3D Transforms
Transformations in OpenGL

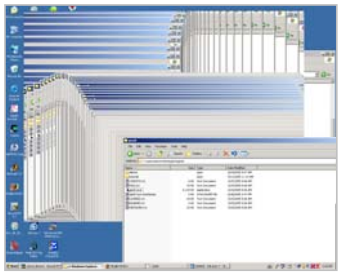
CS148: Intro to CG
Instructor: Dan Morris
TA: Sean Walker
July 5, 2005

Homework 1 is due tonight

- Check the FAQ
- Compile and test on Linux
- Take advantage of extra credit
- Make pretty pictures

Answering old questions

- What happens if you don't update the framebuffer often enough?



Answering old questions

- What does it mean to have a homogeneous coordinate other than 1.0?

translation

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

rotation

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

scale

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

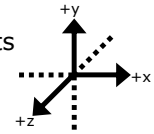
- If (w==0), which transformation(s) still "work" (affect x and y)?
- (w==0) defines an "ideal point" / point at infinity / direction vector

Outline for today

- 3D Transformations
- Transformations in OpenGL
- Video Break
- Transform Hierarchies
- Project 2

OpenGL / CS148 Conventions

- Positive z-axis points out of the monitor



- Counterclockwise rotation is positive (right-hand rule)

Review: Definition of Transformation

- A transformation is a function of a point that returns another point
- When applied to every point in an object, it can change the position, size, or shape of the object
- We do our math using the point-based interpretation, but we're really interested in moving objects

Review: Homogeneous Coordinates

- Writing every point (x,y) as a triple (x,y,w) let us…

Why did we write points in homogeneous coordinates?

Self-plagiarism: Homogeneous coordinates in 3D

- Write a point (x,y,z) as a quadlet:

  [xh,yh,zh,w]

  …where xh = x*w, yh = y*w, zh = z*w
- w is called the 'homogeneous coordinate' and is usually equal to one
- When w = 1, x = xh, y = yh, and z = zh

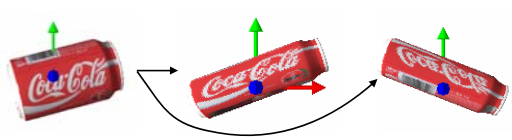
3D Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} tx + x \\ ty + y \\ tz + z \\ 1 \end{bmatrix}$$

What were the signs of tx, ty, and tz?

3D Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} (sx)x \\ (sy)y \\ (sz)z \\ 1 \end{bmatrix}$$



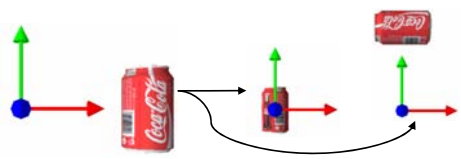
What were the signs of sx, sy, and sz?

Scaling with a fixed point…

- Same as in 2D:
  - Translate the fixed point to the origin
  - Scale
  - Translate the fixed point back where it belongs

3D Rotation

- In 2D, a rotation transformation rotated a point around the origin
- In 3D, saying "rotate a point around the origin" doesn't define a unique transformation



3D Rotation

- 3D rotation is defined by an *axis* of rotation (a 3d-vector) and a rotation angle θ

What were the angles and axes of rotation on the previous slide?

What was the axis of rotation in the previous lecture?

3D Rotation

- So rotation around the z-axis should look familiar:

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

What coordinate shouldn't change if I rotate around the x-axis instead?

Rotation around the x and y axes

x-axis
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y\cos\theta - z\sin\theta \\ y\sin\theta + z\cos\theta \\ 1 \end{bmatrix}$$

y-axis
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta + z\sin\theta \\ y \\ -x\sin\theta + z\cos\theta \\ 1 \end{bmatrix}$$

Looks a lot like the z-axis rotation, right?

Rotation around an arbitrary vector

- Math is a big mess, but there is a magic formula
- OpenGL handles this for you (coming soon to a slide near you)
- But OpenGL only knows how to rotate around an arbitrary vector that passes through the origin…

Rotation around an arbitrary line

How can we rotate around an arbitrary line?

Review: what do our transformations "look like"?

$$\begin{bmatrix} r_1 & r_2 & r_3 & 0 \\ r_4 & r_5 & r_6 & 0 \\ r_7 & r_8 & r_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} r/s & r/s & r/s & t \\ r/s & r/s & r/s & t \\ r/s & r/s & r/s & t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Important CG Terminology: Affine vs. Projective Transforms

- All the transforms we have looked at leave w alone
- This defines the class of *affine transforms*
- Parallel lines remain parallel
- *Projective transforms* can modify w and don't preserve parallelism
- Affine transforms are a subset of projective transforms

Reminder: Composite Transforms

- Any series of transformations can be multiplied together and represented as a single matrix
- Works in 3D just like it did in 2D…

$$P' = C\,P = (R \times T \times S)\,P$$

Three transforms are compressed into one matrix

Outline for today

- 3D Transformations
- Transformations in OpenGL
- Video Break
- Transform Hierarchies
- Project 2

Transformations in OpenGL

- Important concept: GL maintains a "current" transformation matrix that will transform every vertex you send it
- Just like a "current" drawing color
- Defaults to the identity matrix

Translation in OpenGL

glTranslatef(x,y,z);

Dear OpenGL,

Please translate every vertex I send you by (x,y,z) from now on.

Thanks3f(a million,a million,a million),
Dan

What does this really do?

- From the glTranslate docs:

  `Multiply the current matrix by a translation matrix.`

- If the current transformation matrix is C, the new matrix will be:

  C′ = C * T

- GL transforms *post-multiply* the current matrix (the new matrix goes on the *right* of the current matrix)

Rotations and scales

glRotatef(angle, x, y, z);

- Multiply the current matrix by a rotation matrix
- Rotate all subsequent points [angle] degrees around the axis (x,y,z) (which passes through the origin)

glScalef(x,y,z);

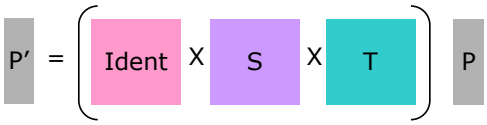
- The same deal…

Where will this point end up?

```
glTranslatef(1.0,0,0);
glScalef(2.0,2.0,2.0);
glBegin(GL_POINTS);
  glVertex3d(1.0,1.0,1.0);
glEnd();
```

Where will this point end up?

```
glScalef(2.0,2.0,2.0);
glTranslatef(1.0,0,0);
glBegin(GL_POINTS);
  glVertex3d(1.0,1.0,1.0);
glEnd();
```



Summary [transforms.cpp]

- OpenGL translations get applied in *reverse* order relative to the order in which they were specified

One more… where will this pt end up?

```
glRotatef(90,0,0,1);
glTranslatef(-1.0,0,0);
glBegin(GL_POINTS);
  glVertex3d(1.0,0.0,0.0);
glEnd();
```

Okay ONE more… what if I switch R and T?

Un-doing transformations

glTranslatef(x,y,z);

- Undo with glTranslatef(-x,-y,-z);

glRotatef(angle, x, y, z);

- **Undo with ???**

glScalef(x,y,z);

- **Undo with ???**

Un-doing transformations

- Matrix interpretation: putting the matrix T^{-1} on the right side of the composite transform

$C = (T_1R_1S_1R_2T_2)$… and we want to undo T_2
$C' = C * T_2^{-1} = (T_1R_1S_1R_2T_2)T_2^{-1}$
$= (T_1R_1S_1R_2)(T_2T_2^{-1})$
$= (T_1R_1S_1R_2)(I)$
$= (T_1R_1S_1R_2)$

- Can only undo the most recent transform
- If you want to undo multiple transforms, you need to undo them in reverse order
- This is not how you usually undo transformations

Outline for today

- 3D Transformations
- Transformations in OpenGL
- Video Break
- Transform Hierarchies
- Project 2

Video break: Unreal Engine 3 Demo

- Demonstrates the state-of-the art in everything that's hard about graphics for video games:
  - Physics
  - Lighting
  - Modeling and animation



Outline for today

- 3D Transformations
- Transformations in OpenGL
- Video Break
- Transform Hierarchies
- Project 2

A square centered at the origin

```
void drawSqare() {
  glBegin(GL_QUADS);
      glVertex3f(-0.5,-0.5,0);
      glVertex3f( 0.5,-0.5,0);
      glVertex3f( 0.5, 0.5,0);
      glVertex3f(-0.5, 0.5,0);
  glEnd();
}
```

A square centered at the origin?

```
glTranslatef(-1.0,0,0);
drawSquare();
```

Where will the square end up?

- Calling glTranslate() moves our origin
- Transformations establish a new coordinate frame

Coordinate Frames

- Can think of every call to glVertex as drawing a vertex relative to the origin
- But transformations move the origin / frame of reference around
- Functions like glutSolidSphere() or our drawSquare() draw *objects* centered at the current origin

Coordinate Frames

- All *transformations* are also defined in the current reference frame

glTranslatef(1,0,0)

- Translation moves all subsequent vertices to the right

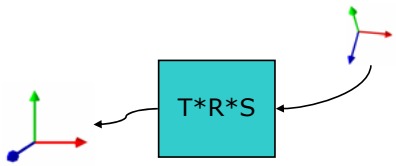
glRotatef(90,0,0,1)

glTranslatef(1,0,0)

- Translation moves all subsequent vertices up (+y)
- Translation is applied in the *rotated coordinate frame*

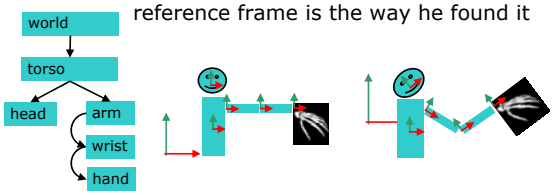
Coordinate Frames

The current transformation matrix defines the transformation from the current reference frame to the global reference frame.



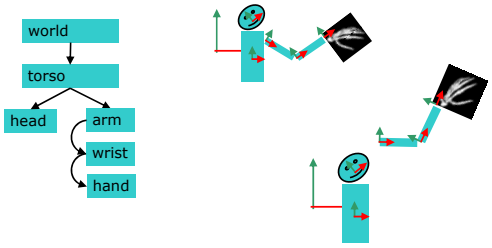
Transformation Hierarchies

- Very common paradigm in 3D graphics:
  - Object sets up new reference frame (translation + rotation) and draws itself
  - Tells "children" to draw themselves
  - Erases his transformations so the reference frame is the way he found it



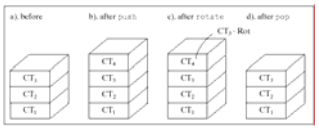
Transformation Hierarchies

What would happen if the head forgot to cancel his reference frame?



Transformation Hierarchies in GL

- GL provides a mechanism to simplify transform hierarchies
- Actually maintains a *stack* of transformations, where the top one is the current transformation
- To create a new local frame, I push the current matrix onto the stack and modify it
- To restore a more global frame, I pop the top of the stack



Transformation Hierarchies in GL

- Useful functions:

```
glPushMatrix();
glPopMatrix();
glLoadIdentity();
```

- Typical GL object [planetup.cpp]:

```
glPushMatrix();
   // Do some transforms
   // Draw myself (maybe by calling a display list)
   // Draw my children
glPopMatrix();
```

Matrix Modes

- GL actually maintains two stacks and two "current" matrices
- glMatrixMode controls which stack you're working with right now

```
glMatrixMode(GL_PROJECTION);
glMatrixMode(GL_MODELVIEW);
```

- Projection matrix is applied last (on the left)

Matrix Modes: What's the Deal?

- Rotations / translations / scales almost always go on the modelview matrix
- Almost always *only* use the projection matrix for perspective projection, which we'll learn about next week.
- Typical convention: if you set the matrix mode to GL_PROJECTION, set it back to GL_MODELVIEW when you're done
- A typical program rarely touches the projection matrix more than once

Outline for today

- 3D Transformations
- Transformations in OpenGL
- Video Break
- Transform Hierarchies
- Project 2

Project 2 Preview

- Project 2: Tobor and Rubix
- Look at the planetup.cpp example
- Start early; it's bigger than pp1!
- Talk to each other
- The next project will (optionally) be a group project….