Review

- Objects
- Classes
- Objects and Arrays

Models of Motion with Objects

- Linear Translation
- Bouncing
- Rotation
- Seeking a Target
- Gravity and Friction
- Accelerating toward a Target
- Perspective (starfield)

Components of the Main Sketch

1. A global array to hold all objects
2. A loop to update and draw all objects, if they exist
3. A global counter to track next available array index
4. A function to create and store new objects

Outline for a Graphic Object Class

1. All fields necessary to maintain object state
   - Variables for (x, y) position at a minimum
2. A constructor to initialize new objects
3. A step() method to update state of the object
   - Including object location, and any other fields desired
4. A draw() method to render object on sketch

```
// A simple Box class
class Box {
  float x, y;

  Box(float tx, float ty) {
    x = tx;    // x position
    y = ty;    // y position
  }

  void step() {}

  void draw() {
    fill(200);
    rect(x, y, 20, 20);
  }
}
```

BoxMaker.pde

Linear Translation

```
class Mover {
  float x, y, vx, vy;

  Mover(float tx, float ty) {
    x = tx;     // x position
    y = ty;     // y position
    vx = 1.0;   // x velocity
    vy = 0.0;   // y velocity
  }

  void step() {
    x = x + vx;    // Motion
  }
…
}
```

How can we make the box bounce off the walls?

BoxMover.pde

Rotation

```
class Rotator {
  float x, y;
  float angle;

  Rotator(float tx, float ty) {
    x = tx;     // x position
    y = ty;     // y position
    angle = 0.0;
  }

  void step() {
    angle = angle + radians(5);
  }

  void draw() {
    fill(200);
    pushMatrix();
    translate(x, y);
    rotate(angle);
    rect(0, 0, 20, 20);
    popMatrix();
  }
}
```

How can we make the box orbit instead of rotate?

BoxRotator.pde

Seeking a Target

How can we visualize the target?

```
class Seeker {
  float x, y;
  float targetx, targety;

  Seeker(float tx, float ty) {
    x = tx;     // x position
    y = ty;     // y position
    targetx = random(width);  // Initial target
    targety = random(height); // location
  }

  void step() {

    x = x + 0.01*(targetx - x);  // New position is
    y = y + 0.01*(targety - y);  // toward target

    if (dist(x, y, targetx, targety) < 40.0) {
      targetx = random(width);   // Change target
      targety = random(height);  // when too close
    }
  }
```

BoxSeeker.pde

Gravity

```
class Dropper {
  float x, y, vx, vy, ay;

  Dropper(float tx, float ty) {
    x = tx;     // x position
    y = ty;     // y position
    vx = 0.0;   // x velocity
    vy = 0.0;   // y velocity
    ay = 0.02; // gravity
  }

  void step() {
    if ( y <= height ) {    // Stop at the floor
      x = x + vx;           // Equations of motion
      y = y + vy;
      vy = vy + ay;
    }

  }
  …
}
```

BoxDropper.pde

Gravity and Friction (and Bounce)

```
class Bouncer {

  …

  void step() {
      x = x + vx;          // Equations of motion
      y = y + vy;
      vy = vy + ay;

    if (y >= height) {     // Bounce off the floor
      y = height;          // Prevent box catch at floor
      vy = -0.7*vy;        // Bounce with friction
    }
  }

  …

}
```

BoxBouncer.pde

Drift Toward a Target

```
class Drifter {
  float x, y, vx, vy;
  float ax, ay;
  float targetx, targety;

  Drifter(float tx, float ty) {
    x = tx;     // x position
    y = ty;     // y position
    vx = 0.0;   // x velocity
    vy = 0.0;   // y velocity
    ax = 0.0;   // x acceleration
    ay = 0.0;   // y acceleration

    // Initialize a random target
    targetx = random(width);
    targety = random(height);
  }

…
```

Box is accelerated toward the target

BoxDrifter.pde

Drift Toward a Target (Cont'd)

```
void step() {

  ax = 0.0002*(targetx-x); // Accelerate toward target
  ay = 0.0002*(targety-y);

  vy = vy + ay;            // Update velocity
  vx = vx + ax;
                           // Constrain velocity
  vx = constrain(vx, -0.5, 0.5);
  vy = constrain(vy, -0.5, 0.5);

  x = x + vx;              // Update position
  y = y + vy;

  // Calculate new target when too close
  if ( dist(x, y, targetx, targety) < 40.0 ) {
    targetx = random(width);
    targety = random(height);
  }

}
```

BoxDrifter.pde

AllBoxes

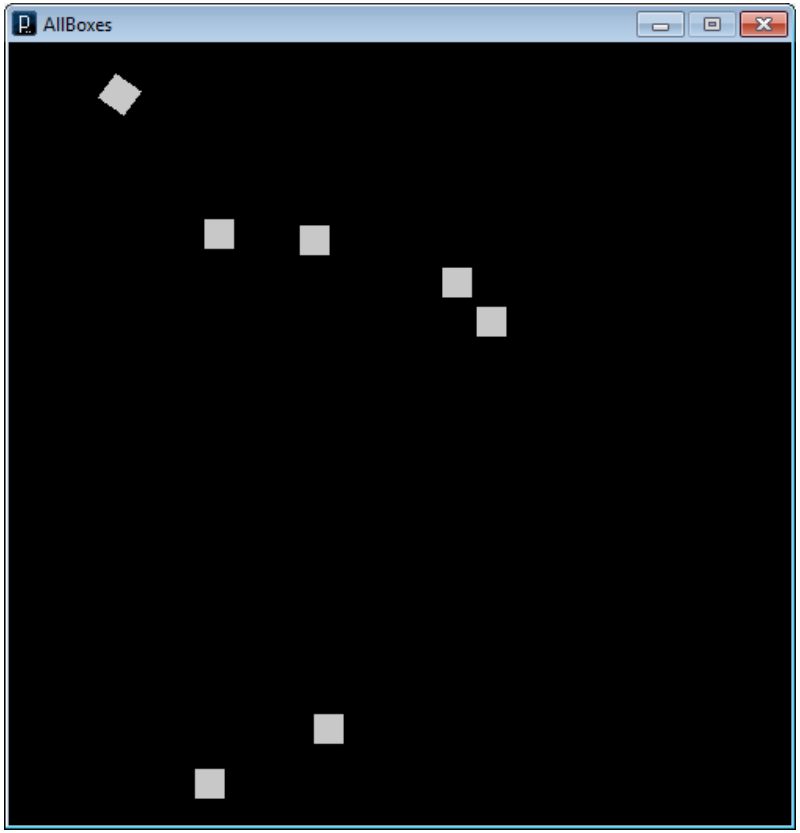
```
// AllBoxes
Box aBox;
Mover aMover;
Rotator aRotator;
Seeker aSeeker;
Dropper aDropper;
Bouncer aBouncer;
Drifter aDrifter;


void setup() {
  size(500, 500);
  rectMode(CENTER);

  aBox = new Box( random(width), random(0.5*height) );
  aMover = new Mover( random(width), random(0.5*height) );
  aRotator = new Rotator( random(width), random(0.5*height) );
  aSeeker = new Seeker( random(width), random(0.5*height) );
  aDropper = new Dropper( random(width), random(0.5*height) );
  aBouncer = new Bouncer( random(width), random(0.5*height) );
  aDrifter = new Drifter( random(width), random(0.5*height) );
}
```

```
void draw() {
  background(0);

  // Update all box objects
  aBox.step();
  aMover.step();
  aRotator.step();
  aSeeker.step();
  aDropper.step();
  aBouncer.step();
  aDrifter.step();

  // Draw all boxes
  aBox.draw();
  aMover.draw();
  aRotator.draw();
  aSeeker.draw();
  aDropper.draw();
  aBouncer.draw();
  aDrifter.draw();
}
```

AllBoxes.pde

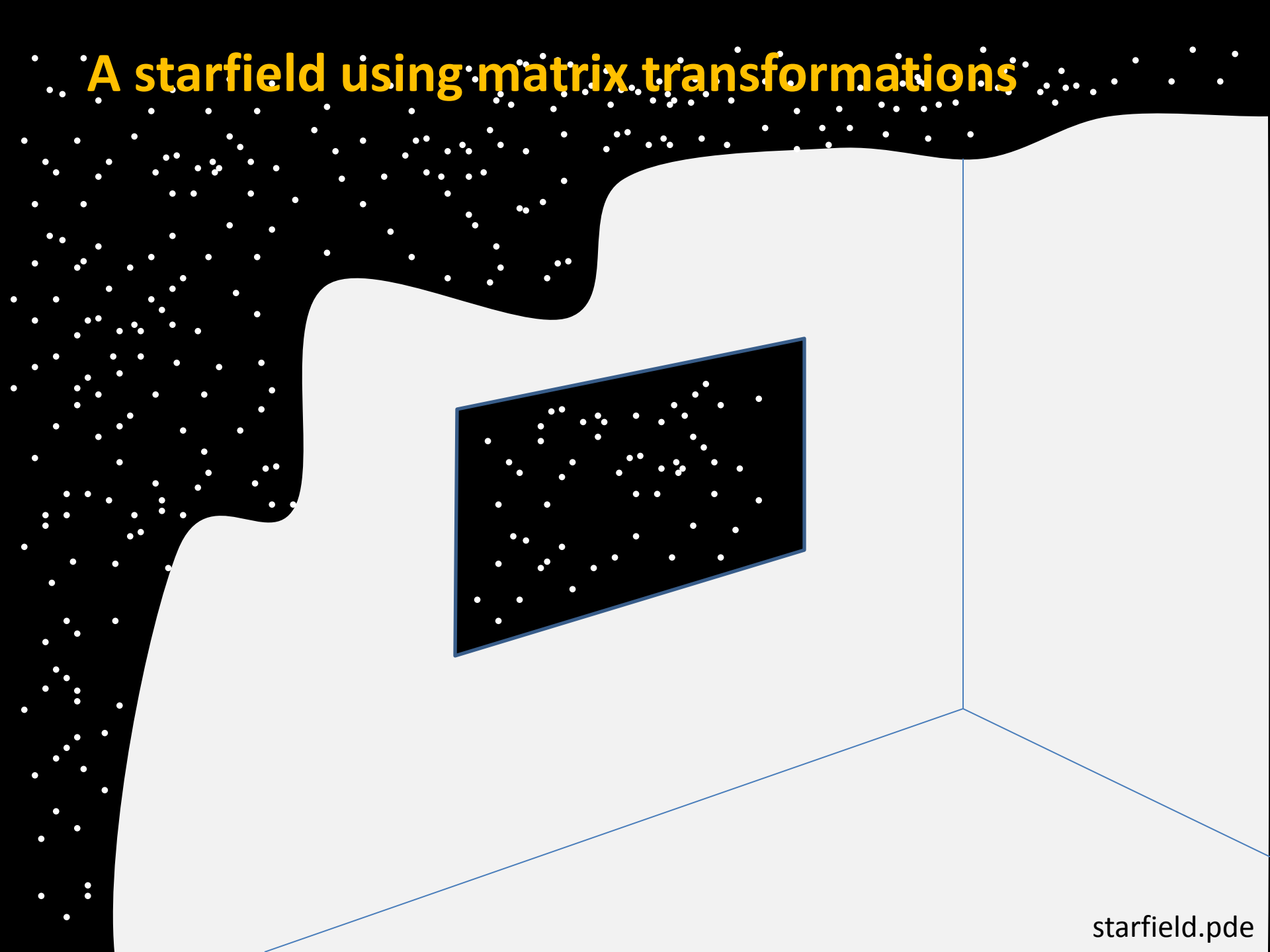
AllBoxes

- Note
  - We changed the x-y position and rotation.
  - Other field values can be changed instead, such as fill color, scale, width, height, …
  - The main program never changed.
  - Each object encapsulates its own behavior, so all can coexist.

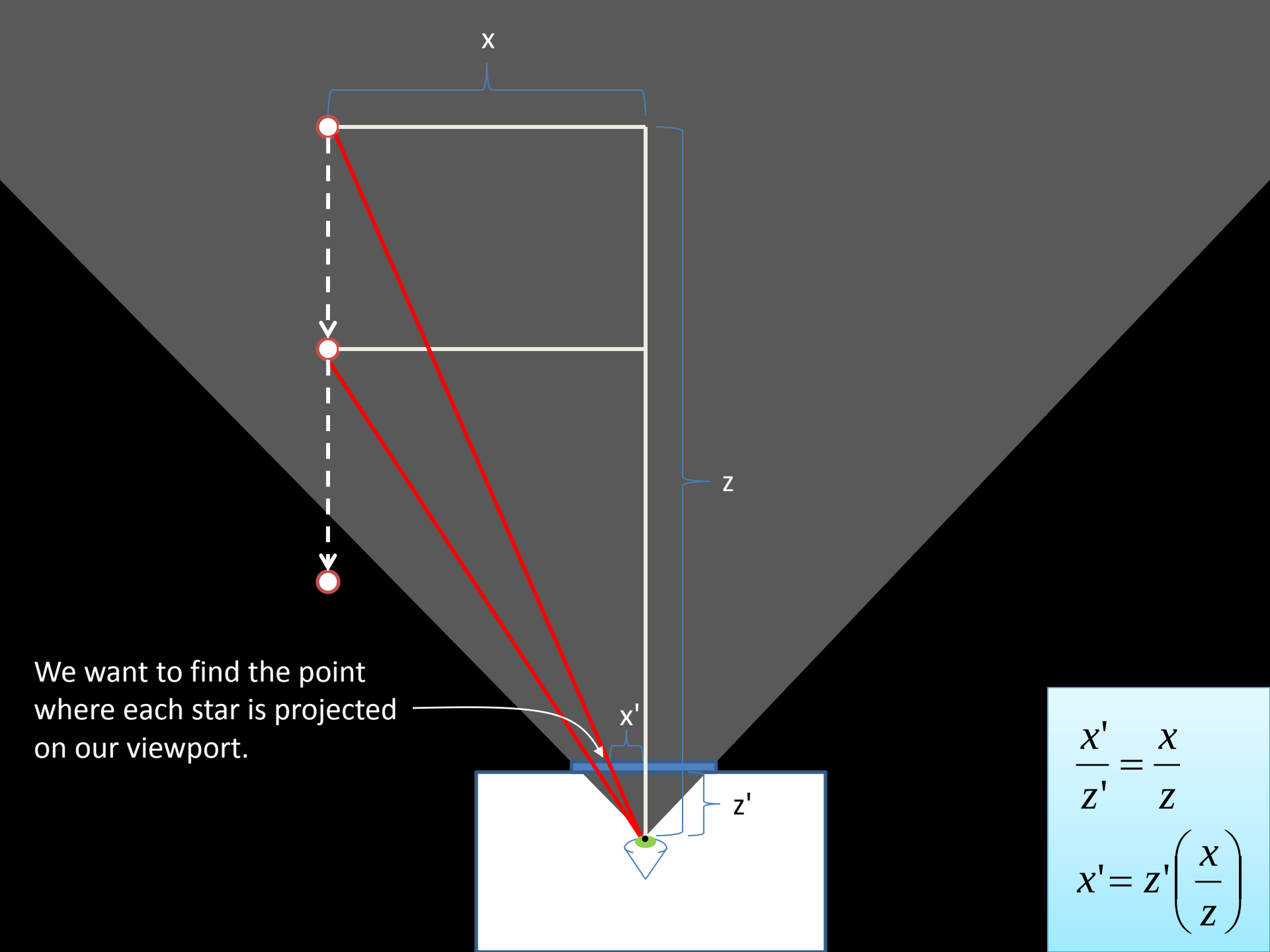


AllBoxes.pde

A starfield using matrix transformations



starfield.pde



We want to find the point where each star is projected on our viewport.

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

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

```
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();
  }
}
```