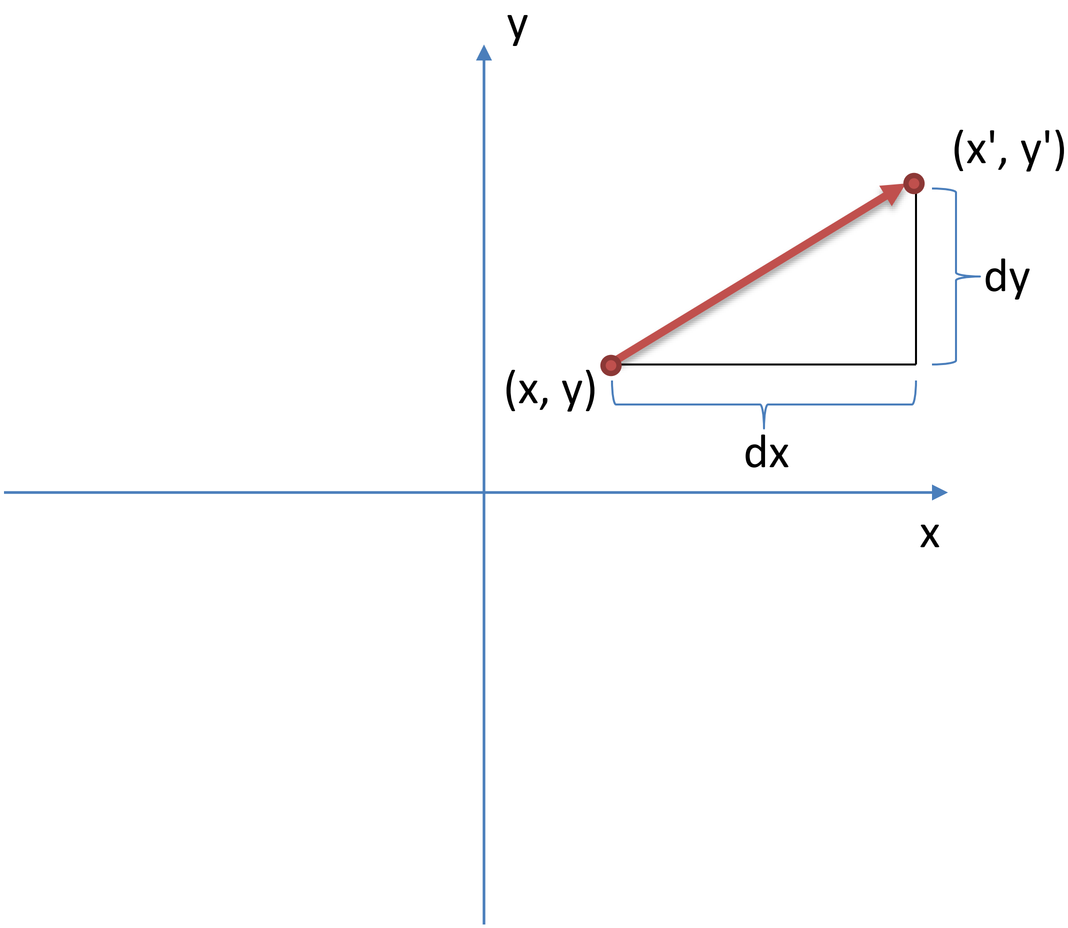
Review

- Transformations
  - Scale
  - Translate
  - Rotate
- Combining Transformations
  - Transformations are cumulative
  - Flipping the y-axis direction
  - Rotating about the center of an object

translate

$x' = x + dx$

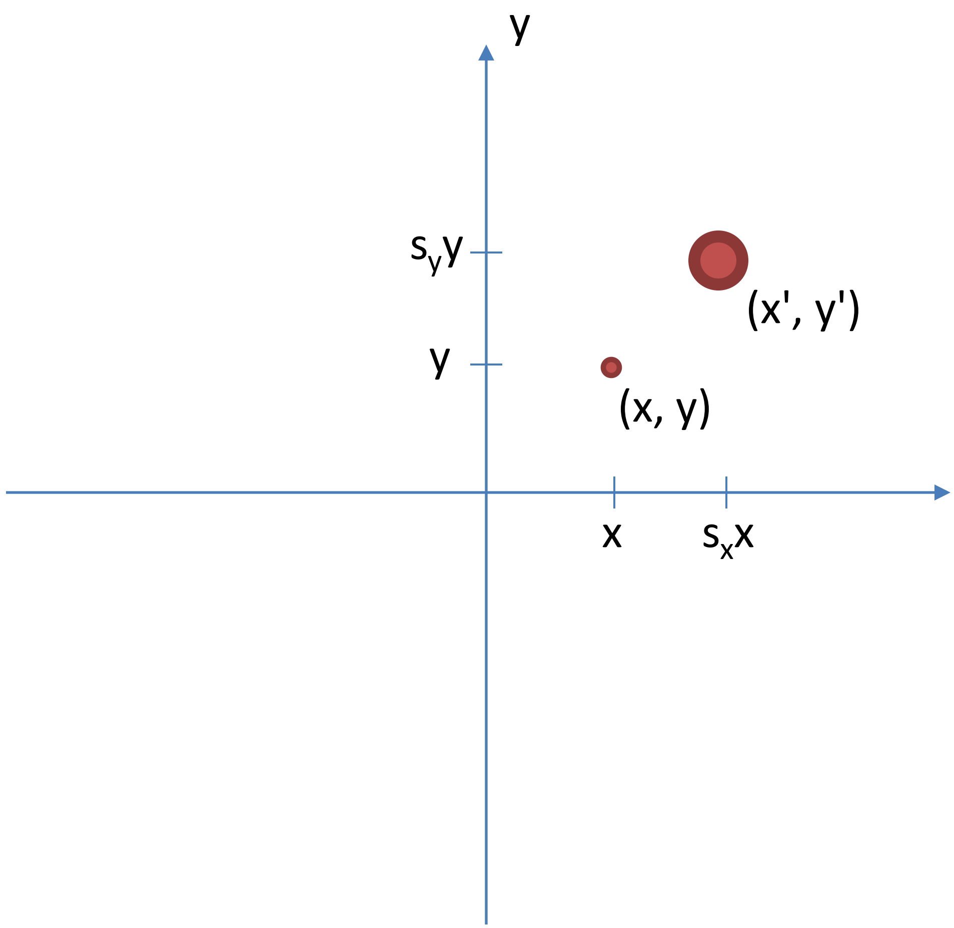
$y' = y + dy$



scale

$$x' = s_x \cdot x$$

$$y' = s_y \cdot y$$

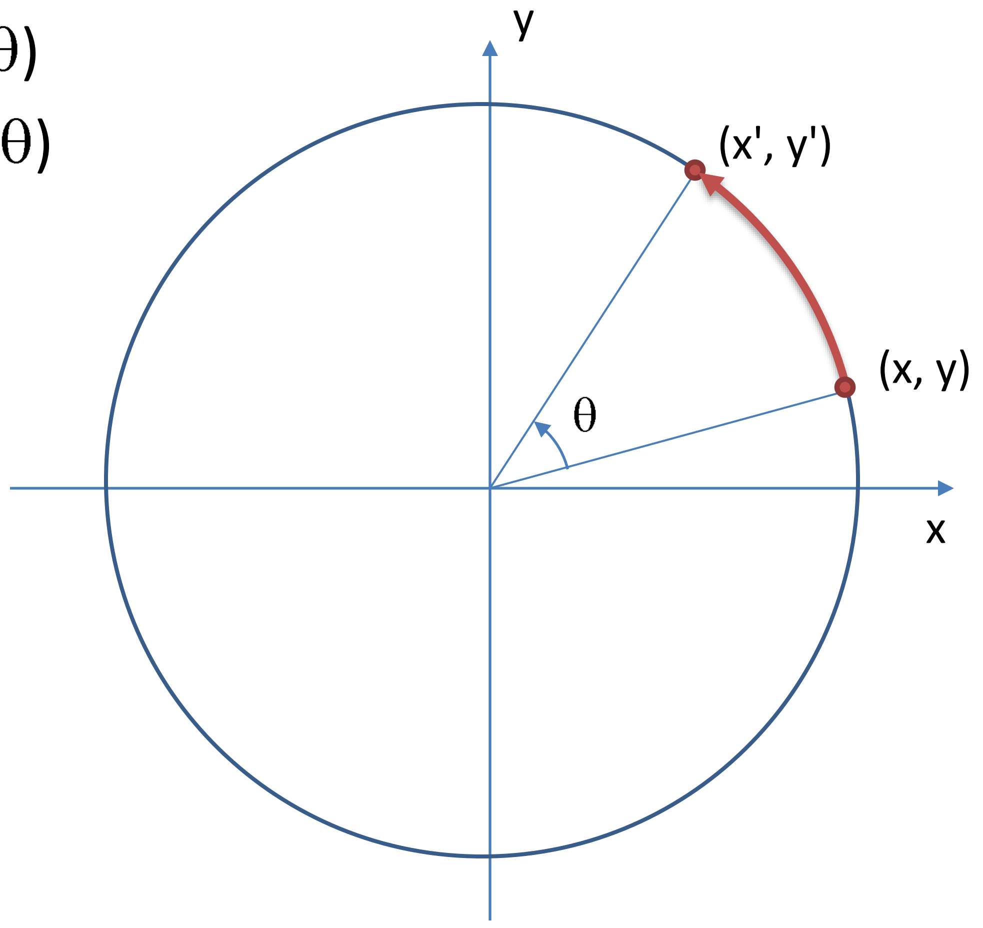


* Watch out. A scale transformation may cause objects to grow **and move**.

rotate

$$x' = x \cos(\theta) - y \sin(\theta)$$

$$y' = x \sin(\theta) + y \cos(\theta)$$



Homogeneous Translation

- The translation of a point by (dx, dy) can be written in matrix form as:

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

- Representing the point as a homogeneous column vector we perform the calculation as:

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \times x + 0 \times y + dx \times 1 \\ 0 \times x + 1 \times y + dy \times 1 \\ 0 \times x + 0 \times y + 1 \times 1 \end{bmatrix} = \begin{bmatrix} x + dx \\ y + dy \\ 1 \end{bmatrix}$$

Recall Matrix Multiplication

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \times x + b \times y + c \times z \\ d \times x + e \times y + f \times z \\ g \times x + h \times y + i \times z \end{bmatrix}$$

Homogeneous Scaling

- The scaling of a point by (s_x, s_y) can be written in matrix form as:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Representing the point as a homogeneous column vector we perform the calculation as:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \times x \\ s_y \times y \\ 1 \end{bmatrix}$$

Homogeneous Rotation

- The rotation of a point about the origin by θ can be written in matrix form as:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Representing the point as a homogeneous column vector we perform the calculation as:

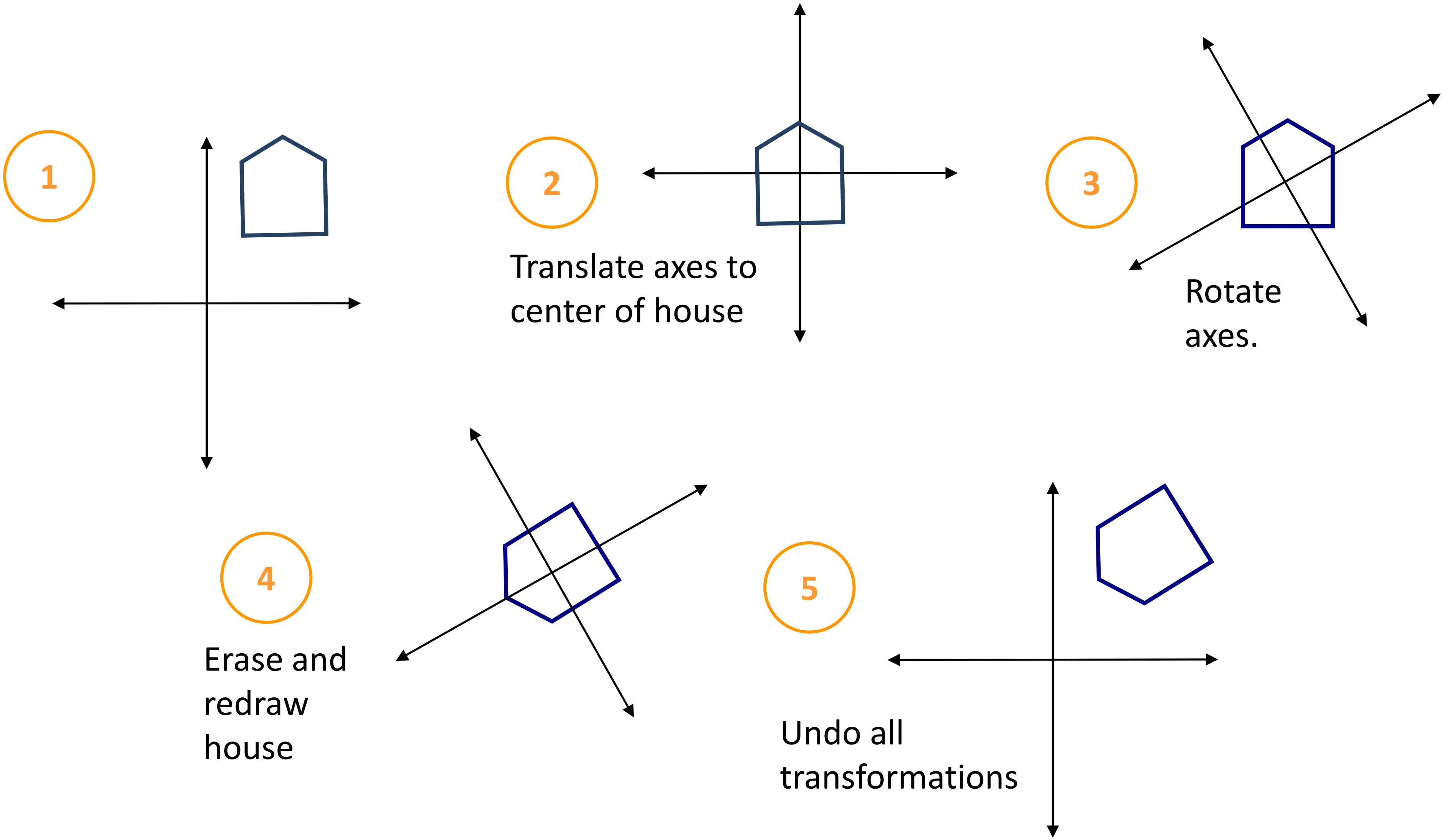
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta \times x - \sin\theta \times y \\ \sin\theta \times x + \cos\theta \times y \\ 1 \end{bmatrix}$$

Homogeneous Transformations

- Multiple transformations can be combined by pre-multiplying all transformation matrices in correct order.
- Pre-multiplied transformation matrix can be used to transform each point.

Draw a house rotated about it's center.

Combines a translation and a rotation



http://www.comp.dit.ie/bmacnamee

Pre-multiplied transformation matrix

Example: a translation followed by a rotation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{bmatrix}$$

Start with identity matrix

Multiply by translation matrix

Multiply by rotation matrix

Combined transformation matrix will be computed with predefined values for θ, dx and dy

Combined transformation matrix is applied to all points

$$\begin{bmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta x - \sin\theta y + dx \\ \sin\theta x + \cos\theta y + dy \\ 1 \end{bmatrix}$$

Transformations can easily be reversed using Inverse Transformations

Inverse Translation

$$T^{-1} = \begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Rotation

$$R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Scale

$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

* *In other words, to undo a transformation, multiply your transformation matrix by an inverse transformation matrix.*

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

resetMatrix()

- Replaces the current transformation matrix with the identity matrix

applyMatrix()

- Multiplies the current transformation matrix with a given custom matrix.

printMatrix()

- Prints the current transformation matrix in effect.

```
// space1

CelestialBody center;

void setup(){
  size(600, 600);
  smooth();
  ellipseMode(CENTER);

  center = new CelestialBody(
                 color(200), 10 );
}

void draw() {
  background(0);

  translate(0.5*width, 0.5*height);
  center.draw(0,0);

  center.update();
}
```

```
class CelestialBody {

  color fillColor;
  float diameter;

  CelestialBody(color clr, float diam){
    fillColor = clr;
    diameter = diam;
  }

  void update() {
  }

  void draw(float x, float y) {
    fill(fillColor);
    noStroke();
    translate(x, y);
    ellipse(0,0,diameter,diameter);
  }
}
```

Modify CelestialBody. Allow it to have its own orbiting CelestialBody.

```
class CelestialBody {

  color fillColor;
  float diameter;

  // Info about the orbiting body
  CelestialBody body;  // Orbiting body
  float orbit;         // Height of orbit
  float angle=0.0;     // Angle of orbit
  float dangle;        // Speed of orbit

  CelestialBody(color clr, float diam,
     CelestialBody b, float o, float da){

    fillColor = clr;
    diameter = diam;

    body = b;
    orbit = o;
    dangle = da;
  }

  …
```

```
void update() {
    // If there is an orbiting body
    if (body != null) {
      // Increment the orbiting body
      angle = (angle + dangle) % TWO_PI;
      body.update();
    }
  }

  void draw(float x, float y) {
    fill(fillColor);
    noStroke();
    translate(x, y);
    ellipse(0,0,diameter,diameter);

    // If there is an orbiting body
    if (body != null) {
      // Draw orbiting body wrt self
      pushMatrix();
      rotate(angle);
      body.draw(orbit, 0);
      popMatrix();
    }
  }
}
```

Create two CelestialBody objects – one orbiting the other.

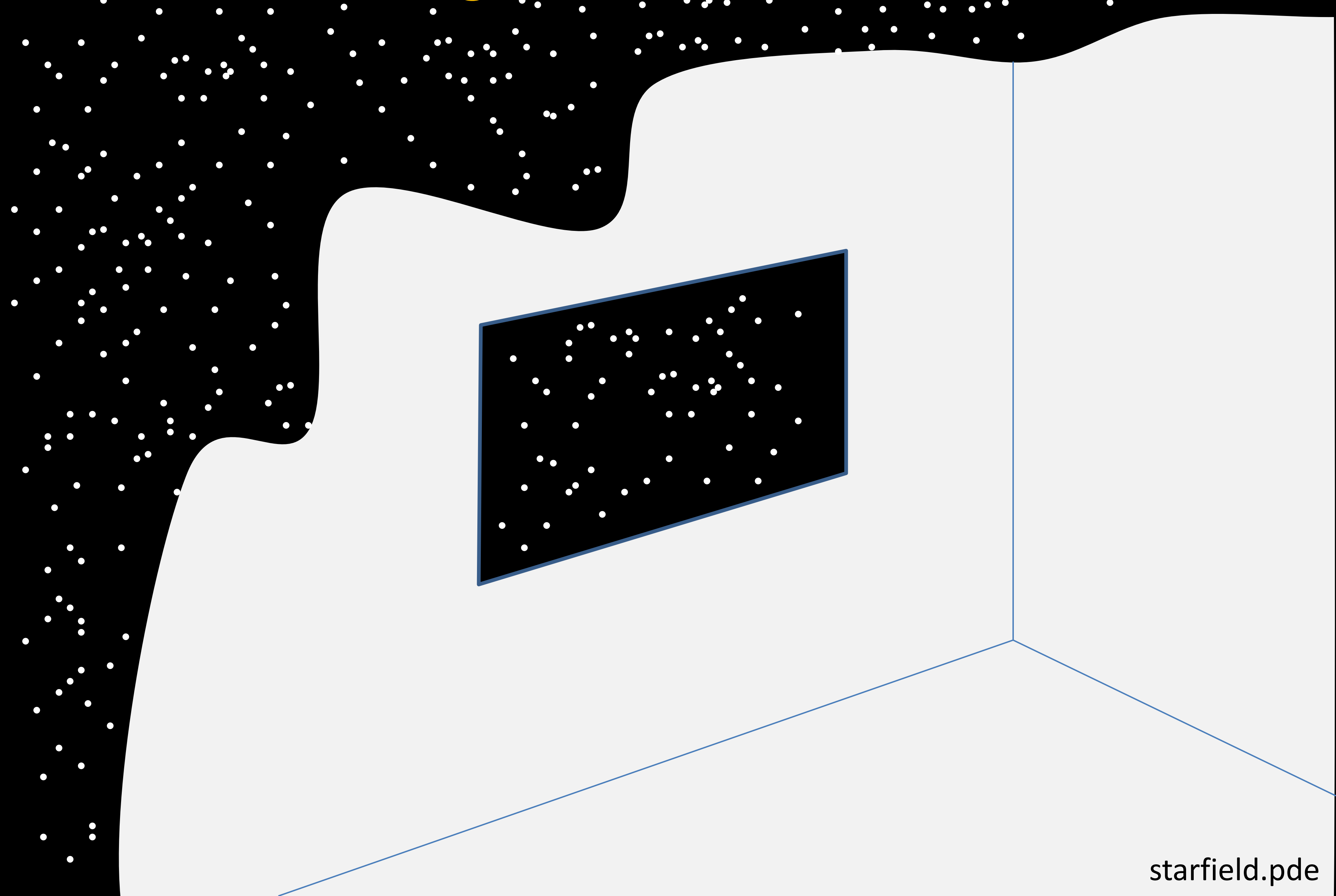
```
// space2

// Celestial body at the center of the universe
CelestialBody center;

void setup(){
  size(600, 600);
  smooth();
  ellipseMode(CENTER);

  // Create the moon with no orbiting body
  CelestialBody moon = new CelestialBody( color(200), 10, null, 0, 0 );

  // Create the center of the universe, with an orbiting moon
  center = new CelestialBody( color(127,127,255), 20, moon, 50, 0.05 );
}

void draw() {
  background(0);

  // Draw the center of the universe at the center of the sketch
  translate(0.5*width, 0.5*height);
  center.draw(0,0);

  // Update the center of the universe
  center.update();
}
```

Add the sun, orbited by the earth, orbited by the moon.

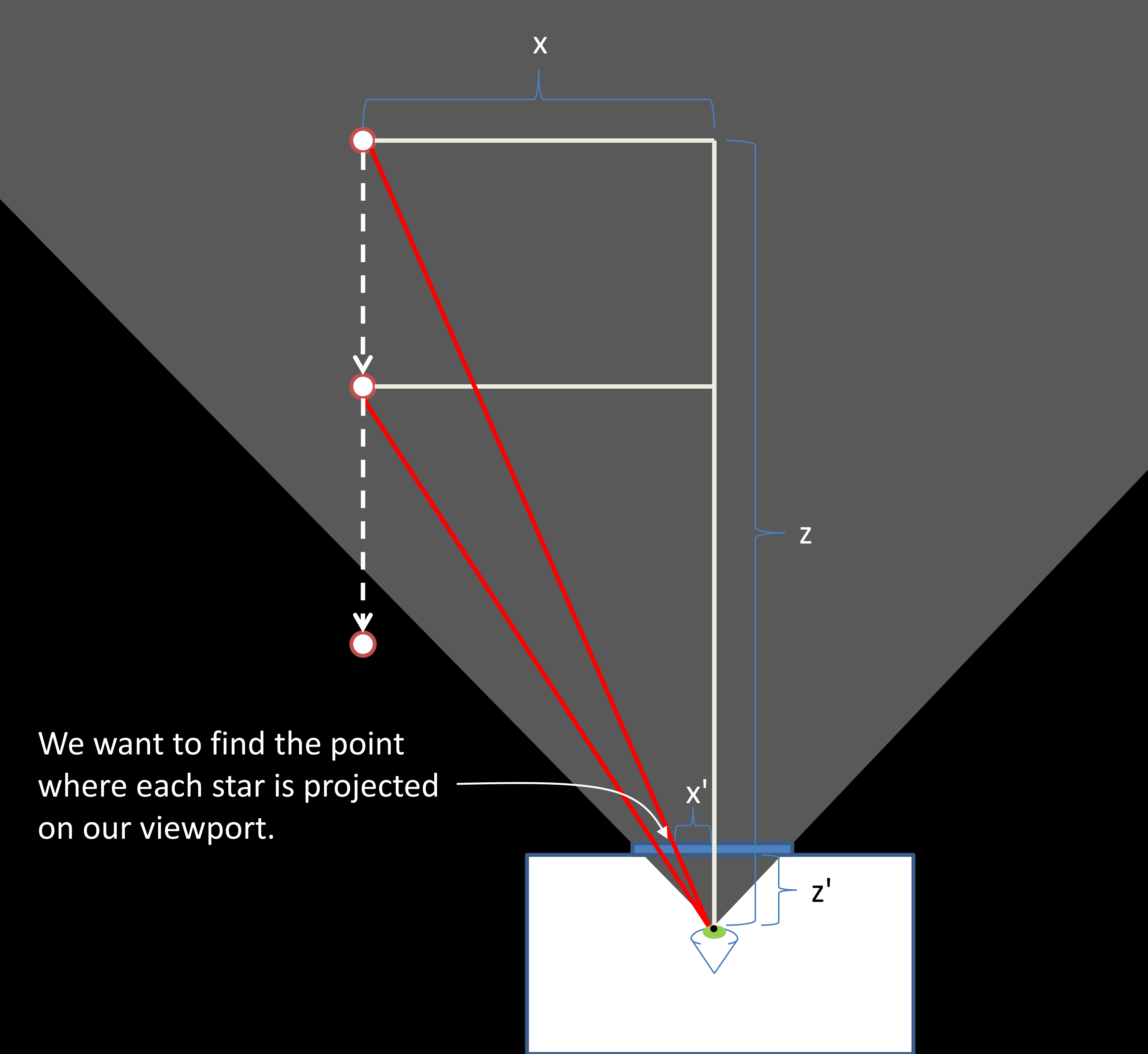
```
// space3

// Celestial body at the center of the universe
CelestialBody center;

void setup(){
  size(600, 600);
  smooth();
  ellipseMode(CENTER);

  // Create the moon with no orbiting body
  CelestialBody moon = new CelestialBody( color(200), 10, null, 0, 0 );

  // Create the earth with an orbiting moon
  CelestialBody earth = new CelestialBody(color(127,127,255), 20, moon, 50, 0.05);

  // Create the center of the universe, with an orbiting body
  center = new CelestialBody( color(255,255,127), 40, earth, 120, 0.02 );
}
```

- Each CelestialBody is in charge of drawing itself at the x,y location provided.
- If a CelestialBody has an orbiting body, it only needs to update the coordinates to the location of the orbiting body, and ask the orbiting body to draw itself.
- Non-trivial dynamics emerge by delegating to each object the job of implementing its own simple rules of motion.
- Note that orbiting objects can be complex
  - (See space5)

A starfield using matrix transformations



starfield.pde



$$\frac{x'}{z'} = \frac{x}{z}$$

$$x' = z'\left(\frac{x}{z}\right)$$

```
class Star {
  // Star coordinates in 3D
  float x;
  float y;
  float z;

  Star() {
    x = random(-5000, 5000);
    y = random(-5000, 5000);
    z = random(0, 2000);
  }

  void update() {
    // Move star closer to viewport
    z-=10;

    // Reset star if it passes viewport
    if (z <= 0.0)
      reset();
  }

…
```

```
 void reset() {
    // Reset star to a position far away
    x = random(-5000, 5000);
    y = random(-5000, 5000);
    z = 2000.0;
  }

  void draw() {
    // Project star only viewport
    float offsetX = 100.0*(x/z);
    float offsetY = 100.0*(y/z);
    float scaleZ = 0.0001*(2000.0-z);

    // Draw this star
    pushMatrix();
    translate(offsetX, offsetY);
    scale(scaleZ);
    ellipse(0,0,20,20);
    popMatrix();
  }
}
```

```
// starfield

// Array of stars
Star[] stars = new Star[400];

void setup() {
  size(600, 600);
  smooth();
  stroke(255);
  strokeWeight(5);
  rectMode(CENTER);

  // Init all stars
  for (int i=0; i<stars.length; i++)
    stars[i] = new Star();
}

void draw() {
  background(0);

  // Draw all stars wrt center of screen
  translate(0.5*width, 0.5*height);

  // Update and draw all stars
  for (int i=0; i<stars.length; i++) {
    stars[i].update();
    stars[i].draw();
  }
}
```

Add the necessary transformations to `draw()` to render the rectangle at the center of the sketch, twice its size, and rotated by 45 degrees (PI/4 radians).

```
void setup() {
    size(400, 400);
    rectMode(CENTER);
}

void draw() {

    // Add transformations here
    translate( width/2, height/2);
    scale(2);
    rotate(PI/4.0);

}
```