Transformations

Motivation

- We want to move objects around the screen
- We want:
  - Efficiency
  - Flexibility
  - Ease of use

Review

- Up to now, we've been moving objects by adding a vector to position:
  gl_Position = vec4(position + translation, -1, 1)
- Or maybe we scale and translate:
  gl_Position = vec4(position * scale + translation, -1, 1)
- This is OK if we have simple, fixed set of transformations
- But we often want more complex effects

Transforms

- The “big three” transformations:
  - Translate
  - Rotate
  - Scale

Translation

- Not much more to say about it right now
- Just add Δx and Δy to position

Compositing

- Compositing translations is easy
  - Just add them all in any order
  - Addition is *associative* and *commutative*

Rotate

- Rotation: Very commonly used
- We have several ways we can represent rotations

Angle

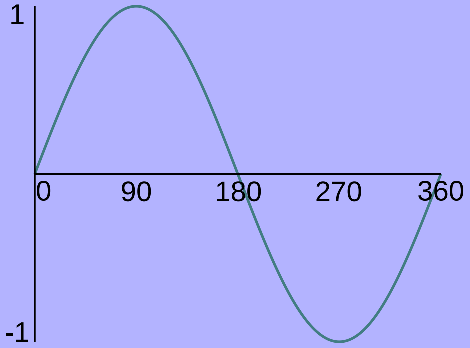
- First way: Angle
- Just a single number: 0...360°
  - Or 0...2π radians

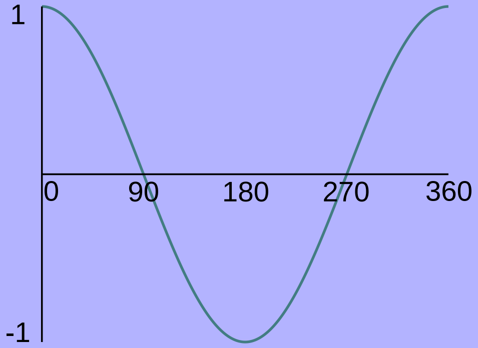
Compositing

- Composing angle-reps is easy
  - Add rotation angles together
- Ex: Rotate clockwise by 40° followed by rotating 15° withershins
  - Same as rotating 35° CW (40 + -15)
- Commutative as well: Same result from doing 15° WS then 40° CW

Implementing

- How to implement?
- Recall our old friends sine and cosine...





Rotation

- Imagine a point p at **x=1, y=0**
  - Let U = p rotated by d degrees
  - Graph the x coordinate. What does it look like as d goes from 0...360°?
  - Graph the y coordinate. What does it look like as d goes from 0...360°?

Rotation

- Imagine a point q at **x=0, y=1**
  - Let V = q rotated by d degrees.
  - Graph the x coordinate. What does it look like as d goes from 0...360°?
  - Graph the y coordinate. What does it look like as d goes from 0...360°?

Result

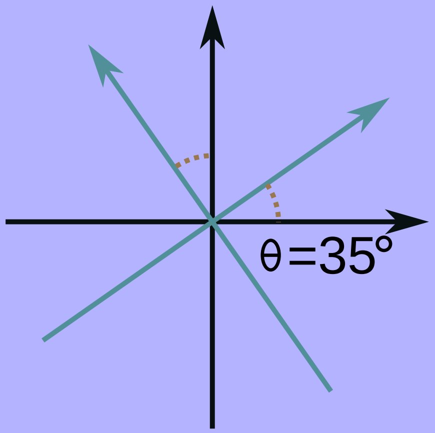
- If we rotate (1,0), it lands at location ($\cos\theta$, $\sin\theta$). Call this U.
- Given (0,1), as we rotate it, it ends up at ($-\sin\theta$, $\cos\theta$). Call this V.

Transformation

- We can view a rotation by angle θ as being a *transformation of axes*
  - If we rotate by $\theta°$, we are rotating our x and y axes accordingly
- An example may help...

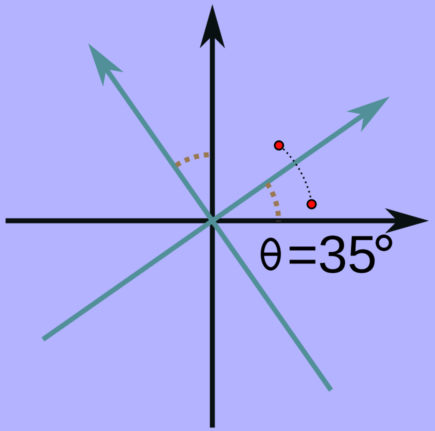
Example

- Suppose we rotate by 35°



Now...

- Suppose we had a point that was at 0.5,0.1 originally
- Where does it end up after rotation?



Answer

- What does coordinate (0.5,0.1) *mean*?
  - Go 0.5 units in the x direction, then 0.1 units in the y direction
- Our transformed point will be 0.5 units down the transformed x axis (U) and 0.1 units down the transformed y axis (V)

Answer

- Recall: Transformed axes:
  - U = ($\cos\theta$, $\sin\theta$)
  - V = ($-\sin\theta$, $\cos\theta$)
- Transformed point (x',y'):
  - 0.5 U + 0.1 V
  - $x' = 0.5\cos\theta + 0.1(-\sin\theta)$
  - $y' = 0.5\sin\theta + 0.1\cos\theta$

In General...

- In general, if (x,y) is the new point, and (x',y') represents rotation of (x,y) by angle θ:
  - $x' = x \cos \theta - y \sin \theta$
  - $y' = x \sin \theta + y \cos \theta$

Implementation

- So we could define a uniform:
  float rotationAngle
- Then send the angle to shader
- But this is not ideal

Problem

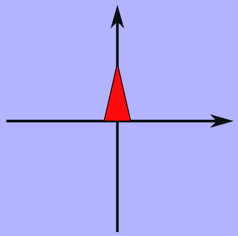
- Trigonometric computations are expensive
  - CPU: About 280 cycles to compute a sine/cosine pair
  - GPU: Varies widely
    - Some GPU’s can do them in one cycle (lookup table)
- But there’s another issue...

Compositing

- What if we want to rotate *and* translate?
- Here, order makes a difference!

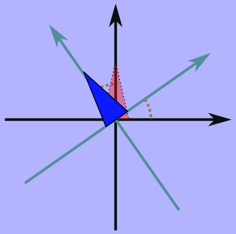
Consider

- Suppose we want to translate by 0.5,0.25 and rotate by 35°
- Suppose we are drawing a triangle
- Here's our triangle to begin with:



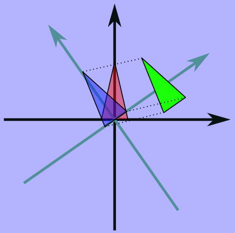
Rotate

- First, we rotate by 35 degrees



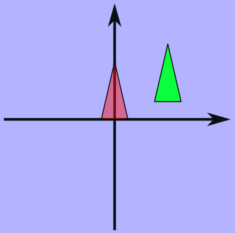
Translate

- Then we translate by 0.5,0.25



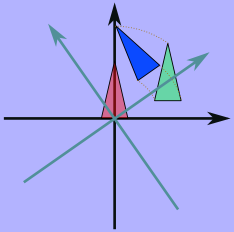
Consider

- Now, suppose we translate first:



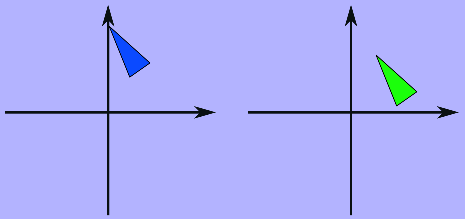
Rotate

- Now we rotate:



Compare

- Compare the results:



So...

- We don't want to hardcode the ordering in the shader
- How can we flexibly support arbitrary sequences of transformations?

And Now...

- For something (apparently) completely different!

Matrices

- Def: Matrix: An $m \times n$ 2D grid of numbers

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

  - Has m rows, n columns

Multiplication

- Define: Suppose we want to multiply two matrices, M and N
  - Let size of M = m_r x m_c
  - Let size of N = n_r x n_c
- $M \cdot N$ is not defined if $m_c \neq n_r$
- Otherwise, result has size m_r x n_c

Example

- Let M = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
- Let N = $\begin{bmatrix} -1 & 0.1 \\ -0.5 & 1 \end{bmatrix}$
- Result: $M \cdot N$ = $\begin{bmatrix} 1 \cdot -1 + 2 \cdot -0.5 & 1 \cdot 0.1 + 2 \cdot 1 \\ 1 \cdot -0.5 + 2 \cdot 1 & 3 \cdot 0.1 + 4 \cdot 1 \end{bmatrix}$
- Or: $\begin{bmatrix} -2 & 2.1 \\ 1.5 & 4.3 \end{bmatrix}$

Example

- We can likewise define multiplication for larger matrices
- Ex:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}$$

- Result:

$$\begin{bmatrix} aj + bm + cp & ak + bn + hq & al + bo + cr \\ dj + em + fp & dk + en + fq & dl + eo + fr \\ gj + hm + ip & gk + hn + iq & gl + ho + ir \end{bmatrix}$$

- Same idea for 2x2 or 4x4 matrices

Multiply

- What if we want to multiply a vector by a matrix?
- Suppose we have a vec2 and a mat2
- We can interpret vec2 as a 2x1 matrix or as a 1x2 matrix
- If we premultiply (v*M): Then we must interpret v as 1x2 (so dimensions are compatible)
- If we postmultiply (M*v): Must interpret v as 2x1

Example

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

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

- In either case, x’=xa+yc and y’=xb+yd

Rotation

- Recall our rotation: To rotate point (x,y) by angle θ:
  - x' = x cos θ - y sin θ
  - y' = x sin θ + y cos θ
- We can express as matrix operation:

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

- Or:

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

Notice

- The rotation matrix for vM is the transpose of the matrix for Mv
- So we have to know which way we're going to do operation when we set up the matrix

Translation

- How do we represent translation?
- No way to do this with vec2's!
- But we can do so if we add another coordinate (w)
  - Now we'll have vec3's: (x,y,w)
- Let w=0 for vectors or 1 for points

Example

- Translate (x,y) by dx,dy
  - x' = x+dx
  - y' = y+dy
- With matrices:

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} x + t_x & y + t_y & 1 \end{bmatrix}$$

- Or:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

- Again, notice matrices are transposes of each other

Vectors

- What if our (x,y) represents a direction (a vector)?

$$\begin{bmatrix} x & y & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} x & y & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

- This is logical: Translating a direction has no meaning

Rotation

- We modify the rotation matrix so it works with vec3's:

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

- This gives:

$$\begin{bmatrix} xcos\,\theta - ysin\,\theta & xsin\,\theta + ycos\theta & w \end{bmatrix}$$

Rotation

- Or:

$$\begin{bmatrix} cos\,\theta & -sin\,\theta & 0 \\ sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} xcos\,\theta - ysin\,\theta \\ xsin\,\theta + ycos\,\theta \\ w \end{bmatrix}$$

- Notice: w is preserved, whether it's zero or one

Orientation

- Suppose we have two *orientations* (rotations) of an object and we want to blend between them
- Let a_1 be one angle of rotation and a_2 be the other one
- We want the angle that's 50% of the way between them.
  - How can we compute this?

Orientation

- What if we want to go 25% of the way from a_1 to a_2?
- What if we want to go 75% of the way from a_1 to a_2?

Interpolation

- This is *linear interpolation*
- Formula: $a' = a_1 + t(a_2 - a_1)$

Assignment

- None!
- Just get caught up on the other labs...

Sources

- Jim Van Verth. Understanding Rotations. http://www.essentialmath.com/GDC2012/GDC2012_JMV_Rotations.pdf
- Intel Corporation. Intel Processor Optimization Manual.

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org