Advanced Graphics: Rotation, 3D, and Scaling

ISE 108: Introduction to Programming
Stony Brook University

List of Topics

- Rotation
- Translation: Moving the origin
- Working in Three Dimensions
- Scaling
- Creating your own polygons
- Grouping transformations

Rotation and Translation

Rotation

- Use `rotate()` to rotate (spin) a shape
- rotation is clockwise
- `rotate()` takes one argument: an angle (in radians)
  - Use the `radians()` command to translate degrees into radians (2π radians = 360°)
  - ex. `rotate(radians(45));`

Rotation: The Catch

- Rotation moves a shape around the origin
- The origin (coordinates (0, 0)) defaults to the top-left corner of the window
  - This means that rotated objects "swing down" instead of spinning around their center
- Solution: move the origin!

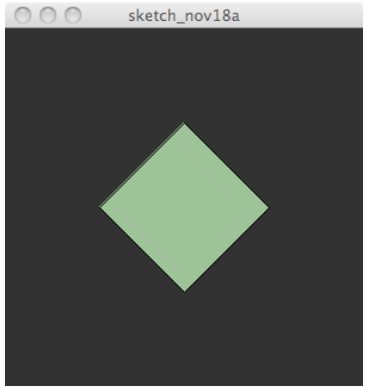
Translation

- Translation shifts the origin from the top left corner to a different location
- Use the `translate()` function to determine the new position of the origin
  - `translate()` takes an x-offset and a y-offset
  - ex. translate(50, 0) moves origin 50 px right
- Note: this changes where every shape is drawn!
- The origin resets to the top left when draw() restarts

Translation and Rotation

```
size(300, 300);
background(50);
fill(150, 200, 150);


translate(150, 150);
rotate(radians(45));
rect(-50, -50, 100, 100);
```



Note: the new origin is at the center of the shape!

Working in 3D

Adding A Third Dimension

- The third dimension runs along the Z axis
  - +Z: out of the screen
  - -Z: into the screen
- To make a 3D sketch, we add an extra argument to size():

size(width, height, P3D);

  - "P3D": use Processing's 3D renderer

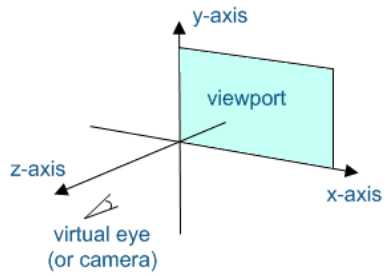


Image credit: http://polymathprogrammer.com/2008/09/01/cartesian-coordinates-and-transformation-matrices/

Translation in 3D

- Translate also accommodates three dimensions
- Just add a third argument to `translate()`:
  - translate(50, 100, -30) shifts the origin 50 pixels right, 100 pixels down, and 30 pixels "away"
- NOTE: 3D translation doesn't look different from 2D unless we also rotate the sketch!

Rotation in 3D

- We can rotate a shape around all three axes
  - By default, shapes rotate around the Z axis (which points out of the screen)
- Use `rotateX()`, `rotateY()`, and `rotateZ()` to specify which axis to rotate around
  - `rotateZ()` is identical to `rotate()`

3D Shapes

- Processing provides two built-in 3D shapes
  - Box — a 3D rectangle
    - Syntax: box (width, height, depth)
  - Sphere — a 3D circle/ellipse
    - Syntax: sphere (radius)
    - Use noStroke() to turn off the "facets"/lines between the pieces that make up the sphere

Stacking Transformations

Combining Rotations

- Suppose that we want to rotate two shapes independently of one another
- Problem: `rotate()` and `translate()` are relative to the window's previous coordinates
  - Calling `rotate()` again causes the second object to rotate around the first, not the origin
- To solve this, we need to "reset" the window's state between rotations, to start "fresh"

Welcome to the Matrix

- Processing uses a matrix (a table of numbers) to store information about the coordinate system
  - Every rotation/translation updates the matrix
- We can save and restore the state of the matrix at any point
  - This lets us "undo" one transformation before we apply the next

Brief Interlude: Stacks

- A stack stores data in Last-In, First-Out order
- Push: add a new item to the top of the stack
- Pop: remove the item at the top of the stack
- Processing uses a stack to save copies of the coordinate matrix

Image source: http://static.dezeen.com/uploads/2008/04/shay_alkalay_stack_1800mm_o.jpg

Repeating History...

- To prevent one transformation from affecting the rest, we need to reset the coordinate matrix after working with each shape
- General strategy:
  1. Save current state of matrix
  2. Perform translation/rotation operations
  3. Restore the old version of the matrix
  4. Repeat steps 1–3 for the next shape

Matrix Operations

- Use `pushMatrix()` to save the current state of the coordinate matrix
- Use `popMatrix()` to restore the coordinate matrix to its last saved state

- You must have an equal number of `pushMatrix()` and `popMatrix()` operations
- The stack has an unlimited storage capacity

Scaling and Polygons

Scaling Images

- Use the `scale()` function to change the size of any objects onscreen
- `scale()` takes a floating-point argument representing the percentage to scale by
  - 1.0 = 100%, 3.0 = 300%, 0.5 = 50%, etc.
- Use two or three arguments to scale by different amounts in those directions/dimensions
  - e.g., scale(1.0, 2.0, 3.0) stretches an object 200% vertically and 300% in depth

Drawing Your Own Shapes

- Processing allows us to create our own polygons
- Use the `vertex()` command to set one point
  - vertex() takes x, y, and (optional) z coordinates
- Start the definition by calling `beginShape()`
- End with a call to `endShape()`
  - use `endShape(CLOSE)` to join the first and last vertices

Simple Polygon Example

- Drawing a square using `vertex()`:

```
beginShape();
vertex(50, 50);   // top left
vertex(150, 50);  // top right
vertex(150, 150); // lower right
vertex(50, 150);  // lower left
endShape(CLOSE);
```

Advanced Polygon Tricks

- Use `curveVertex()` instead of `vertex()` to join points with curved lines
- `beginShape()` can take an optional "mode" argument
  - This argument affects how vertices are connected
- Modes: TRIANGLES, TRIANGLE_FAN, TRIANGLE_STRIP, QUADS, QUAD_STRIP, POINTS, LINES

see http://processing.org/reference/beginShape_.html

Polygon Modes

- Supplying an argument to `beginShape()` tells Processing how to treat the vertices it gets
  - "TRIANGLES"/"QUADS": Every group of 3 or 4 vertex points make up a triangle/quadrangle
  - Adding "_STRIP": triangles/quadrangles are connected by intermediate shapes
  - TRIANGLE_FAN: triangles are positioned around a central point (a common vertex)