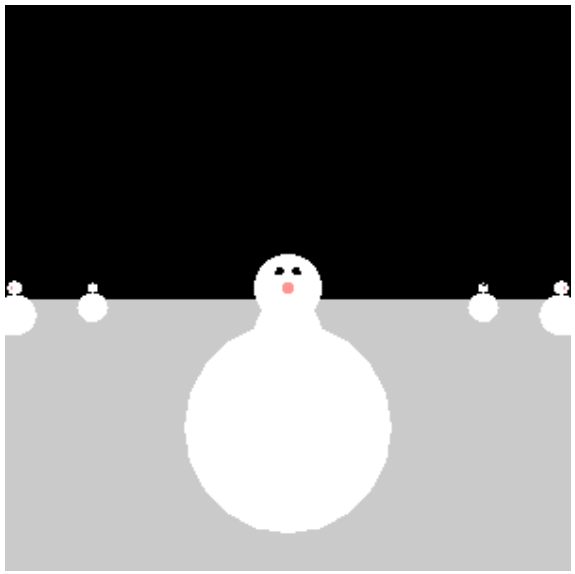
- precompile/cache block of OpenGL code for reuse
  - usually more efficient than immediate mode
    - exact optimizations depend on driver
  - good for multiple instances of same object
    - but cannot change contents, not parametrizable
  - good for static objects redrawn often
    - display lists 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(Dlid);
    glPopMatrix(); }
```

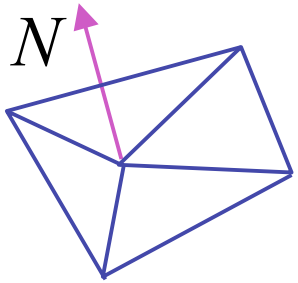
36K polygons, 153 FPS

Transforming Normals

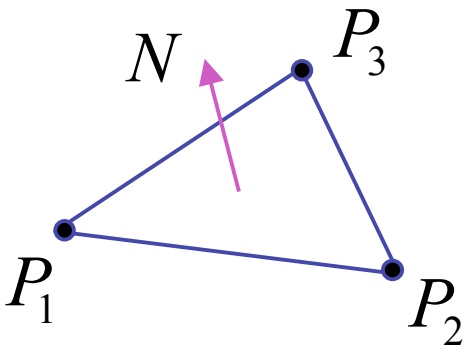
Transforming Geometric Objects

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

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



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

Transforming Normals

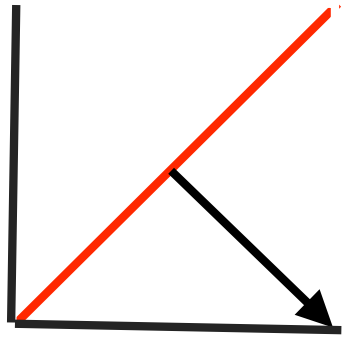
$$\begin{bmatrix} x' \\ y' \\ z' \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

- 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 Normals

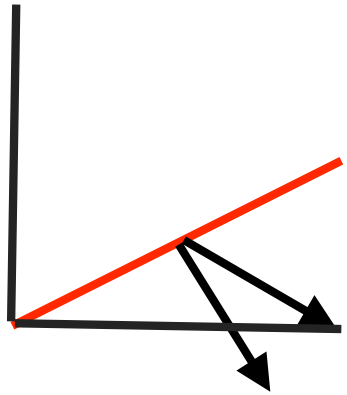
- nonuniform scaling does not work
- x-y=0 plane
  - line x=y
  - normal: [1,-1,0]
    - direction of line x=-y
    - (ignore normalization for now)



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
  - $N \bullet P = 0$ (with dot product)
  - $N^T \bullet P = 0$ (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

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

given M, what should Q be?

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

stay perpendicular

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

substitute from above

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

$$(\mathbf{AB})^{\mathbf{T}} = \mathbf{B}^{\mathbf{T}}\mathbf{A}^{\mathbf{T}}$$

$$Q^T M = I$$

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

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

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