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

Tamara Munzner

Transformations III

Week 2, Fri Jan 19

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

Readings for Jan 15-22

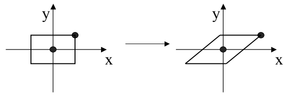
- FCG Chap 6 Transformation Matrices
  - *except* 6.1.6, 6.3.1
- FCG Sect 13.3 Scene Graphs
- RB Chap Viewing
  - Viewing and Modeling Transforms *until* Viewing Transformations
  - Examples of Composing Several Transformations *through* Building an Articulated Robot Arm
- RB Appendix Homogeneous Coordinates and Transformation Matrices
  - *until* Perspective Projection
- RB Chap Display Lists

News

- reminder: office hours today after class in 011 lab
- reminder: course newsgroup is ubc.courses.cpsc.414

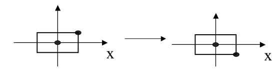
Review: Shear, Reflection

- shear along x axis
  - push points to right in proportion to height



$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}1 & sh_x\\0 & 1\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\0\end{bmatrix}$$

- reflect across x axis
  - mirror



$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\0\end{bmatrix}$$

Review: 2D Transformations

matrix multiplication

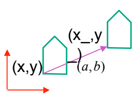
$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}a & 0\\0 & b\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$

scaling matrix

matrix multiplication

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}\cos(\theta) & -\sin(\theta)\\\sin(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$

rotation matrix



vector addition

$$\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}a\\b\end{bmatrix}=\begin{bmatrix}x+a\\y+b\end{bmatrix}=\begin{bmatrix}x'\\y'\end{bmatrix}$$

$$\begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}x'\\y'\end{bmatrix}$$

translation multiplication matrix??

Review: Linear Transformations

- linear transformations are combinations of
  - shear
  - scale
  - rotate
  - reflect

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}\qquad x'=ax+by \qquad y'=cx+dy$$

- properties of linear transformations
  - satisifes T(*s***x**+*t***y**) = *s* T(**x**) + *t* T(**y**)
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Correction: Composing Transformations

- scaling

$$S2 \bullet S1 = \begin{bmatrix}sx_1 * sx_2 & & & \\ & sy_1 * sy_2 & & \\ & & 1 & \\ & & & 1\end{bmatrix}$$

so scales multiply

- rotation

$$R2 \bullet R1 = \begin{bmatrix}\cos(\theta 1+\theta 2) & -\sin(\theta 1+\theta 2) & & \\ \sin(\theta 1+\theta 2) & \cos(\theta 1+\theta 2) & & \\ & & 1 & \\ & & & 1\end{bmatrix}$$

so rotations add

Review: 3D Homog Transformations

- use 4x4 matrices for 3D transformations

translate(a,b,c)

$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}1 & & & a\\ & 1 & & b\\ & & 1 & c\\ & & & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

scale(a,b,c)

$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}a & & & \\ & b & & \\ & & c & \\ & & & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

$\text{Rotate}(x,\theta)$

$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=\begin{bmatrix}1 & & & \\ & \cos\theta & -\sin\theta & \\ & \sin\theta & \cos\theta & \\ & & & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

$\text{Rotate}(y,\theta)$

$$\begin{bmatrix}\cos\theta & & \sin\theta & \\ & 1 & & \\ -\sin\theta & & \cos\theta & \\ & & & 1\end{bmatrix}$$

$\text{Rotate}(z,\theta)$

$$\begin{bmatrix}\cos\theta & -\sin\theta & & \\ \sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1\end{bmatrix}$$

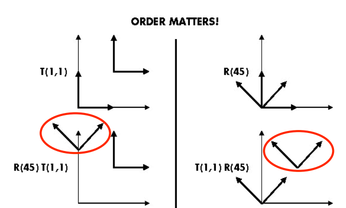
Review: Affine Transformations

- affine transforms are combinations of
  - linear transformations
  - translations

$$\begin{bmatrix}x'\\y'\\w\end{bmatrix}=\begin{bmatrix}a & b & c\\d & e & f\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\\w\end{bmatrix}$$

- properties of affine transformations
  - origin does not necessarily map to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

More: Composing Transformations



Ta Tb = Tb Ta, but Ra Rb != Rb Ra and Ta Rb != Rb Ta

- rotations around different axes do not commute

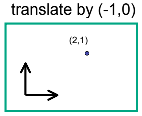
Review: Composing Transformations

$$\mathbf{p'} = \mathbf{TRp}$$

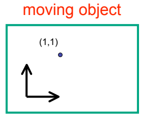
- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
    - moving object
  - left to right **OpenGL pipeline ordering!**
    - interpret operations wrt local coordinates
    - changing coordinate system
    - OpenGL updates current matrix with postmultiply
      - glTranslatef(2,3,0);
      - glRotatef(-90,0,0,1);
      - glVertexf(1,1,1);
    - specify vector last, in final coordinate system
    - first matrix to affect it is specified second-to-last

Interpreting Transformations

translate by (-1,0)

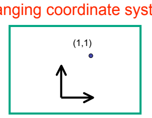


moving object



intuitive?

changing coordinate system



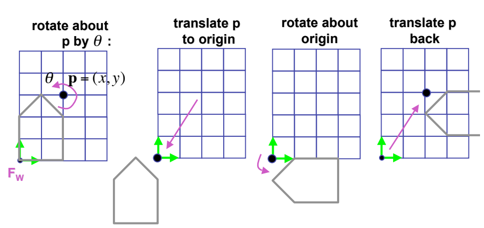
OpenGL

- same relative position between object and basis vectors

Matrix Composition

- matrices are convenient, efficient way to represent series of transformations
  - general purpose representation
  - hardware matrix multiply
  - matrix multiplication is associative
    - **p_** = (T*(R*(S***p**)))
    - **p_** = (T*R*S)***p**
- procedure
  - correctly order your matrices!
  - multiply matrices together
  - result is one matrix, multiply vertices by this matrix
  - all vertices easily transformed with one matrix multiply

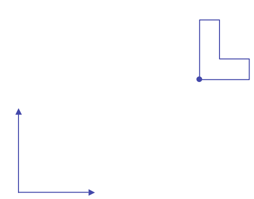
Rotation About a Point: Moving Object



$$\mathbf{T}(x,y,z)\mathbf{R}(z,\theta)\mathbf{T}(-x,-y,-z)$$

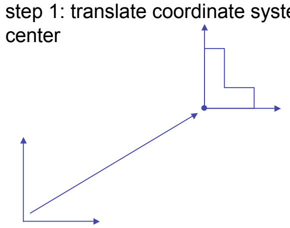
Rotation: Changing Coordinate Systems

- same example: rotation around arbitrary center



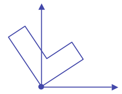
Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 1: translate coordinate system to rotation center



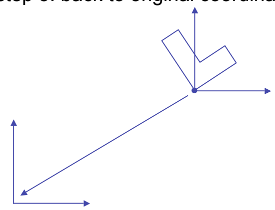
Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 2: perform rotation



Rotation: Changing Coordinate Systems

- rotation around arbitrary center
  - step 3: back to original coordinate system



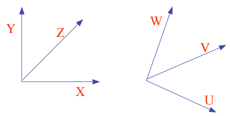
General Transform Composition

- transformation of geometry into coordinate system where operation becomes simpler
  - typically translate to origin
- perform operation
- transform geometry back to original coordinate system

Rotation About an Arbitrary Axis

- axis defined by two points
- translate point to the origin
- rotate to align axis with z-axis (or x or y)
- perform rotation
- undo aligning rotations
- undo translation

Arbitrary Rotation



- problem:
  - given two orthonormal coordinate systems *XYZ* and *UVW*
  - find transformation from one to the other
- answer:
  - transformation matrix R whose columns are *U,V,W:*

$$R = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

Arbitrary Rotation

- why?

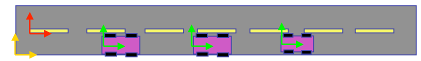
$$R(X) = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = (u_x, u_y, u_z) = U$$

- similarly $R(Y) = V$ & $R(Z) = W$

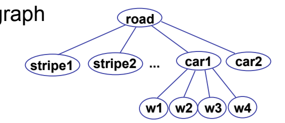
Transformation Hierarchies

Transformation Hierarchies

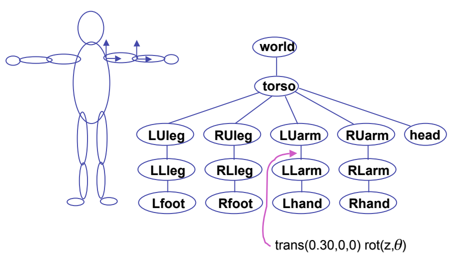
- scene may have a hierarchy of coordinate systems
  - stores matrix at each level with incremental transform from parent's coordinate system



- scene graph

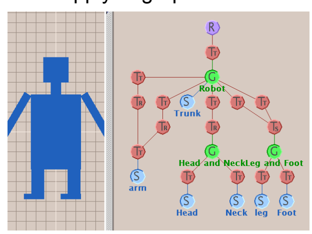


Transformation Hierarchy Example 1



Transformation Hierarchies

- hierarchies don't fall apart when changed
- transforms apply to graph nodes beneath

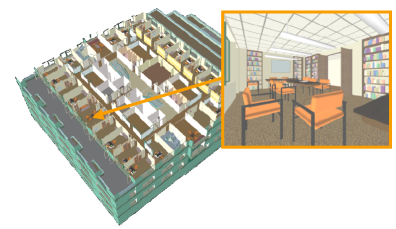


Demo: Brown Applets

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

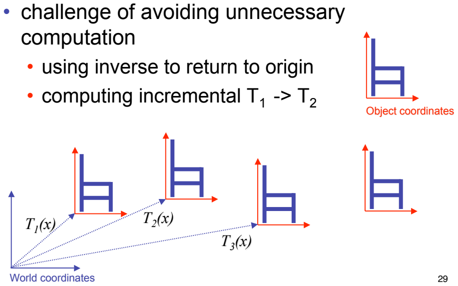
Transformation Hierarchy Example 2

- draw same 3D data with different transformations: instancing

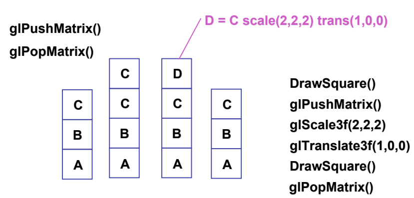


Matrix Stacks

- challenge of avoiding unnecessary computation
  - using inverse to return to origin
  - computing incremental T_1 -> T_2



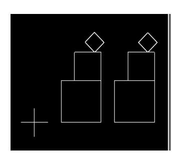
Matrix Stacks



Modularization

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

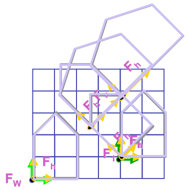
```
void drawBlock(float k) {
  glPushMatrix();

  glScalef(k,k,k);
  glBegin(GL_LINE_LOOP);
  glVertex3f(0,0,0);
  glVertex3f(1,0,0);
  glVertex3f(1,1,0);
  glVertex3f(0,1,0);
  glEnd();

  glPopMatrix();
}
```



Matrix Stacks

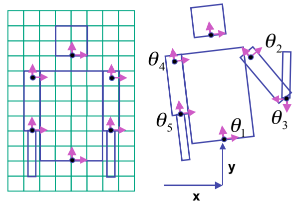
- advantages
  - no need to compute inverse matrices all the time
  - modularize changes to pipeline state
  - avoids incremental changes to coordinate systems
    - accumulation of numerical errors
- practical issues
  - in graphics hardware, depth of matrix stacks is limited
    - (typically 16 for model/view and about 4 for projective matrix)

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)
```

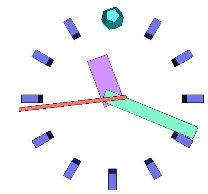

Hierarchical Modelling

- 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

Single Parameter: Simple

- parameters as functions of other params
  - clock: control all hands with seconds s

m = s/60, h=m/60,
theta_s = (2 pi s) / 60,
theta_m = (2 pi m) / 60,
theta_h = (2 pi h) / 60



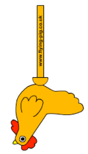
Single Parameter: Complex

- mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk

Single Parameter: Complex

- mechanisms not easily expressible with affine transforms



http://www.flying-pig.co.uk/mechanisms/pages/irregular.html