Transformations III

Week 3, Mon Jan 18

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

• CS dept announcements

• Undergraduate Summer Research Award (USRA)
  • applications due Feb 26
  • see Guiliana for more details
Events this week

Drop-in Resume/Cover Letter Editing
Date: Tues., Jan 19
Time: 12:30 – 2 pm
Location: Rm 255, ICICS/CS Bldg.

Interview Skills Workshop
Date: Thurs., Jan 21
Time: 12:30 – 2 pm
Location: DMP 201
Registration: Email dianejoh@cs.ubc.ca

Project Management Workshop
Speaker: David Hunter (ex-VP, SAP)
Date: Thurs., Jan 21
Time: 5:30 – 7 pm
Location: DMP 110

CSSS Laser Tag
Date: Sun., Jan 24
Time: 7 – 9 pm
Location: Planet Laser @ 100 Braid St., New Westminster

Event next week
Public Speaking 11
Date: Mon., Jan 25
Time: 5 – 6 pm
Location: DMP 101
Assignments
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
    • build iguana out of cubes and 4x4 matrices
      • think cartoon, not beauty
    • template code gives you program shell, Makefile
      • http://www.ugrad.cs.ubc.ca/~cs314/Vjan2010/p1.tar.gz
  
• written homework 1
  • out today, due 5pm sharp Wed Feb 6
  • theoretical side of material
Demo

- animal out of boxes and matrices
Real Iguanas

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

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

Armadillos!
Armadillos!
Monkeys!
Monkeys!
Giraffes!
Giraffes!
Project 1 Advice

• do not model everything first and only then worry about animating
• interleave modelling, animation
  • for each body part: add it, then jumpcut animate, then smooth animate
  • discover if on wrong track sooner
  • dependencies: can’t get anim credit if no model
  • use 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
  - make your cubes wireframe to see inside
- thinking about transformations
  - move physical objects around
  - play with demos
    - Brown scenegraph applets
Project 1 Advice

• smooth transition
  • change happens gradually over X frames
  • key click triggers animation
  • one way: redraw happens X times
    • linear interpolation:
      each time, param += (new-old)/30
  • or redraw happens over X seconds
    • even better, but not required
Project 1 Advice

• transitions
  • safe to linearly interpolate parameters for glRotate/glTranslatef/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, svn/subversion
  - all are installed on lab machines
  - svn tutorial is part of next week’s lab
Graphical File Comparison

- installed on lab machines
  - xfdiff4 (side by side comparison)
  - xwdiff (in-place, with crossouts)
- Windows: windiff
- Macs: FileMerge
  - in /Developer/Applications/Utilities
Readings for Transformations I-IV

• 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
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:
\[
\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}
\]

Translation multiplication matrix??
Review: Linear Transformations

- Linear transformations are combinations of:
  - Shear
  - Scale
  - Rotate
  - Reflect

- Properties of linear transformations:
  - Satisfies $T(sx + ty) = s \ T(x) + t \ T(y)$
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

$$
\begin{align*}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\
x' &= ax + by \\
y' &= cx + dy
\end{align*}
$$
Review: Homogeneous Coordinates

- point in 2D cartesian + weight $w = \frac{1}{w}$ = point $P$ in 3D homog. coords
  - multiples of $(x,y,w)$ form 3D line $L$
  - all homogeneous points on $L$ represent same 2D cartesian point
- homogenize to convert homog. 3D point to cartesian 2D point:
  - divide by $w$ to get $(\frac{x}{w}, \frac{y}{w}, 1)$
  - projects line to point onto $w=1$ plane
  - like normalizing, one dimension up
Review: Homogeneous Coordinates

• 2D transformation matrices are now 3x3:

\[
\text{Rotation} = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}, \quad \text{Scale} = \begin{bmatrix}
a & 0 & 0 \\
0 & b & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\text{Translation} = \begin{bmatrix}
1 & 0 & T_x \\
0 & 1 & T_y \\
0 & 0 & 1
\end{bmatrix}
\]

use rightmost column!

\[
\begin{bmatrix}
1 & 0 & a \\
0 & 1 & b \\
0 & 0 & 1
\end{bmatrix}\begin{bmatrix}x \\
y \\
1\end{bmatrix} = \begin{bmatrix}x*1 + a*1 \\
y*1 + b*1 \\
1\end{bmatrix} = \begin{bmatrix}x + a \\
y + b \\
1\end{bmatrix}
\]
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
# Review: 3D Transformations

The transformations can be represented as matrices:

\[
\begin{bmatrix}
1 & hyx & hzx & 0 \\
hxy & 1 & hzy & 0 \\
hxz & hyz & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\text{shear}(hxy,hxz,hyx,hyz,hzx,hzy)
\]

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 1 & b & 0 \\ 1 & c & 0 \\ 1 \end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

\[
\text{translate}(a,b,c)
\]

\[
\text{scale}(a,b,c)
\]

\[
\text{shear}(hxy,hxz,hyx,hyz,hzx,hzy)
\]

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

\[
\text{Rotate}(x, \theta)
\]

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

\[
\text{Rotate}(y, \theta)
\]

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

\[
\text{Rotate}(z, \theta)
\]

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]
Review: Composing Transformations

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

- translations commute
- rotations around same axis commute
- rotations around different axes do not commute
- rotations and translations do not commute
Review: Composing Transformations

\[ p' = TRp \]

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

\[ p' = TRp \]

- which direction to read?
  - right to left
    - interpret operations \( \text{wrt} \) fixed coordinates
    - moving object
      - draw thing
      - rotate thing by \(-90\) degrees \( \text{wrt} \) origin
      - translate it \((-2, -3)\) over
More: Composing Transformations

\[ p' = TRp \]

- which direction to read?
  - left to right
    - interpret operations \textit{wrt} local coordinates
    - changing coordinate system
      - translate coordinate system \((2, 3)\) over
      - rotate coordinate system 90 degrees \textit{wrt} origin
      - draw object in current coordinate system
  - in OpenGL, cannot move object once it is drawn!!
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

- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems $XYZ$ and $ABC$
    - $A$’s location in the XYZ coordinate system is $(a_x, a_y, a_z, 1)$, ...
arbitrary rotation: change of basis

- given two orthonormal coordinate systems \( XYZ \) and \( ABC \)
- \( A' \)'s location in the XYZ coordinate system is \( (a_x, a_y, a_z, 1) \), ...
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 $A, B, C$:

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

```
road
  /   
stripe1 stripe2 ... car1 car2 ... w1 w2 w3 w4
```
Transformation Hierarchy Example 1

world

trans(0.30,0,0) rot(z,\theta)
Transformation Hierarchy Example 2

- draw same 3D data with different transformations: instancing
Transformation Hierarchies Demo

- transforms apply to graph nodes beneath

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

- transforms apply to graph nodes beneath

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

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

Object coordinates

World coordinates
Matrix Stacks

\[ \text{D} = \text{C scale}(2,2,2) \text{ trans}(1,0,0) \]

\begin{align*}
glPushMatrix() & \quad C \\
glPushMatrix() & \quad B \\
\text{DrawSquare()} & \quad A \\
glPopMatrix() & \\
glPopMatrix() & \\
\end{align*}

\begin{align*}
\text{glPushMatrix()} & \\
\text{glScale3f}(2,2,2) & \\
\text{glTranslate3f}(1,0,0) & \\
\text{glPopMatrix()} & \\
\text{DrawSquare()} & \\
\text{glPushMatrix()} & \\
\text{glScale3f}(2,2,2) & \\
\text{glTranslate3f}(1,0,0) & \\
\text{glPopMatrix()} & \\
\text{DrawSquare()} & \\
\end{align*}
Modularization

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

```cpp
void drawBlock(float k) {
    glPushMatrix();
    glScalef(k,k,k);
    glBegin(GL_LINE_LOOP);
    glVertex3f(0,0,0);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);
    glVertex3f(1,0,0);
    glVertex3f(1,1,0);
    glVertex3f(1,1,0);
    glVertex3f(0,1,0);
    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

```gl
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslate(1,0,0);
glPopMatrix();
glPopMatrix();
```

Transformation Hierarchy Example 4

```cpp
glTranslatef(x, y, 0);
glRotatef(\theta_1, 0, 0, 1);
DrawBody();

glPushMatrix();
  glTranslatef(0, 7, 0);
  DrawHead();
glPopMatrix();

glPushMatrix();
  glTranslatef(2.5, 5.5, 0);
  glRotatef(\theta_2, 0, 0, 1);
  DrawUArm();
  glTranslatef(0, -3.5, 0);
  glRotatef(\theta_3, 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