Assignments

- project 1
  - out today, due 5pm sharp Fri Jan 29
    - projects will go out before we’ve covered all the material
      - 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

- written homework 1
  - out today, due 5pm sharp Wed Feb 6
  - theoretical side of material

Demo

- animal out of boxes and matrices
- 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
- center of object: range - .5 to +.5
- corner of object: range 0 to 1

News

- CS dept announcements
- Undergraduate Summer Research Award (USRA)
  - applications due Feb 26
  - see Guiliana for more details

Assignments

- project 1
  - out today, due 5pm sharp Fri Jan 29
    - projects will go out before we’ve covered all the material
      - 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

- written homework 1
  - out today, due 5pm sharp Wed Feb 6
  - theoretical side of material

Demo

- animal out of boxes and matrices
- 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
- center of object: range - .5 to +.5
- corner of object: range 0 to 1

Assignments

- project 1
  - out today, due 5pm sharp Fri Jan 29
    - projects will go out before we’ve covered all the material
      - 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

- written homework 1
  - out today, due 5pm sharp Wed Feb 6
  - theoretical side of material

Demo

- animal out of boxes and matrices
- 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
- center of object: range - .5 to +.5
- corner of object: range 0 to 1

Assignments

- project 1
  - out today, due 5pm sharp Fri Jan 29
    - projects will go out before we’ve covered all the material
      - 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

- written homework 1
  - out today, due 5pm sharp Wed Feb 6
  - theoretical side of material
**Affine Transformations**
- Affine transforms are combinations of linear transformations and translations.
- Properties of affine transformations:
  - Origin does not necessarily map to origin.
  - Parallel lines remain parallel.
  - Ratios are preserved.
  - Closed under composition.

**Linear Transformations**
- Linear transformations are combinations of:
  - Scale
  - Translate
  - Rotate
  - Shear

**Homogeneous Coordinates**
- Linear transformations before homogenization:
  - Matrix multiplication:
    \[
    \begin{bmatrix}
    x' \\
    y'
    \end{bmatrix} = \begin{bmatrix}
    a & b & c \\
    d & e & f
    \end{bmatrix} \begin{bmatrix}
    x \\
    y
    \end{bmatrix}
    \]
- Properties of linear transformations:
  - Satisfies \( T(x,y) \) = \( T(x+1,y+1) \)
  - Lines map to lines.
  - Parallel lines remain parallel.
  - Ratios are preserved.
  - Closed under composition.
- Homogeneous to cartesian coordinates:
  - Point in 2D cartesian + weight \( w \) = 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.

**2D Transformations**
- Translation matrix:
  \[
  \begin{bmatrix}
  1 & 0 & a \\
  0 & 1 & b
  \end{bmatrix}
  \]
- Rotation matrix:
  \[
  \begin{bmatrix}
  \cos(t) & -\sin(t) & 0 \\
  \sin(t) & \cos(t) & 0
  \end{bmatrix}
  \]
- Shear matrix:
  \[
  \begin{bmatrix}
  1 & b & 0 \\
  0 & 1 & 0
  \end{bmatrix}
  \]
- Matrix multiplication:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} = \begin{bmatrix}
  a & b & c \\
  d & e & f
  \end{bmatrix} \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

**3D Transformations**
- Transformations commute:
  - Rotations around same axis commute.
  - Rotations around different axes do not commute.
  - Rotations and translations do not commute.

**Composing Transformations**
- Order matters:
  - \( T(a,b,c) = T(b,c,a) \)
  - \( T(a,b,c) \circ T(b,c,a) = T(a,b,c) \circ T(c,a,b) \)

**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 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.
  - p, atomevents.cpp, pfixesbug.cpp.
- Much better: use version control software.
  - Stronly recommended.
**Composing Transformations**

- \( p' = T R p \)
- which direction to read?
  - right to left
  - interpret operations wrt fixed coordinates
  - moving object
  - draw thing
  - rotate thing by -90 degrees wrt origin
  - translate it (-2, -3) over

**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

**Arbitrary Rotation**

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

**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

**Modularization**

- drawing a scaled square
  - push/pop ensures no coord system change
  - using inverse to return to origin
  - computing incremental \( T_1 \to T_2 \)

```c
void drawBlock(float k) {
  glPushMatrix();
  glScalef(k,k,k);
  glPushMatrix();
  DrawSquare();
  gPushMatrix();
  glTranslatef(0,1,0);
  glPushMatrix();
  DrawSquare();
  gPopMatrix();
  gPopMatrix();
  gPopMatrix();
}
```
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

```plaintext
GLfloat identity[];
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

```plaintext
glTranslatef(x, y, 0);
glRotatef(θ, 0, 0, 1);
DrawBody();
glPushMatrix();
glTranslate3f(0, 7, 0);
DrawHead();
glPopMatrix();
```

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