Rob Saunders

web: http://web.arch.usyd.edu.au/~rob

e-mail: rob@arch.usyd.edu.au

office: Room 274, Wilkinson Building
Motion
Easing

- Easing, also called interpolation, is a technique for moving between two points.
- By moving a fraction of the total distance each frame, a shape can decelerate (or accelerate) as it approaches a target location.
float x = 20.0; // Initial x-coordinate
float y = 10.0; // Initial y-coordinate
float targetX = 70.0; // Destination x-coordinate
float targetY = 80.0; // Destination y-coordinate
float easing = 0.05; // Size of each step along the path

void setup() {
    size(100, 100);
    noStroke();
    smooth();
}

void draw() {
    fill(0, 12);
    rect(0, 0, width, height);
    float d = dist(x, y, targetX, targetY);
    if (d > 1.0) {
        x += (targetX - x) * easing;
        y += (targetY - y) * easing;
    }
    fill(255);
    ellipse(x, y, 20, 20);
}
The simple curves covered in Lecture 4 can provide paths for shapes in motion. Instead of drawing the entire curve in one frame, it’s possible to calculate each step of the curve on consecutive frames. The following example presents a very general way to write this code. Changing the variables at the top of the code changes the start and stop position, the curve shape, and the number of steps to take along the curve.
float beginX = 20.0; // Initial x-coordinate
float beginY = 10.0; // Initial y-coordinate
float endX = 70.0; // Final x-coordinate
float endY = 80.0; // Final y-coordinate
float distX; // X-axis distance to move
float distY; // Y-axis distance to move
float exponent = 0.5; // Determines the curve
float x = 0.0; // Current x-coordinate
float y = 0.0; // Current y-coordinate
float step = 0.01; // Size of each step along the path
float pct = 0.0; // Percentage traveled (0.0 to 1.0)
void setup() {
    size(100, 100);
    noStroke();
    smooth();
    distX = endX - beginX;
    distY = endY - beginY;
}

void draw() {
    fill(0, 2);
    rect(0, 0, width, height);
    pct += step;
    if (pct < 1.0) {
        x = beginX + (pct * distX);
        y = beginY + (pow(pct, exponent) * distY);
    }
    fill(255);
    ellipse(x, y, 20, 20);
}
int direction = 1;

void draw() {
    fill(0, 2);
    rect(0, 0, width, height);
    pct += step * direction;
    if ((pct > 1.0) || (pct < 0.0)) {
        direction = direction * -1;
    }
    if (direction == 1) {
        x = beginX + (pct * distX);
        float e = pow(pct, exponent);
        y = beginY + (e * distY);
    } else {
        x = beginX + (pct * distX);
        float e = pow(1.0 - pct, exponent * 2);
        y = beginY + (e * -distY) + distY;
    }
    fill(255);
    ellipse(x, y, 20, 20);
}
Speed Along Curves

- As a shape moves along a curve, its speed changes.
- A curve can be used to control the speed of a visual element that moves on a straight line.
void draw() {
    fill(0, 2);
    rect(0, 0, width, height);
    if (pct < 1.0) {
        pct = pct + step;
        float rate = pow(pct, exponent);
        x = beginX + (rate * distX);
        y = beginY + (rate * distY);
    }
    fill(255);
    ellipse(x, y, 20, 20);
}

The transformation functions can also create motion by changing the parameters to `translate()`, `rotate()`, and `scale()`.

Transformations reset at the beginning of each `draw()`. Calling `translate(5,0)` will always move the coordinate system 5 pixels to the right in each frame. It will not move the system 5 right on the first frame, 10 on the next, 15 on the next etc.
float y = 50.0;
float speed = 1.0;
float radius = 15.0;

void setup() {
  size(100, 100);
  smooth();
  noStroke();
  ellipseMode(RADIUS);
}

void draw() {
  fill(0, 12);
  rect(0, 0, width, height);
  fill(255);
  translate(0, y); // Set the y-coordinate of the circle
  ellipse(33, 0, radius, radius);
  y += speed;
  if (y > height + radius) {
    y = -radius;
  }
}
float angle = 0.0;

void setup() {
    size(100, 100);
    smooth();
    noStroke();
}

void draw() {
    fill(0, 12);
    rect(0, 0, width, height);
    fill(255);
    angle = angle + 0.02;
    translate(70, 40);
    rotate(angle);
    rect(-30, -30, 60, 60);
}
Mechanical Motion

- The \( \sin() \) function is often used to produce elegant motion.
- It can generate an accelerating and decelerating speed as a shape moves from one frame to another.
float angle = 0.0; // Current angle
float speed = 0.1; // Speed of motion
float radius = 40.0; // Range of motion

void setup() {
  size(100, 100);
  noStroke();
  smooth();
}

void draw() {
  fill(0, 12);
  rect(0, 0, width, height);
  fill(255);
  angle += speed;
  float sinval = sin(angle);
  float yoffset = sinval * radius;
  ellipse(50, 50 + yoffset, 80, 80);
}
Periodic Motion

- Adding values from \( \sin() \) and \( \cos() \) can produce more complex movement that remains periodic.
- In the following example, a small dot moves in a circular pattern using values from \( \sin() \) and \( \cos() \). A larger dot uses the same values for its base position but adds additional \( \sin() \) and \( \cos() \) calculations to produce an offset.
float angle = 0.0; // Current angle
float speed = 0.05; // Speed of motion
float radius = 30.0; // Range of motion
float sx = 2.0;
float sy = 2.0;

void setup() {
  size(100, 100);
  noStroke();
  smooth();
}
void draw() {
    fill(0, 4);
    rect(0, 0, width, height);
    angle += speed; // Update the angle
    float sinval = sin(angle);
    float cosval = cos(angle);

    // Set the position of the small circle based on new
    // values from sine and cosine
    float x = 50 + (cosval * radius);
    float y = 50 + (sinval * radius);
    fill(255);
    ellipse(x, y, 2, 2); // Draw smaller circle

    // Set the position of the large circles based on the
    // new position of the small circle
    float x2 = x + cos(angle * sx) * radius / 2;
    float y2 = y + sin(angle * sy) * radius / 2;
    ellipse(x2, y2, 6, 6); // Draw larger circle
}

Phase Shifting

- The phase of a function is one iteration through its possible values—for example, a single rise and fall sequence of a sine curve.
- Phase shifting occurs when the function is offset to start at a different point within the phase, such as the downward part of a sine curve rather than the top.
float angle = 0.0;
float speed = 0.1;

void setup() {
    size(100, 100);
    noStroke();
    smooth();
}

void draw() {
    background(0);
    angle = angle + speed;
    ellipse(50 + (sin(angle + PI) * 5), 25, 30, 30);
    ellipse(50 + (sin(angle + HALF_PI) * 5), 55, 30, 30);
    ellipse(50 + (sin(angle + QUARTER_PI) * 5), 85, 30, 30);
}
Examples of organic movement include a leaf falling, an insect walking, a bird flying, a person breathing, a river flowing, and smoke rising.

This type of motion is often considered idiosyncratic and stochastic.

It is often non-periodic or “almost” periodic.
```cpp
float x = 50.0; // X-coordinate
float y = 80.0; // Y-coordinate

void setup() {
  size(100, 100);
  randomSeed(0); // Force the same random values
  background(0);
  stroke(255);
}

void draw() {
  x += random(-2, 2); // Assign new x-coordinate
  y += random(-2, 2); // Assign new y-coordinate
  point(x, y);
}
```
The \texttt{sin()} and \texttt{cos()} functions can be used to create unpredictable motion when employed with the \texttt{random()} function.

The following example the position of the line at each frame is based on its current position and the slight variation to its direction. The \texttt{cos()} function uses the angle to calculate the next \texttt{x}-coordinate for the line, and the \texttt{sin()} function uses the same angle to calculate the next \texttt{y}-coordinate.
float x = 0.0; // X-coordinate
float y = 50.0; // Y-coordinate
float angle = 0.0; // Direction of motion
float speed = 0.5; // Speed of motion

void setup() {
    size(100, 100);
    background(0);
    stroke(255, 130);
    randomSeed(121); // Force the same random values
}

void draw() {
    angle += random(-0.3, 0.3);
    x += cos(angle) * speed; // Update x-coordinate
    y += sin(angle) * speed; // Update y-coordinate
    translate(x, y);
    rotate(angle);
    line(0, -10, 0, 10);
}
Animating Transformations

- In the following example, the angle variable for the `tail()` function is continually changing to produce a swaying motion.
- Because the angles for each shape accumulate with each unit, the longest shapes with the most units swing from side to side with a greater curvature.
```cpp
void draw() {
    background(0);
    inc += 0.01;
    float angle = sin(inc) / 10.0 + sin(inc * 1.2) / 20.0;
    tail(18, 9, angle / 1.3);
    tail(33, 12, angle);
    tail(44, 10, angle / 1.3);
    tail(62, 5, angle);
    tail(88, 7, angle*2);
}

void tail(int x, int units, float angle) {
    pushMatrix();
    translate(x, 100);
    for (int i = units; i > 0; i--) {
        strokeWeight(i);
        line(0, 0, 0, -8);
        translate(0, -8);
        rotate(angle);
    }
    popMatrix();
}
```
Lab Exercises
Lab Exercises

- Move two shapes continuously, but constrain their positions to the display window.
- Move three shapes on different curves to create a kinetic composition.
- Use the transformation functions to animate a shape.
Lab Exercises

› Make a shape move with numbers returned from sin() and cos().

› Develop a kinetic composition using the concept of phase shifting.

› Use code 32-06 as a base for creating a more advanced organism.