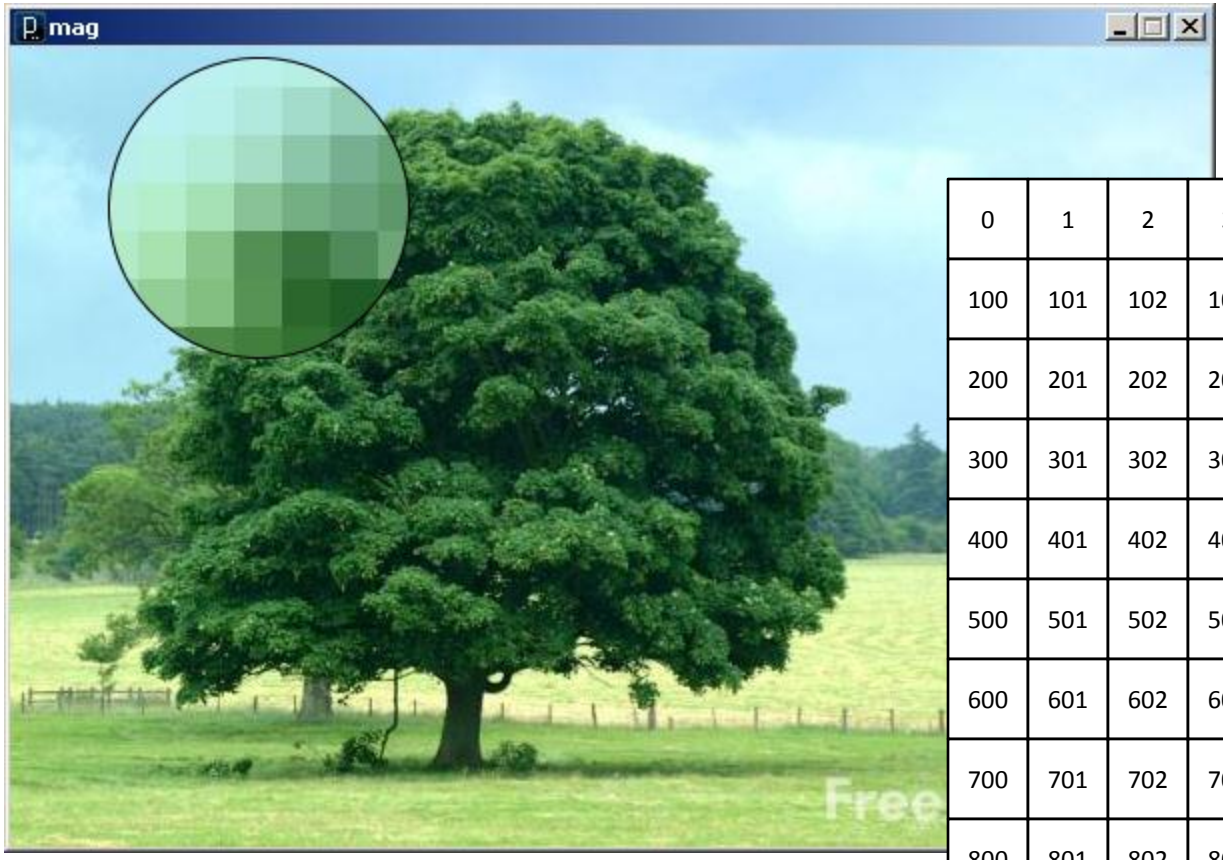
Exam 2 Review

Image Processing, Transformations, Recursion

An image is an array of colors



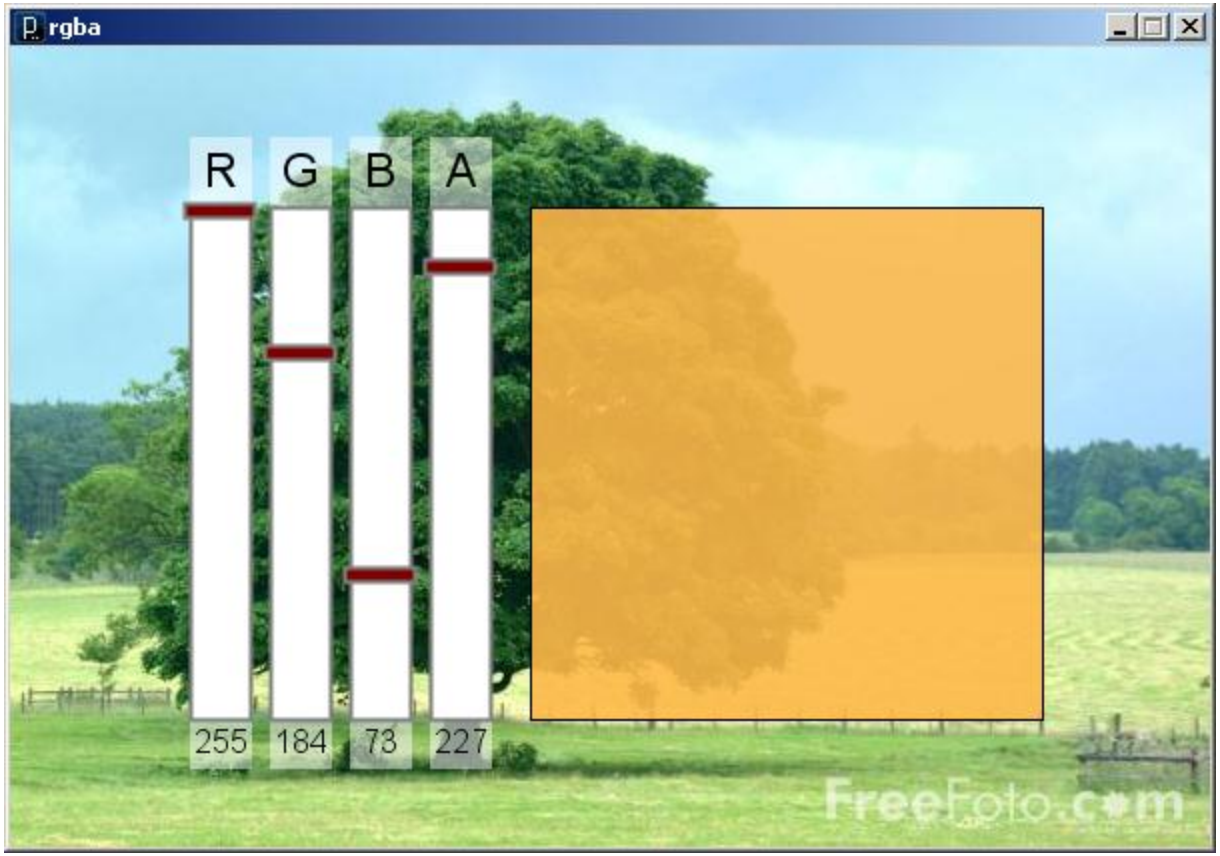
0	1	2	3		...	98	99
100	101	102	103		...	198	199
200	201	202	203		...	298	299
300	301	302	303		...	398	399
400	401	402	403		...	498	499
500	501	502	503		...	598	599
600	601	602	603		...	698	699
700	701	702	703		...	798	799
800	801	802	803		...	898	899
⋮	⋮	⋮	⋮		...	⋮	⋮

Pixel : Picture Element

mag.pde

Color

- A triple of bytes [0, 255]
  - RGB or HSB
- Transparency (alpha)
  - How to blend a new pixel color with an existing pixel color



rgba.pde

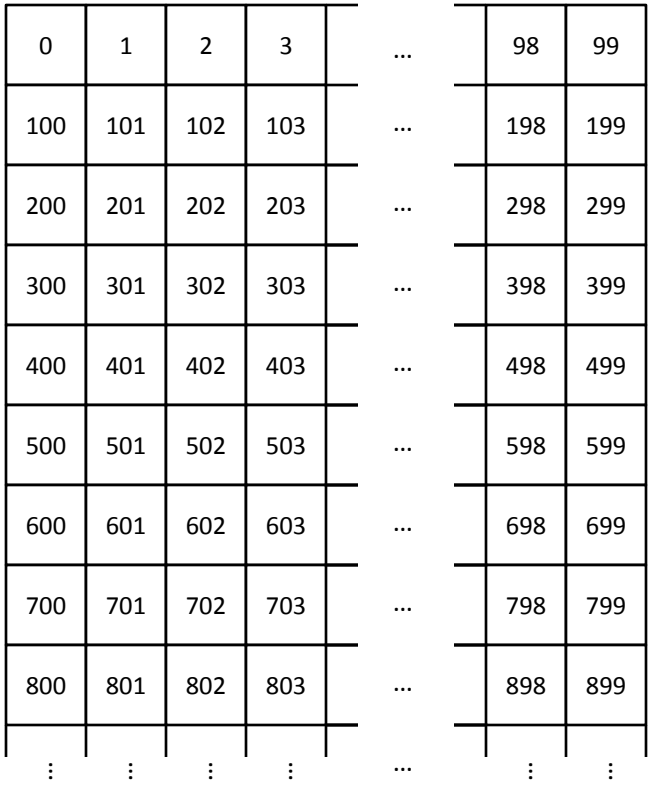
Accessing the pixels of a sketch

- loadPixels()
  - Copies the color data out of the sketch window into a 1D array of colors named pixels[]
  - The pixels[] array can be modified

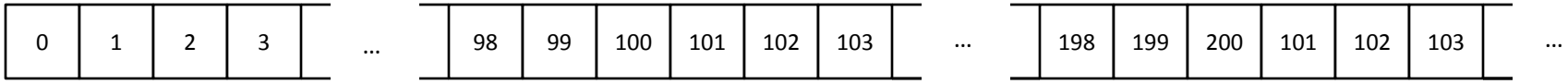
- updatePixels()
  - Copies the color data from the pixels[] array back to the sketch window

A 100-pixel wide image

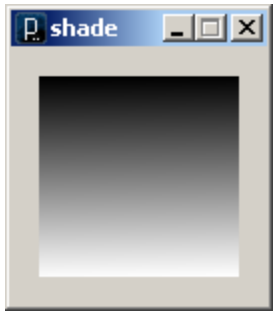
- First pixel at index 0
- Right-most pixel in first row at index 99
- First pixel of second row at index 100

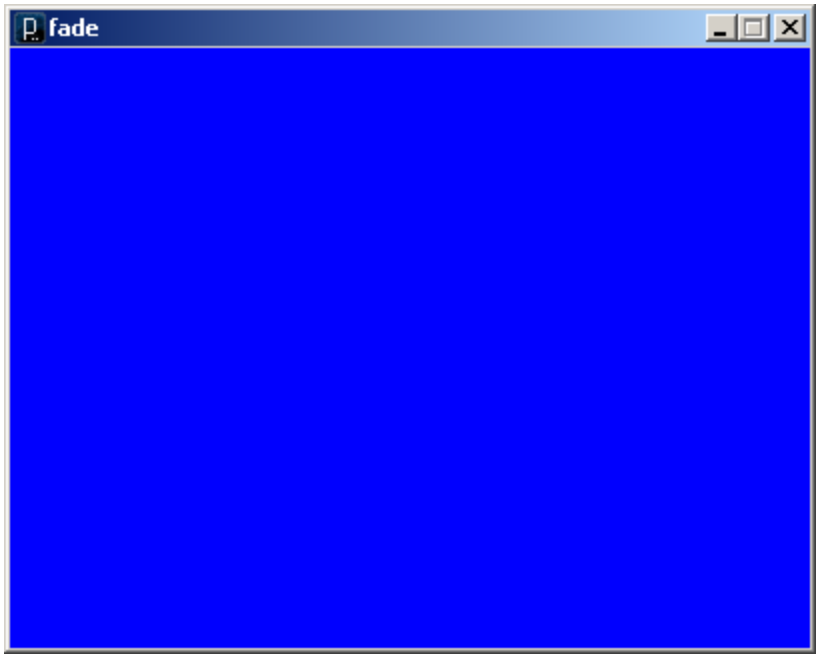


The pixels[] array is one-dimensional



```
// shade
void setup() {
  size(100, 100);

  float b = 0.0;

  // Load colors into the pixels array
  loadPixels();

  // Fill pixel array with a grayscale value
  // based on pixel array index
  for (int i=0; i<pixels.length; i++) {
    b = map(i, 0, 10000, 0, 255);
    pixels[i] = color(b);
  }

  // Update the sketch with pixel data
  updatePixels();
}
```



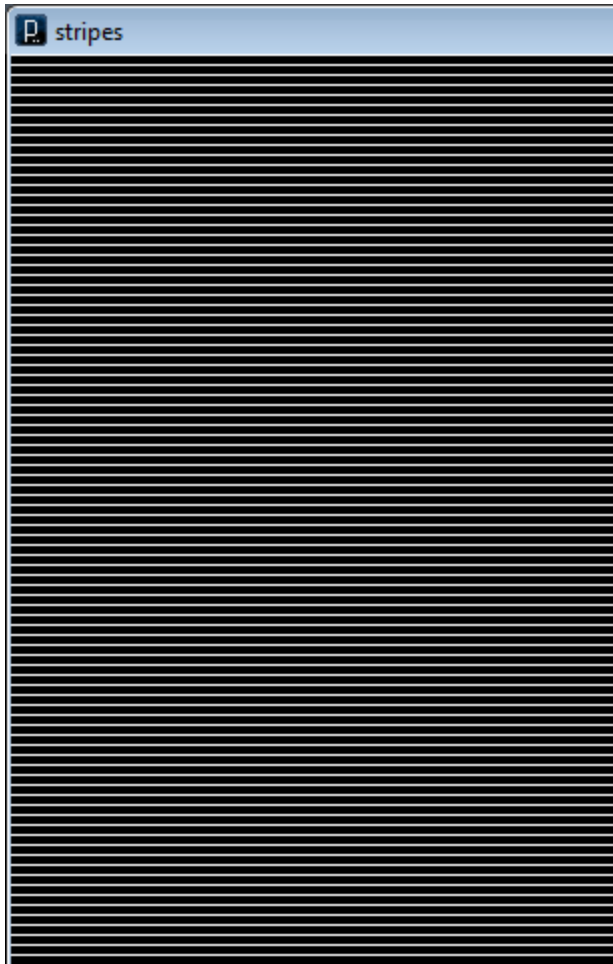



```
// fade red to blue
void setup() {
  size(400, 300);
  background(255, 0, 0);
}

void draw() {

  // Load colors into the pixels array
  loadPixels();

  for (int i=0; i<pixels.length; i++)
  {
    color p = pixels[i];        // Get color from pixels array

    float r = red( p );         // Extract color components
    float g = green( p );
    float b = blue( p );

    r = r * 0.99;               // Fade red to blue
    b = 255-r;

    pixels[i] = color(r,g,b);   // Rebuild and replace color
  }

  // Update the sketch with pixel data
  updatePixels();
}
```

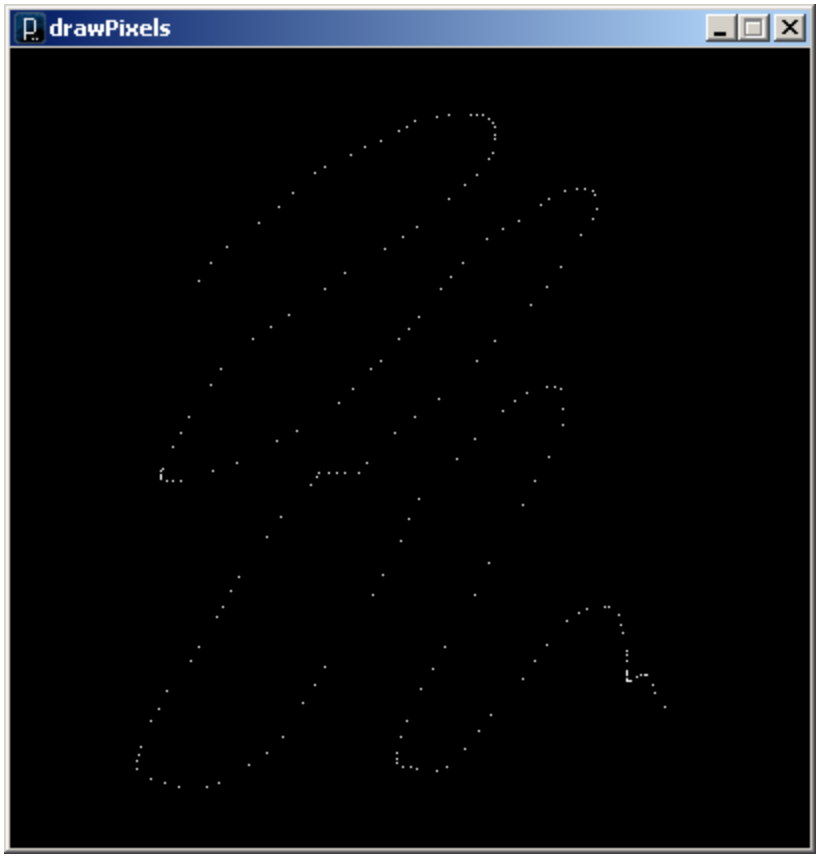
fade.pde

Accessing pixel colors with row and column

```
void setup() {
  size(500, 500);
  background(0);

  loadPixels();

  for (int r=0; r<height; r++) {
    for (int c=0; c<width; c++) {
      if (r%5 == 0) {
        int i = r*width + c;
        pixels[i] = color(255);
      }
    }
  }

  updatePixels();
}
```



```
// drawPixels
void setup() {
  size(400, 400);
  background(0);
}

void draw() {
  if (mousePressed == true)
  {
    // Load colors into the
    // pixels array
    loadPixels();

    // Compute pixel location
    // from mouse coordinates
    int i = mouseY*width + mouseX;

    // Set pixel color
    pixels[i] = color(255);

    // Update the sketch with pixel data
    updatePixels();
  }
}
```

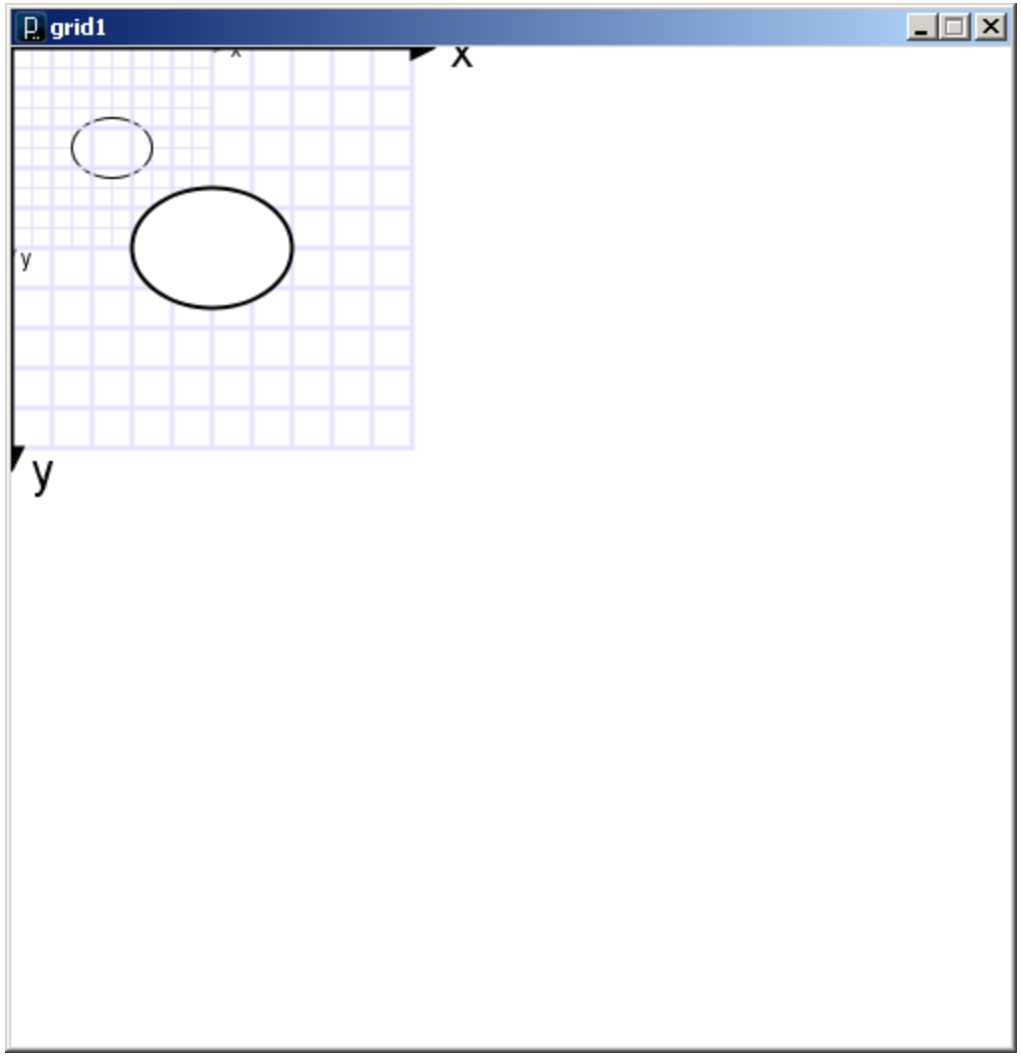


drawPixels.pde

Three ways to transform the coordinate system:

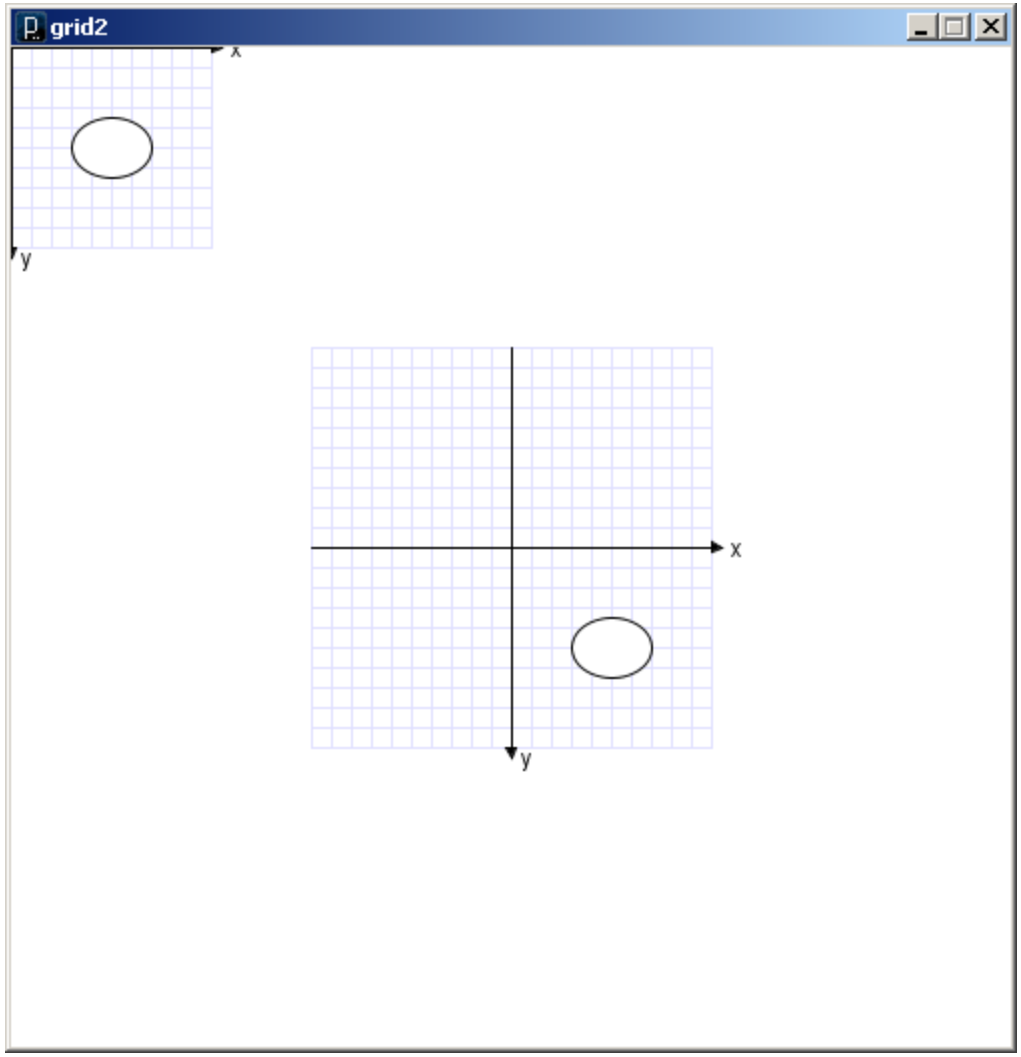
1. **Translate**
   - Move axes left, right, up, down …
2. **Scale**
   - Magnify, zoom in, zoom out …
3. **Rotate**
   - Tilt clockwise, tilt counter-clockwise …

```
void draw() {
  grid();
  fill(255);
  ellipse(50,50,40,30);

  scale(2,2);
  grid();
  fill(255);
  ellipse(50,50,40,30);
}
```

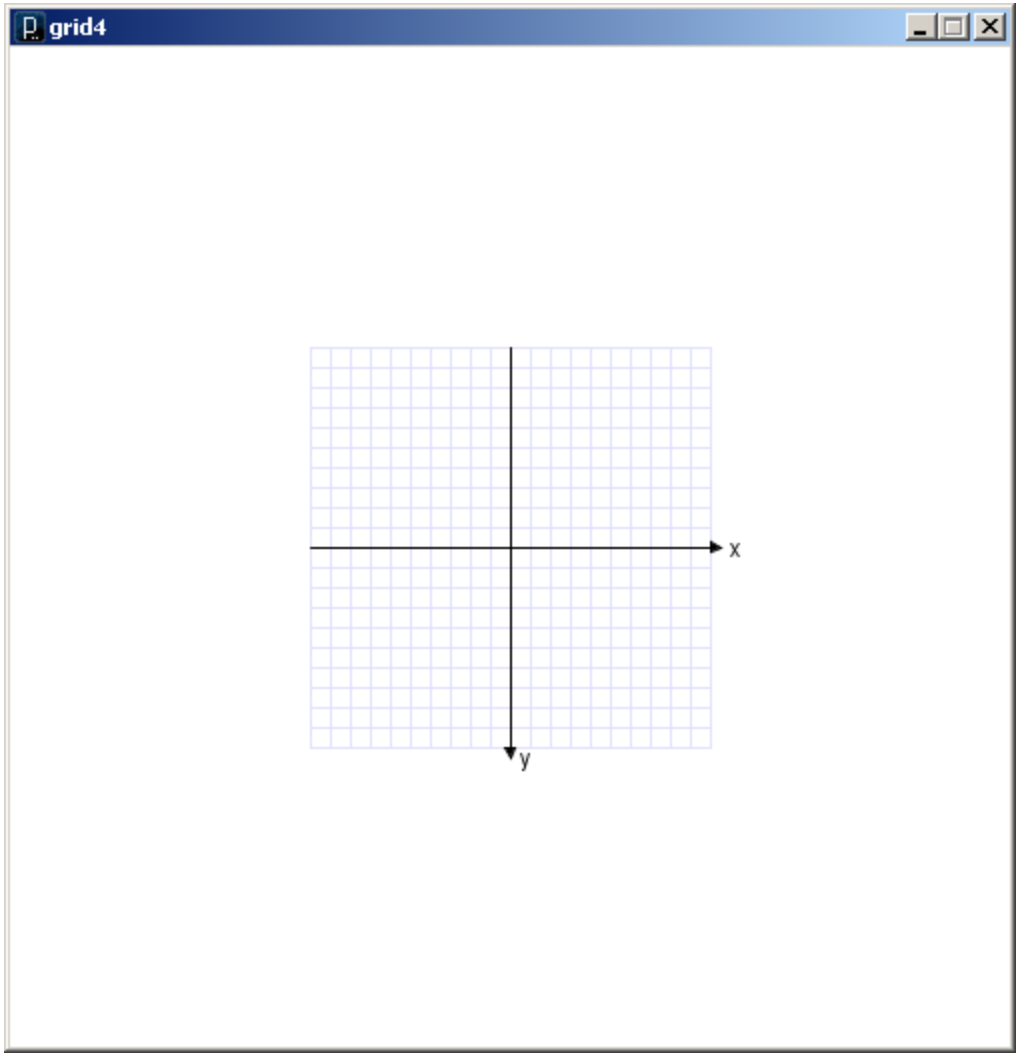


grid1.pde

```
void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);

  translate(250, 250);
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
```

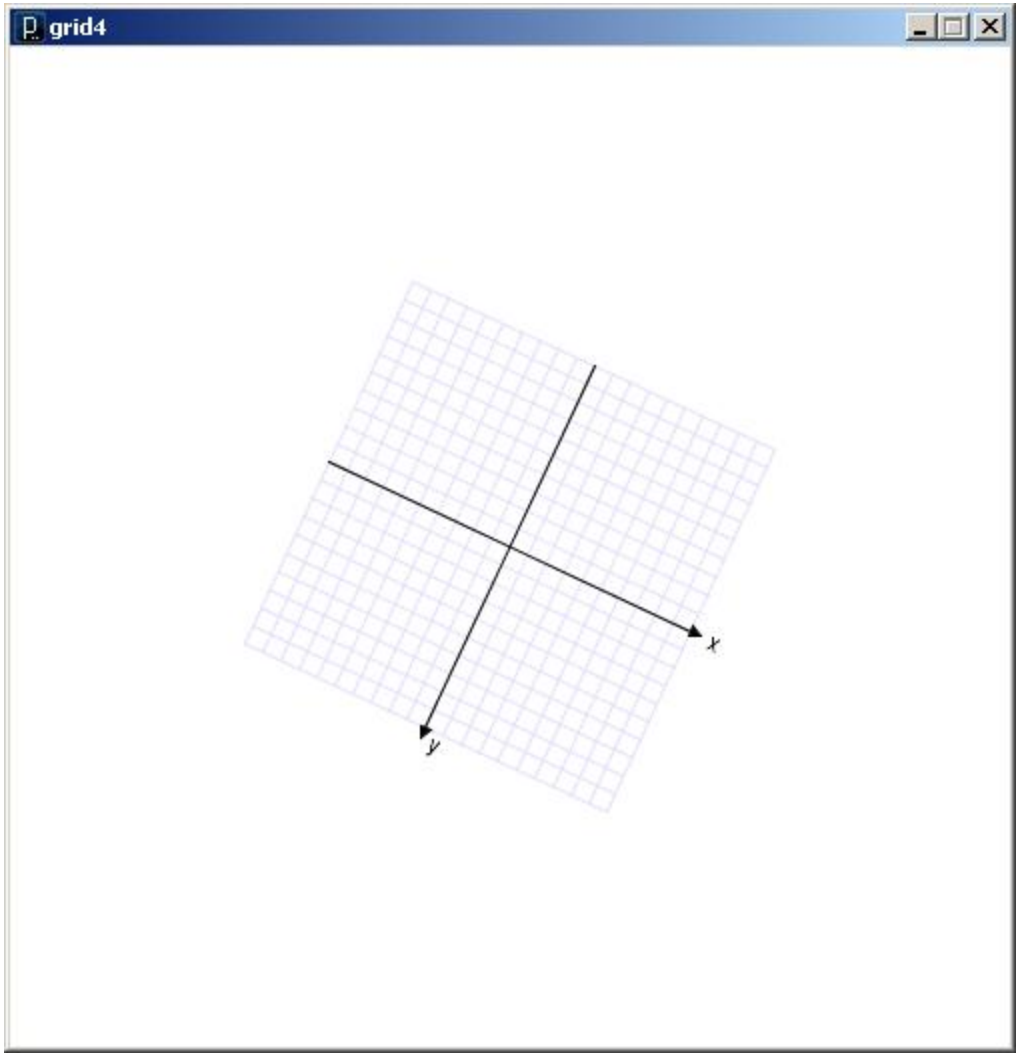


```
void draw() {
  translate(250.0, 250.0);
  //rotate( radians(25.0) );
  //scale( 2 );
  grid();
}
```



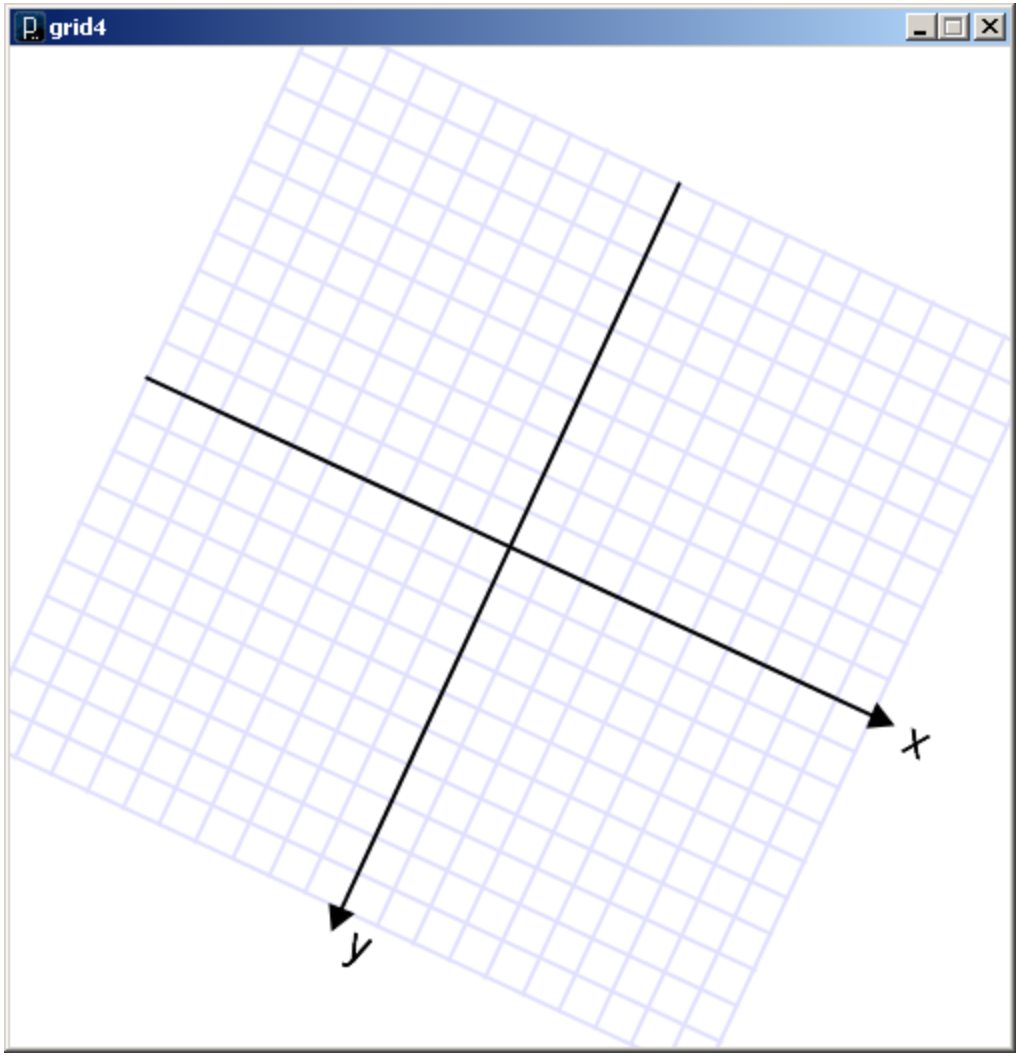
grid4.pde

```
void draw() {
  translate(250.0, 250.0);
  rotate( radians(25.0) );
  //scale( 2 );
  grid();
}
```



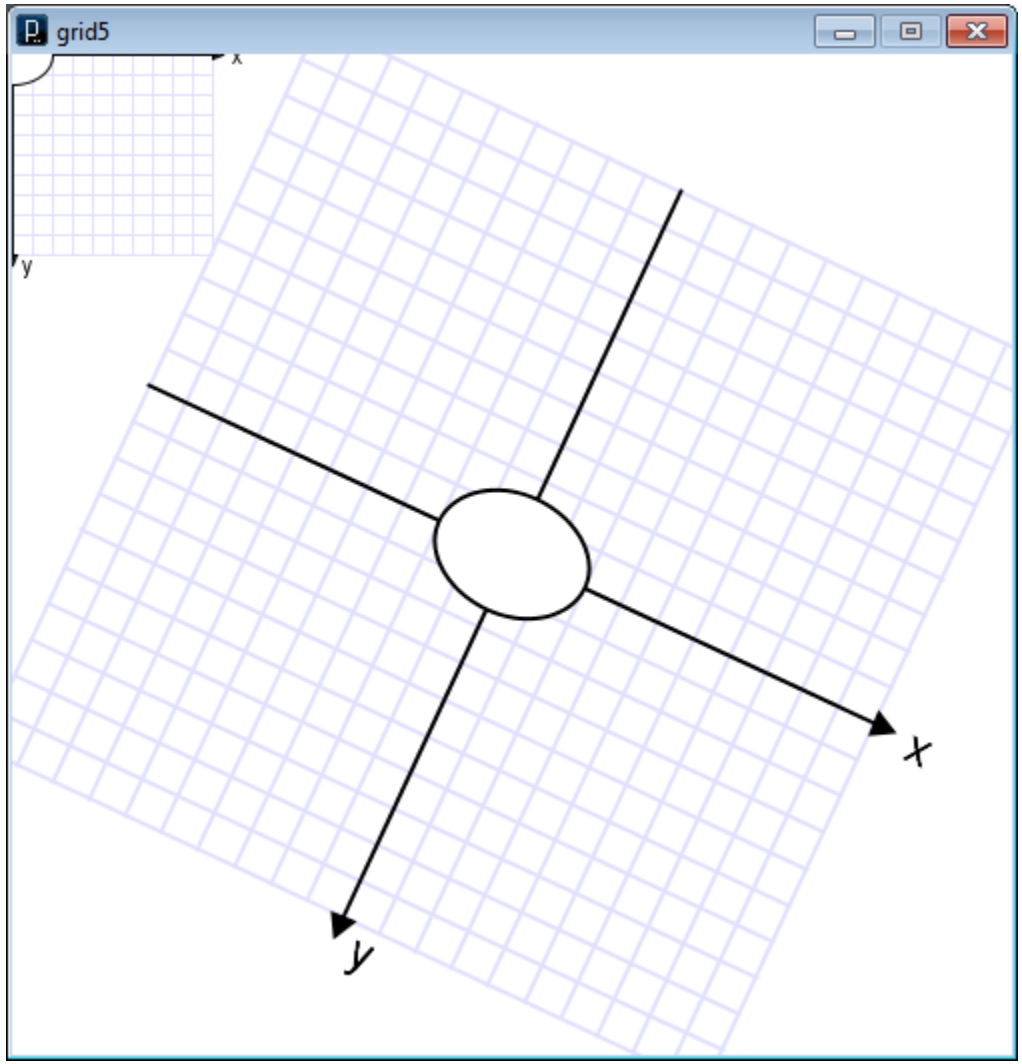
grid4.pde

```
void draw() {
  translate(250.0, 250.0);
  rotate(radians(25.0) );
  scale( 2 );
  grid();
}
```



grid4.pde

```
void draw() {
 grid();
  fill(255);
  ellipse(0, 0, 40, 30);

  translate(250.0, 250.0);
  rotate( radians(25.0) );
  scale(2);
  grid();
  fill(255);
  ellipse(0, 0, 40, 30);
}
```



grid5.pde

Transformation Matrix Stack

- Transformation matrices can be managed using the **Matrix Stack.** (Recall, a stack is LIFO)
- The current transformation matrix can be temporarily pushed on to the Matrix Stack, and popped off for use later on.
- The Matrix Stack can hold multiple transformation matrices.
- Enables the idea of recursive drawing coordinate systems
    - … when you want to draw a part of something w.r.t. that something's master coordinate system

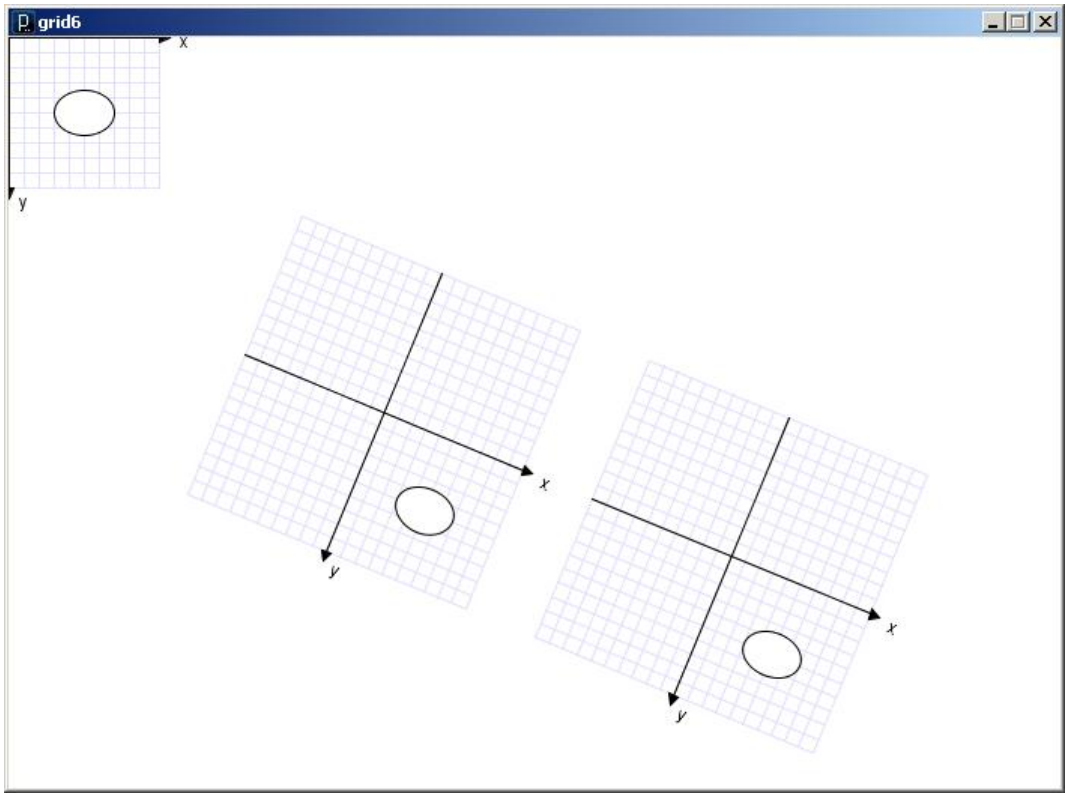
pushMatrix()

- Pushes a copy of the current transformation matrix onto the Matrix Stack

popMatrix()

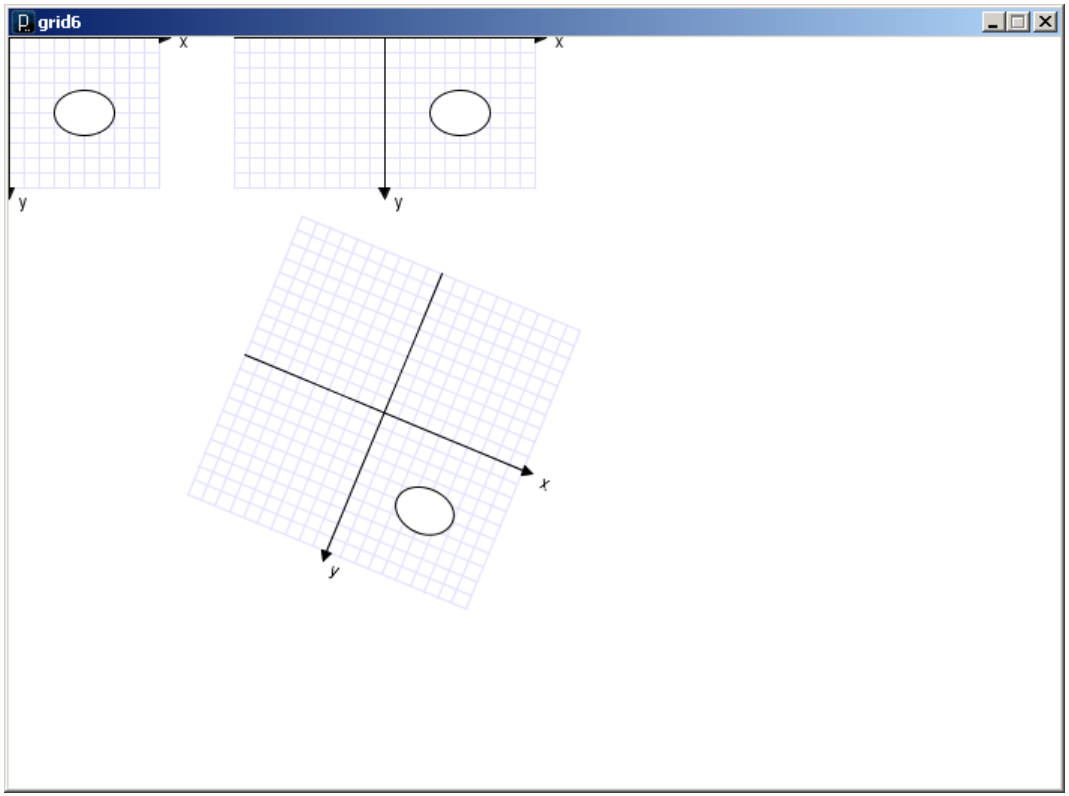
- Pops the last pushed transformation matrix off the Matrix Stack and replaces the current matrix

```
void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);


  translate(250, 250);
  rotate(PI/8.0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);


  translate(250, 0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
```



relXform.pde

```
void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);

  pushMatrix();
  translate(250, 250);
  rotate(PI/8.0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
  popMatrix();

  translate(250, 0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
```



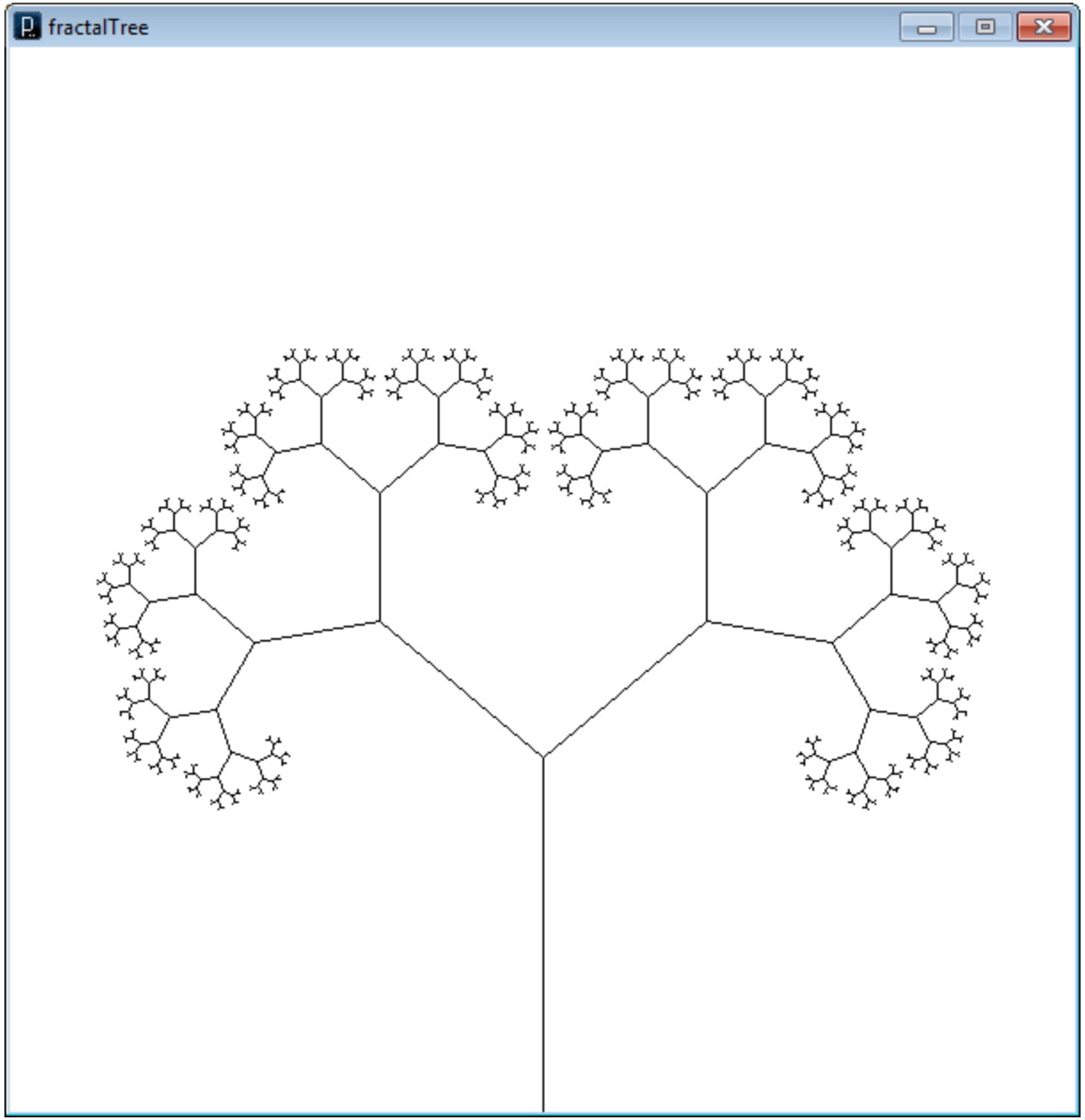
relXform.pde

Some things to remember:

1. Transformations are cumulative.
2. All transformations are cancelled each time draw() exits.
   - They must be reset each time at the beginning of draw() before any drawing.
3. Rotation angles are measured in radians
   - π radians = 180°
   - radians = (PI/180.0) * degrees
4. Order matters

Recursive Function Model

```
type recursiveFunction( type arg ) {

  if ( base case ) {
    return something;
  } else {

    // supporting statements here

    type val = recursiveFunction( another arg );
    return val;
  }

}
```



fractalTree.pde

```
class FractalTree {
  float len;

  FractalTree left;    // Left branch
  FractalTree right;   // Right branch

  FractalTree(float len, int depth) {
    this.len = len;

    if (depth > 1) {
      depth--;
      left  = new FractalTree(0.6*len, depth);
      right = new FractalTree(0.6*len, depth);
    }
  }
```

```
  void draw(float angle) {
    stroke(0);
    line(0, 0, 0, len);
    if (left != null && right != null) {
      translate(0, len);
      pushMatrix();
      rotate( radians(angle) );
      left.draw(angle);
      popMatrix();
      pushMatrix();
      rotate( radians(-angle) );
      right.draw(angle);
      popMatrix();
    }
  }
}
```

fractalTree.pde

```
FractalTree f;

void setup() {
  size(600, 600);
  smooth();
  background(255);

  f = new FractalTree(-200, 10);

  translate(300, 600);
  f.draw(50);

  println(f.countBranches());
}
```

Write a recursive method named countBranches() that returns the number of branches of the FractalTree()