#3: Hierarchical Transforms. Geometric Calculations

CSE167: Computer Graphics
Instructor: Ronen Barzel
UCSD, Winter 2006
Outline for Today:

- *Hierarchical Transforms*
- Geometric Calculations
Object and World Coordinates

- In project1, constructed matrix to transform points of cube
  - Cube defined using (-1,1,1), …
  - Transformed each point to final position
Object Coordinates

- Each object is defined using some convenient coordinates
  - Often “Axis-aligned”. (when there are natural axes for the object)
  - Origin of coordinates is often in the middle of the object
  - Origin of coordinates is often at the “base” or corner of the object
  - E.g. cube in project1 was -1,-1,1… could also use just 0,1 Axes could be lined up any way.
    - Model of a book: put the Z axis along the spine? Front-to-back?
- No “right” answer. Just what’s most convenient for whomever builds model
- Notice: build, manipulate object in object coordinates
  - Don’t know (or care) where the object will end up in the scene.
- Also called
  - *Object space*
  - *Local coordinates*
World Coordinates

- The common coordinate system for the scene
- Also called World Space
- Also chosen for convenience, no “right” answer.
  - Typically if there’s a ground plane, it’s XY horizontal and Z up
    - That’s most common for people thinking of models
    - I tend to use it a lot

Aside: Screen Coordinates

- X to the right, Y up, Z towards you
  - That’s the convention when considering the screen (rendering)
  - Handy when drawing on the blackboard, slides
  - In project1, World Coordinates == Screen Coordinates
Placing object in the world

- Bundle together a single composite transform.
- Known as:
  - Model Matrix
  - Model Transform
  - World Matrix
  - World Transform
  - Model-to-World Transform
  - Local-to-World Matrix
  - In OpenGL: included in MODELVIEW matrix (composed model & view matrix)
- Transforms each point
Placing object coordinates in the world

- Place the coordinate frame for the object in the world
  - Don’t know or care about the shape of the object
  - World matrix columns = object’s frame in world coordinates
Relative Transformations

- Until now, used a separate world matrix to place each object into the world separately.
- But usually, objects are organized or grouped together in some way
- For example…
  - A bunch of moons and planets orbiting around in a solar system
  - Several objects sitting on a tray that is being carried around
  - A hotel with 1000 rooms, each room containing a bed, chairs, table, etc.
  - A robot with torso and jointed arms & legs
- Placement of objects is described more easily relative to each other rather than always in world space
Sample Scene

KK 5045
1500 x 450 x 760mm

KK 5060
1500 x 600 x 760mm
Schematic Diagram (Top View)
Top view with Coordinates
Relative Transformations

- Put the objects on the tables
  - Each table has a simple coordinate system
  - E.g. Book1 at (3.75,1,0) on Table1’s top
  - E.g. Keyboard at (3,.5,0) on Table2’s top
  - Don’t care where the tables are in order to do this part

- Put the tables in the room
  - Books etc. should end up in the right place

- How do we do this…?
Current coordinate system

- In our code, we maintain a “current coordinate system”
- Everything we draw will be in those coordinates
- I.e. we keep a variable with a matrix known as the “current transformation matrix” (CTM)
  - Everything we draw we will transform using that matrix
  - Transforms from current coordinates to world coordinates
Drawing with a CTM

- Old \texttt{drawCube}:

\begin{verbatim}
drawCube\text{(Matrix M)}
\{
    p1 = M*Point3( 1,-1, 1);
    p2 = M*Point3( 1,-1,-1);
    p3 = M*Point3( 1, 1,-1);
    p4 = M*Point3( 1, 1, 1);
    p5 = M*Point3(-1,-1, 1);
    p6 = M*Point3(-1,-1,-1);
    p7 = M*Point3(-1, 1,-1);
    p8 = M*Point3(-1, 1, 1);
    .
    .
    .
\}
\end{verbatim}

- New \texttt{drawCube}:

\begin{verbatim}
// global CTM
drawCube\text{()}
\{
    p1 = CTM*Point3( 1,-1, 1);
    p2 = CTM*Point3( 1,-1,-1);
    p3 = CTM*Point3( 1, 1,-1);
    p4 = CTM*Point3( 1, 1, 1);
    p5 = CTM*Point3(-1,-1, 1);
    p6 = CTM*Point3(-1,-1,-1);
    p7 = CTM*Point3(-1, 1,-1);
    p8 = CTM*Point3(-1, 1, 1);
    .
    .
    .
\}
\end{verbatim}
Using a CTM

- As we go through the program, we incrementally update the CTM
- Start with the current coordinates=world coordinates
  - CTM = I
- Before we draw an object, we update the CTM
  - from the current location to the object’s location
  - We perform a relative transformation.
  - The CTM accumulates the full current-to-world transformation.
- Draw from the outside in.
  - Draw containers before the things they contain.
Top view, just frames
Table1 and Book1
Draw Table1 and Book1

// Start in World coords, on floor of room
CTM = Matrix::IDENTITY;

// Move to Table1 position, draw table
CTM = CTM*Matrix.MakeTranslate(2,8,0);
drawTable();

// Move up to tabletop height
CTM = CTM*Matrix.MakeTranslate(0,0,3);

// Move to Book1 position & orientation, draw
CTM = CTM*Matrix.MakeTranslate(3.75,1,0);
CTM = CTM*Matrix.MakeRotateZ(90);
drawBook();
Simplify the idiom

Routines that affect the CTM:
- LoadIdentity ()  { CTM = Matrix::IDENTITY }
- Translate(V)     { CTM = CTM*Matrix::MakeTranslate(V) }
- RotateZ(angle)    { CTM = CTM*Matrix::MakeRotateZ(angle) }
- Etc…
- Transform(M)     { CTM = CTM*M }
Draw Table1 and Book, redux

// Start in World coords, on floor of room
LoadIdentity();

// Move to Table1 position, draw table
Translate(2,8,0);
drawTable();

// Move up to tabletop height
Translate(0,0,3);

// Move to Book1 position & orientation, draw
Translate(3.75,1,0);
RotateZ(90);
drawBook();
Table2 and Keyboard
Draw Table2 and Keyboard

    // Start in World coords, on floor of room
    LoadIdentity();

    // Move to Table2 position & orientation, draw
    Translate(2,8,0);
    RotateZ(-81);
    drawTable();

    // Move up to tabletop height
    Translate(0,0,3);

    // Move to Keyboard position, draw
    Translate(3,0.5,0);
    drawKeyboard();
What about drawing entire scene?

- After we drew Book1 or Keyboard, our coordinate system had moved deep into the world somewhere.
- How do we get back...?
  - To the tabletop coordinates so we can place another book?
  - To the room coordinates so we can place another table?
- Don’t want to start over at the beginning for each object.
- At each stage, need to remember where we are so we can get back there
Keep a Stack for the CTM

- Add two more routines:
  - PushCTM() -- saves a copy of the CTM on the stack
  - PopCTM() -- restores the CTM from the stack
Draw whole scene, hierarchically

PushCTM();
  Translate(2,8,0);
  drawTable()
  Translate(0,0,3);
PushCTM();
  Translate(3.75,1,0);
  RotateZ(90);
  drawBook()
PopCTM();
PushCTM();
  Translate(...);
  Rotate(...);
  drawBook();
PopCTM();
...etc...
PopCTM()
...etc...
Hierarchical grouping within a model

- Model can be composed of parts
  - Draw parts using Push & Pop CTM

```c
drawTable(){
    PushCTM() // save
    PushCTM() // draw leg1
    Translate(…);
    drawLeg();
    PopCTM();
    PushCTM() // draw leg2
    Translate(…);
    drawLeg();
    PopCTM();
    ...etc leg3 & leg4…
    PushCTM(); // draw top
    Translate(…);
    drawTableTop();
    PopCTM();
    PopCTM() // restore
}
```

- Has no effect outside this routine.
Access something in the middle?

- CTM always contains the complete Local-to-world transform for what we’re currently drawing.
- Sometimes need to hold on to copy of CTM in the middle
  ```
  pushCTM();
  ...stuff...
  pushCTM();
  ...transform...
  Book1Matrix = CTM;
  drawBook();
  popCTM();
  ...stuff...
  popCTM();
  ```

- Later in code, mosquito lands on Book1
  ```
  pushCTM();
  LoadMatrix(Book1Matrix);
  Translate(...);
  drawMosquito();
  popCTM();
  ```
CTM and matrix stack in OpenGL

- OpenGL provides
  - `glTranslatef(…)`
  - `glRotatef(…)`
  - `glPushMatrix()`
  - `glPopMatrix()

(But don’t use them for proj2--need to know how to do it yourself)

- Actually, other properties, such as color, are also part of “current state” and can be pushed and popped.
Thinking top-down vs bottom-up

- Transforms for World-to-Keyboard (ignoring pushes, pops, etc.):
  1. Translate(2,8,0)
  2. RotateZ(-81)
  3. Translate(0,0,3)
  4. Translate(3,0.5,0)
  5. drawKeyboard()

- Top-down: transform the coordinate frame:
  - Translate the frame, then rotate it, then translate twice more, then draw the object

- Bottom-up: transform the object:
  - Create a keyboard, translate it in X&Y, then in Z, then rotate about the origin, then translate again

- Both ways give same result
- Both ways useful for thinking about it.
Another example:

- Sample sequence:
  1. RotateZ(45)
  2. Translate(0,5,0)
  3. Scale(2,1,1)
  4. drawCube()

- Top-down: transform a coordinate frame:
  - rotate it 45 degrees about its origin, then translate it along its Y, then stretch it in X, then draw the primitive.

- Bottom-up: transform the object
  - create a square, then scale it in X then translate it along the Y axis, then rotate 45 degrees about the origin.

- Both ways useful
Outline for Today:

- Hierarchical Transforms
- Geometric Calculations
Example: Normal of a Triangle

- The concept of a “normal” will be essential to lighting
- Find the unit length normal of the triangle defined by 3D points a, b, c
Example: Normal of a Triangle

- Find the unit length normal of the triangle defined by 3D points \( a, b, \) and \( c \)

\[
\vec{n}^* = (\vec{b} - \vec{a}) \times (\vec{c} - \vec{a})
\]

\[
\vec{n} = \frac{\vec{n}^*}{|\vec{n}^*|}
\]
Example: Alignment to Target

- An object is at position $p$ with a unit length heading of $\vec{h}$. We want to rotate it so that the heading is facing some target at position $t$. Find a unit axis $a$ and an angle $\theta$ to rotate around.
Example: Alignment to Target

\[ \vec{d} = \frac{\vec{t} - \vec{p}}{|\vec{t} - \vec{p}|} \]

\[ \vec{a} = \frac{\vec{h} \times \vec{d}}{|\vec{h} \times \vec{d}|} \]

\[ \theta = \cos^{-1}(\vec{h} \cdot \vec{d}) \]
Example: Distance to Plane

- A plane is described by a point $p$ on the plane and a unit normal $\mathbf{n}$. Find the (perpendicular) distance from point $x$ to the plane.
Example: Distance to Plane

- The distance is the length of the projection of $\mathbf{x} - \mathbf{p}$ onto $\mathbf{n}$:

$$\text{dist} = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}$$
Example: Area of a Triangle

Find the area of the triangle defined by 3D points a, b, and c
Example: Area of a Triangle

\[ area = \frac{1}{2} \left| \overrightarrow{(b - a)} \times \overrightarrow{(c - a)} \right| \]
Scaling and volume

- Scale Matrix:
  \[
  S = \begin{bmatrix}
  s_x & 0 & 0 & 0 \\
  0 & s_y & 0 & 0 \\
  0 & 0 & s_z & 0 \\
  0 & 0 & 0 & 1 \\
  \end{bmatrix}
  \]

- Scaling an object changes its volume by \( s_x \cdot s_y \cdot s_z \)
  - A uniform scale \( s \) changes volume by \( s^3 \)
    - E.g. scaling uniformly by 2 increases volume by 8
  - A non-uniform scale with 0 (projection into a plane) flattens object
    - New volume is 0
Volume Preserving Scale

- Stretch one axis while squashing the other two
  - E.g., a volume preserving scale along the x-axis by a factor of $s_x$ would scale the y and z axes by $1/\sqrt{s_x}$

$$
S_{vx}(s_x) =
\begin{bmatrix}
  s_x & 0 & 0 & 0 \\
  0 & \frac{1}{\sqrt{s_x}} & 0 & 0 \\
  0 & 0 & \frac{1}{\sqrt{s_x}} & 0 \\
  0 & 0 & 0 & 1 \\
\end{bmatrix}
$$

- This way, the total volume change is $s_x \cdot \frac{1}{\sqrt{s_x}} \cdot \frac{1}{\sqrt{s_x}} = 1$
Transforming normals

- A “normal vector” perpendicular to object
  - Not really a vector (a “co-vector”/linear functional/covariant tensor)
  - Represents relationship to other vectors

- After a transform $M$, want it to still be normal.
  - No problem if transform is rigid, or has uniform scale
  - But if $M$ has non-uniform scale, angle won’t stay right:

![Diagram showing normal vector before and after transformation]

- scaled by 0.5 along the $x$ dimension
Transforming normals

- Answer: transform using inverse of transpose of $M$

\[ \bar{v} \cdot \bar{n} = 0 \quad \text{// } \bar{n} \text{ is perpendicular to } \bar{v} \]

\[ \bar{v}' = M \bar{v} \quad \text{// transform } \bar{v} \text{ using } M \]

\[ \bar{n}' = (M^T)^{-1} \bar{n} \quad \text{// transform } \bar{n} \text{ using inverse transpose of } M \]

\[ \bar{v}' \cdot \bar{n}' = (M \bar{v}) \cdot \left((M^T)^{-1} \bar{n}\right) \quad \text{// check resulting dot product...} \]

\[ = (M \bar{v})^T \left((M^T)^{-1} \bar{n}\right) \]

\[ = \bar{v}^T M^T (M^T)^{-1} \bar{n} \]

\[ = \bar{v}^T \bar{n} \]

\[ = 0 \quad \text{// } \ldots \bar{n}' \text{ is perpendicular to } \bar{v}' \]

- For rotations, inverse=transpose, so inverse of transpose is inverse of inverse, i.e. $M$ unchanged.

  - Only makes a difference if $M$ has non-uniform scale (or shear)
A shear transformation matrix looks something like this:

\[
\begin{pmatrix}
1 & z_1 & z_2 & 0 \\
z_3 & 1 & z_4 & 0 \\
z_5 & z_6 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

- With pure shears, only one of the constants is non-zero.
- A shear is equivalent to a non-uniform scale along a rotated set of axes.
- Shears are sometimes used in computer graphics for simple deformations or cartoon-like effects.
Next class

- I’ll be away
- Class will be held by TA’s
  - will go over project code, OpenGL, review transforms, etc…
  - Discussion… bring your questions!
- Next Tuesday:
  - Models
  - Hierarchical Modeling / Scene Graphs