CS324e - Elements of Graphics and Visualization

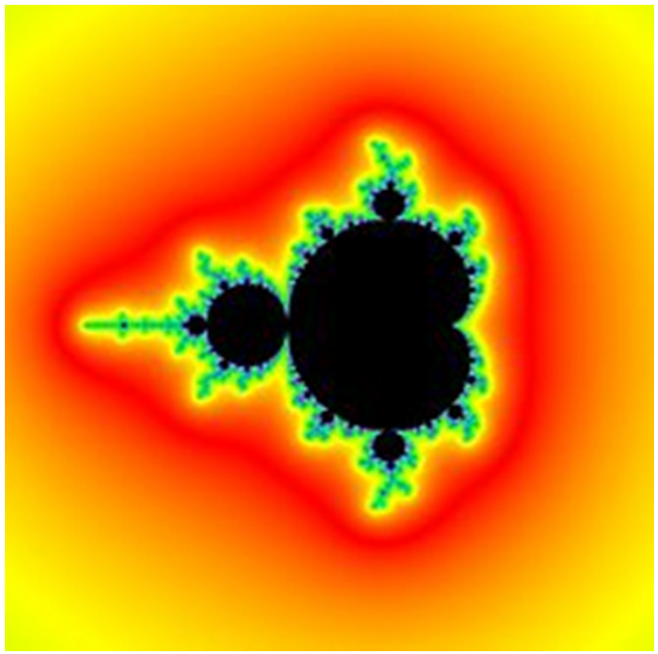
Fractals and 3D Landscapes

Fractals

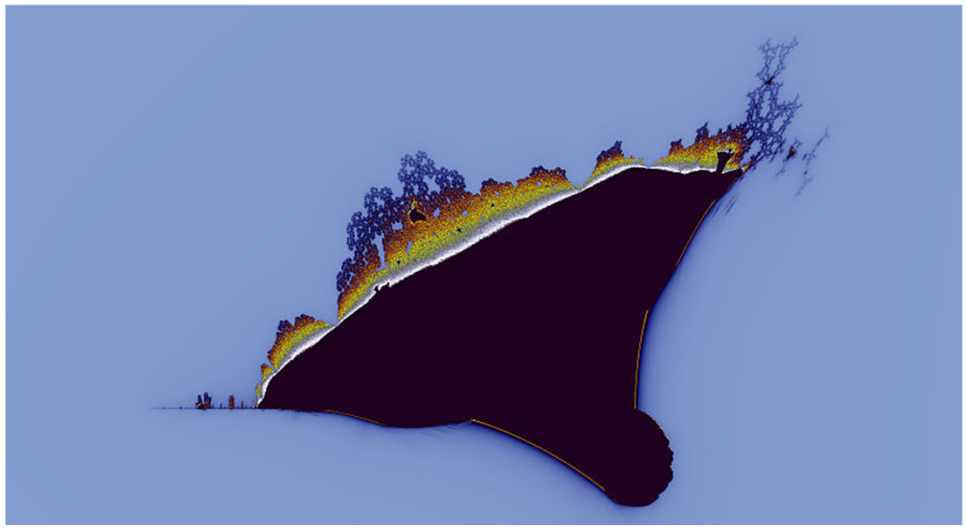
- A geometric figure in which smaller parts share characteristics of the entire figure
  - a detailed pattern that repeats itself
  - contain self similar patterns
  - appearance of details matches the overall figure
  - often described mathematically

Fractals

- Mandelbrot Set



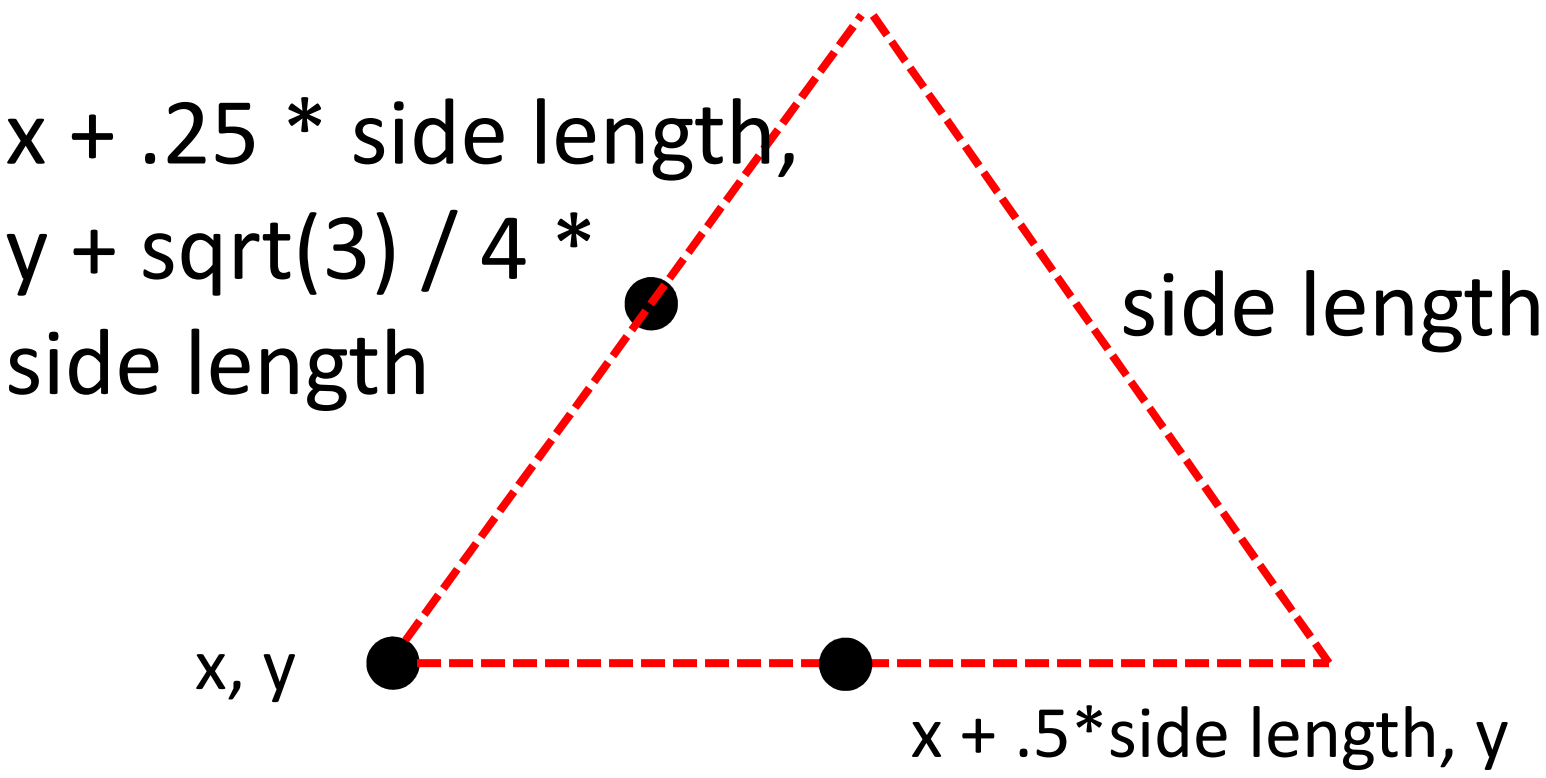
- Burning Ship Fractal



Sierpinski Triangle Fractal

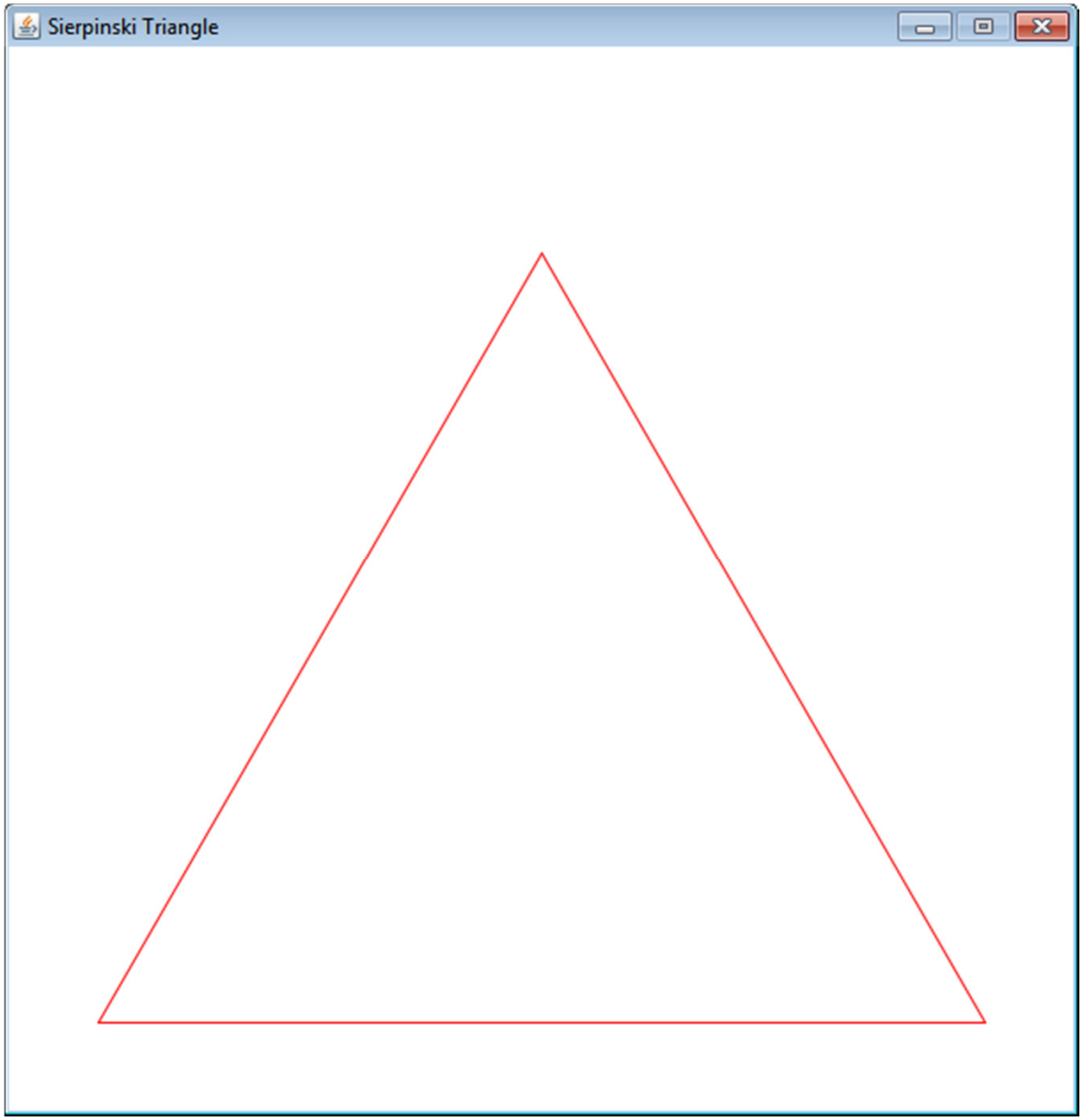
- Described by Polish mathematician Wacław Sierpiński in 1915
- Algorithm
  - Pick a side length and the lower left vertex for an equilateral triangle
  - If the side length is less than some minimum draw the triangle
  - else draw three smaller Sierpinski Triangles

Sierpinski Triangle



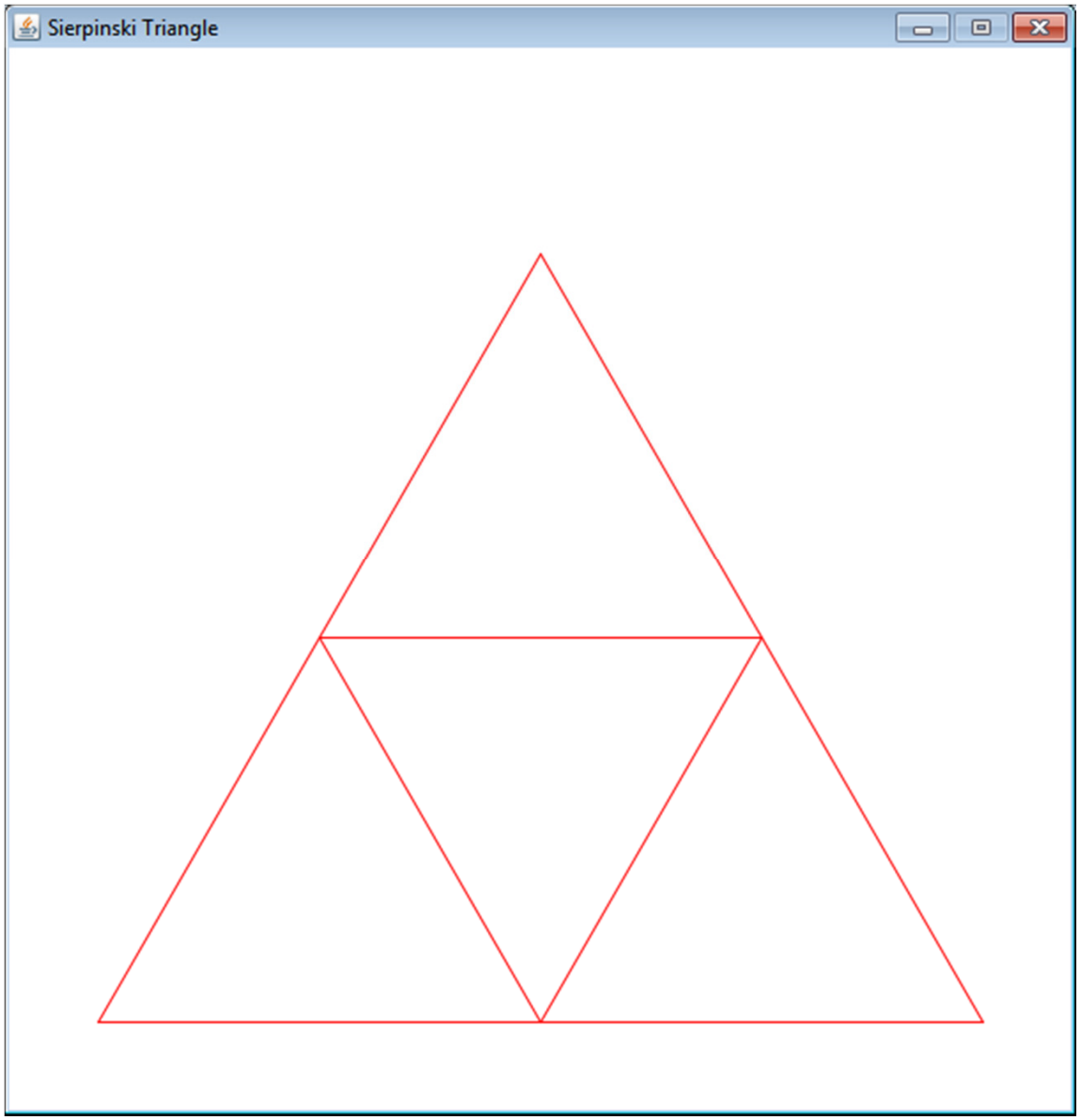
Sierpinski Triangle

- min length
  = start length



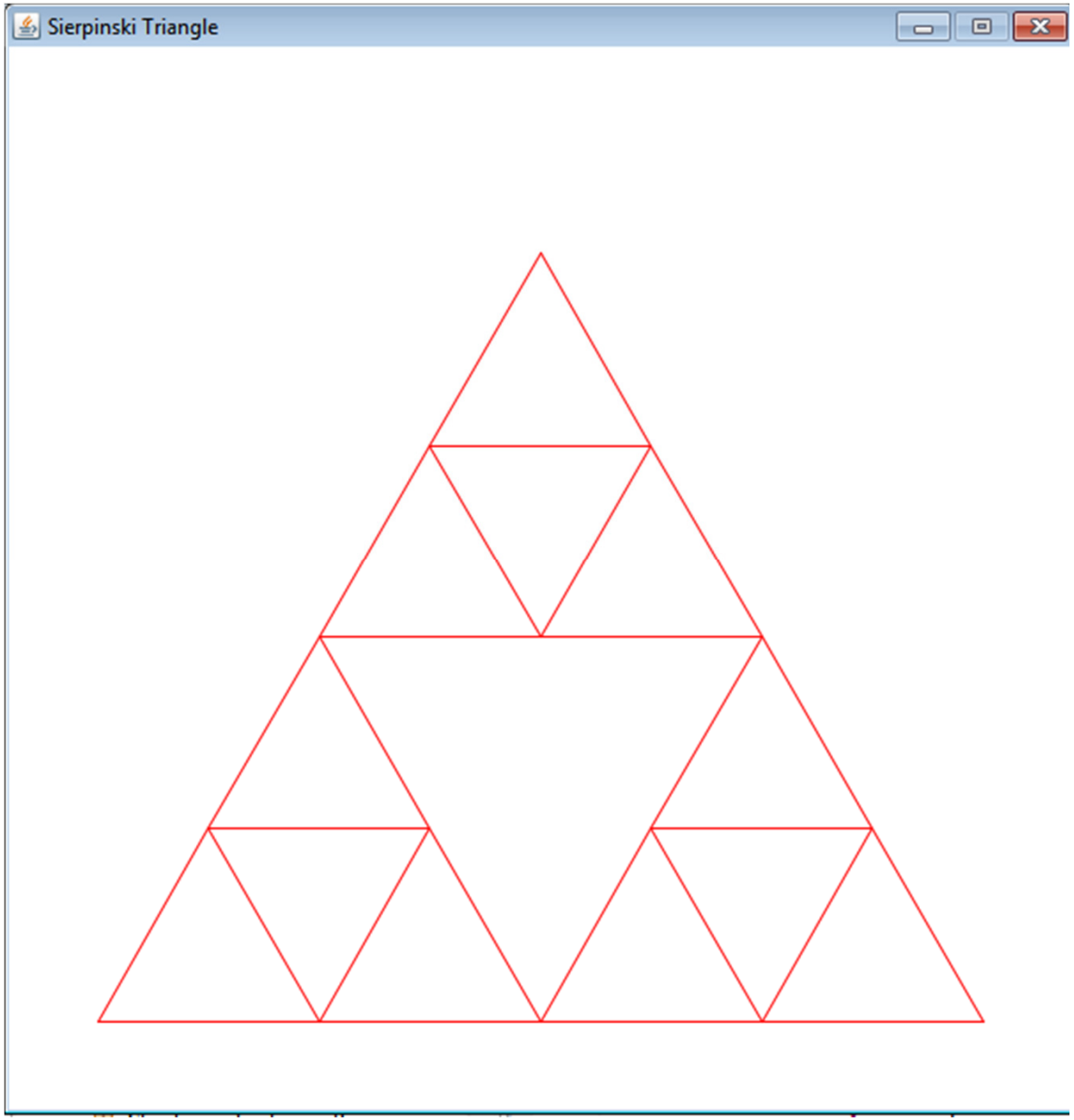
Sierpinski Triangle

- min length
  = start length / 2



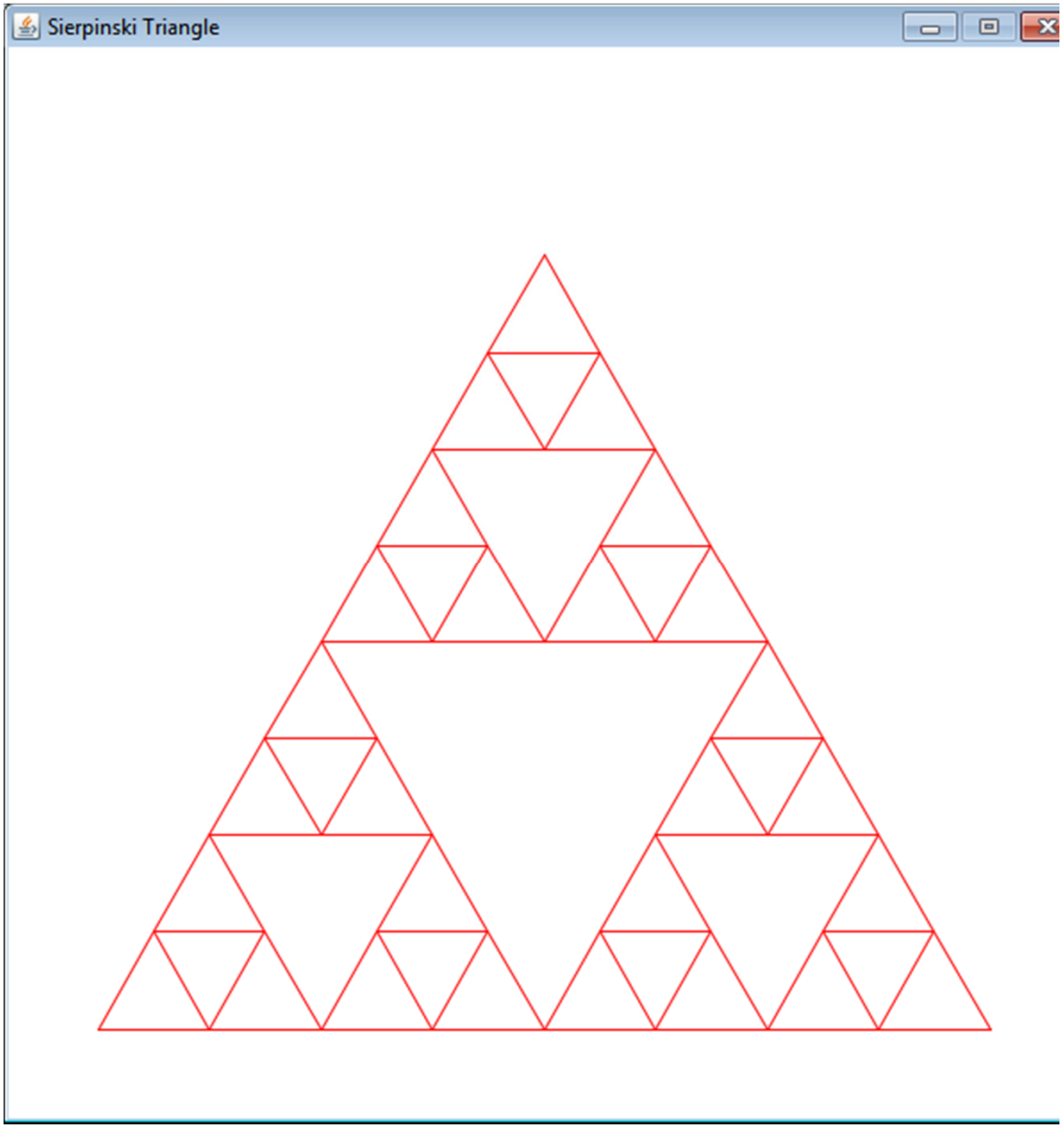
Sierpinski Triangle

- min length
  = start length / 4



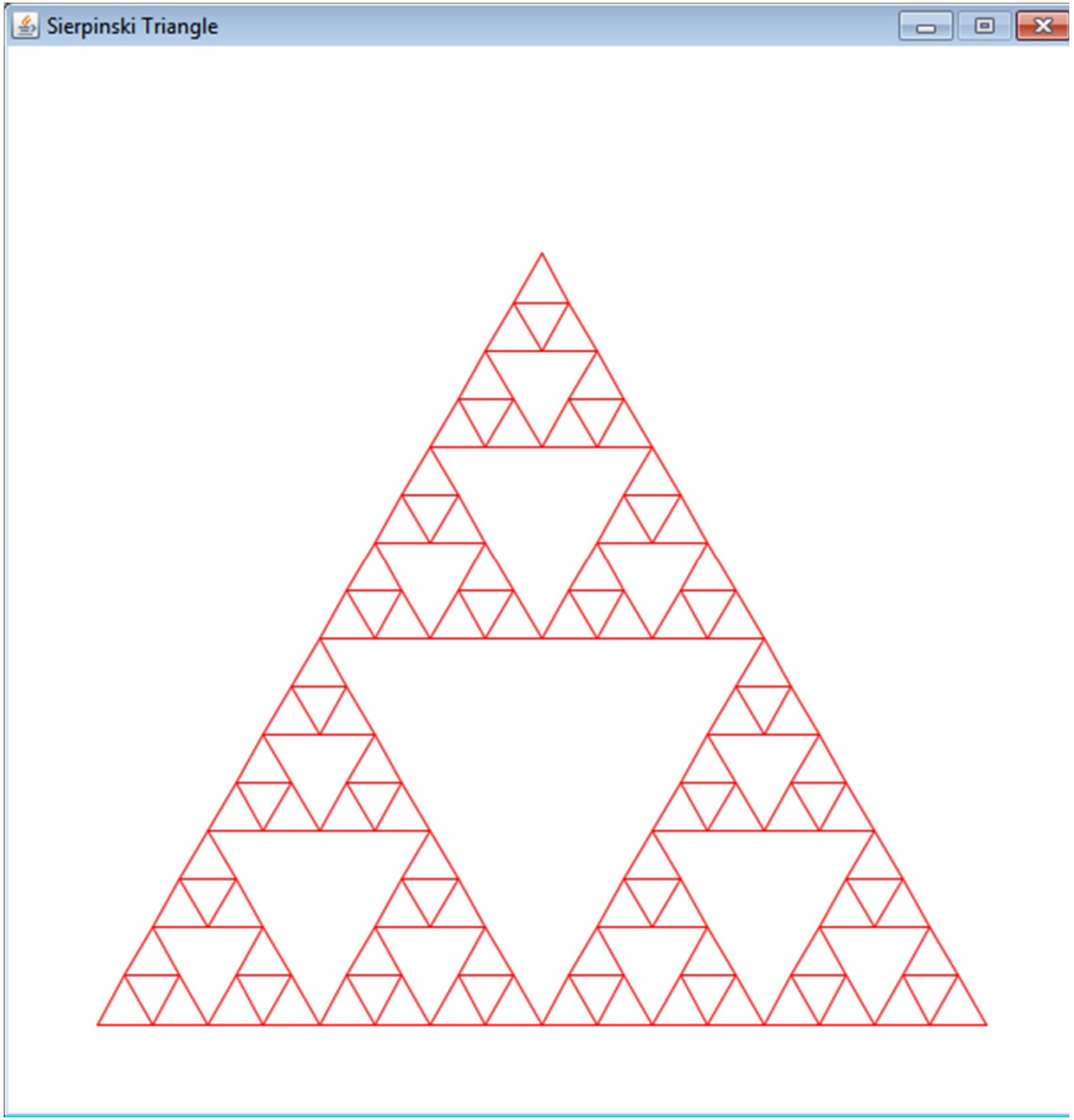
Sierpinski Triangle

- min length
  = start length / 8



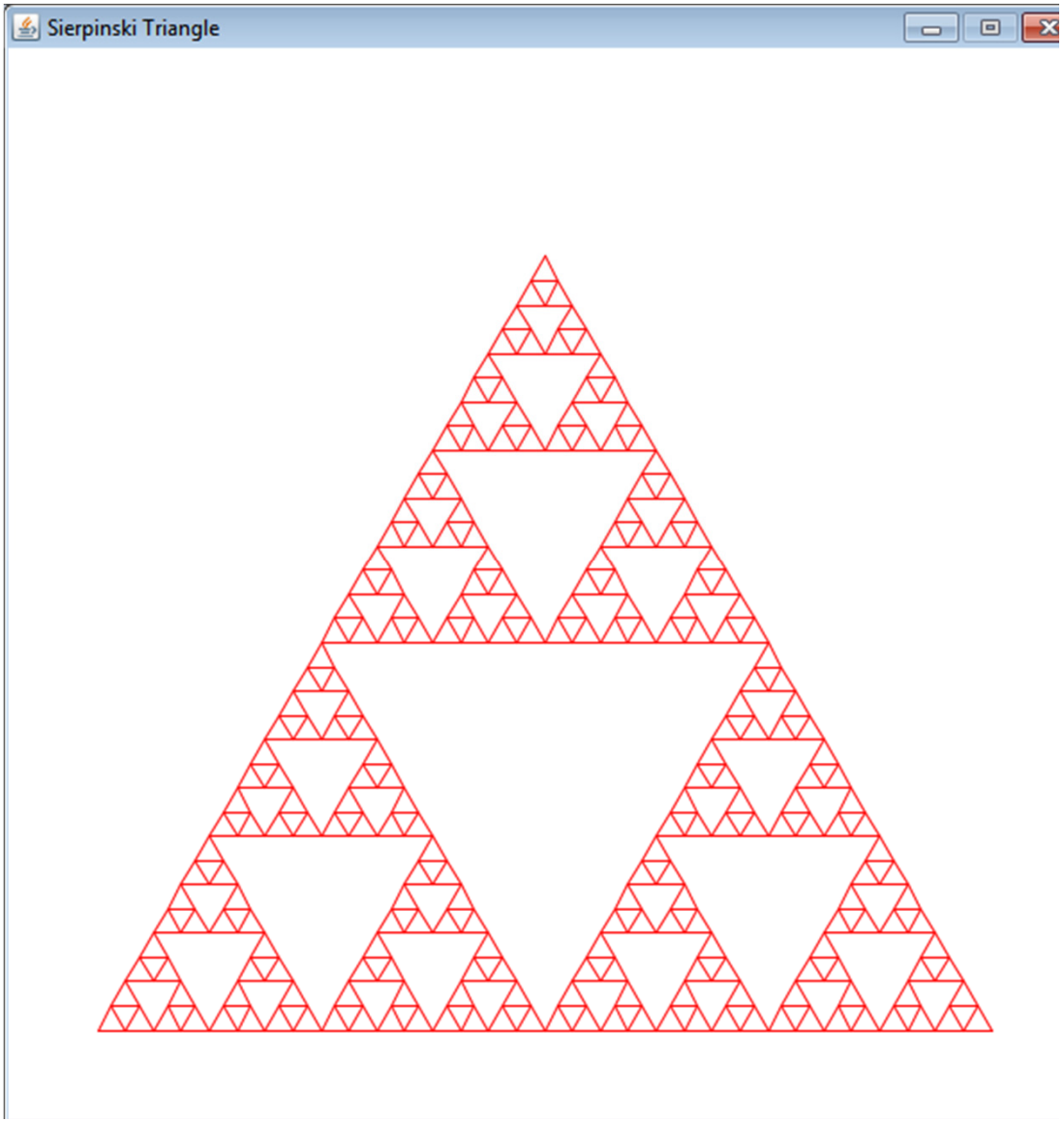
Sierpinski Triangle

- min length
  = start length / 16



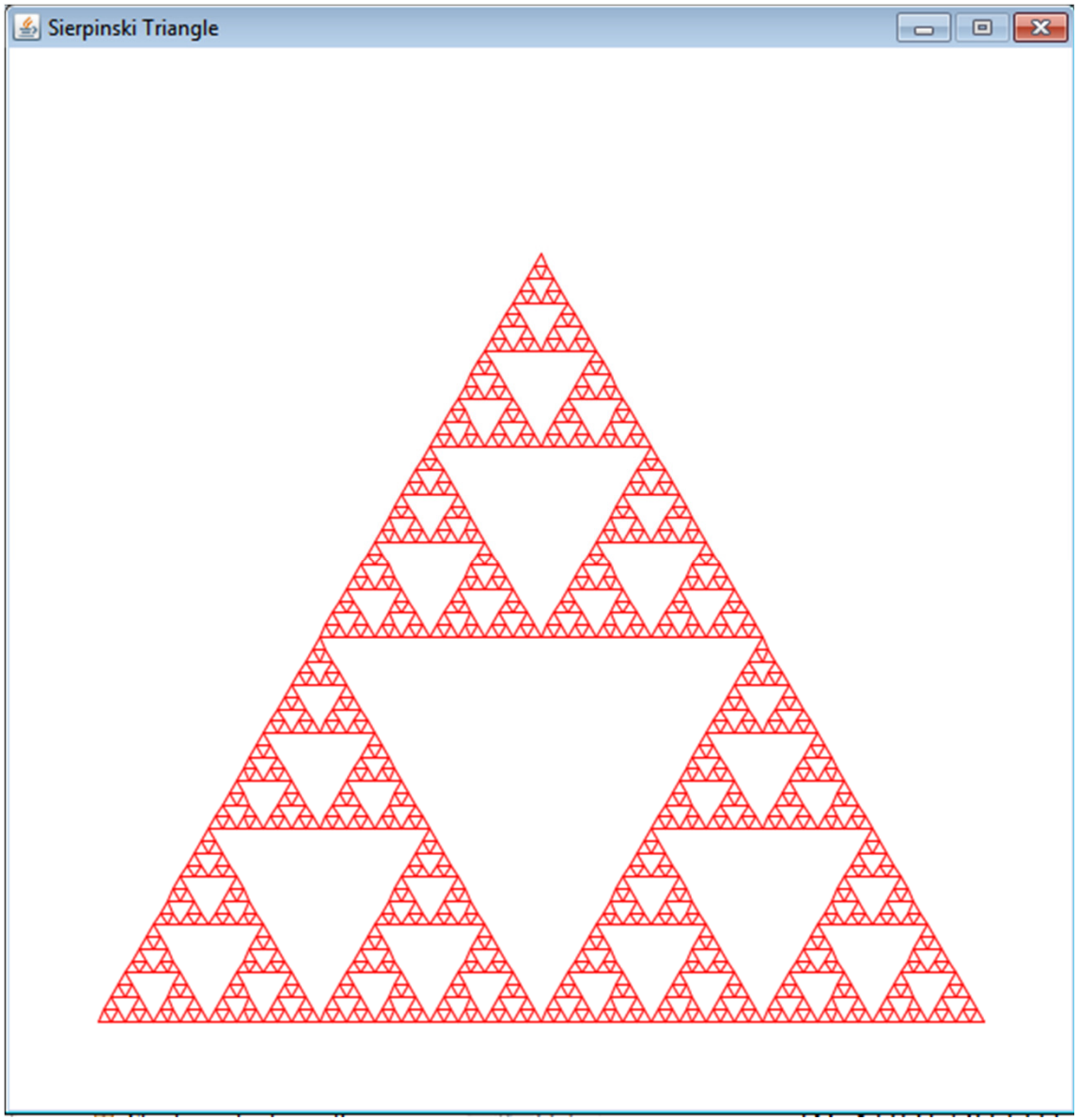
Sierpinski Triangle

- min length
  = start length / 32



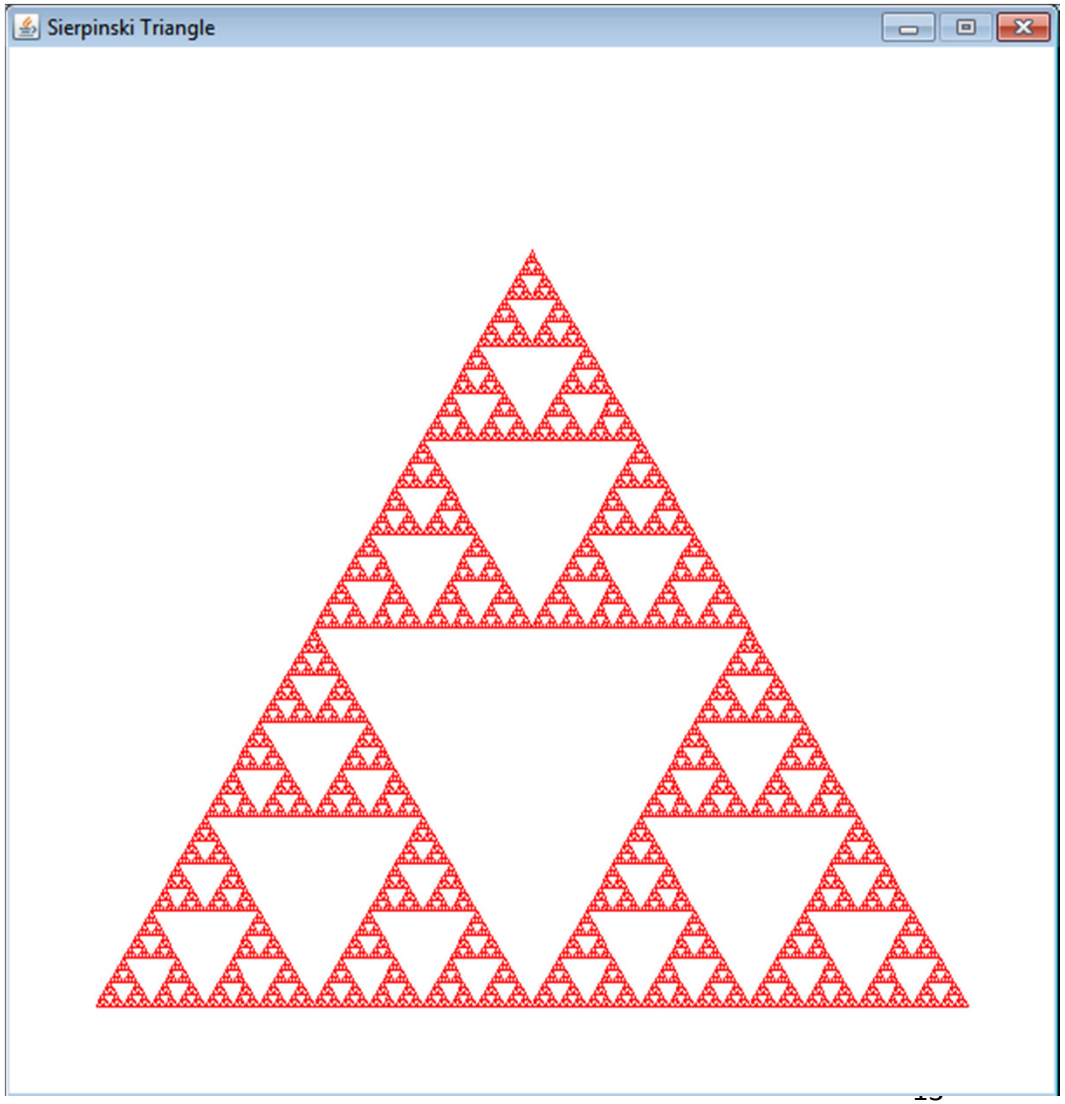
Sierpinski Triangle

- min length
  = start length / 64



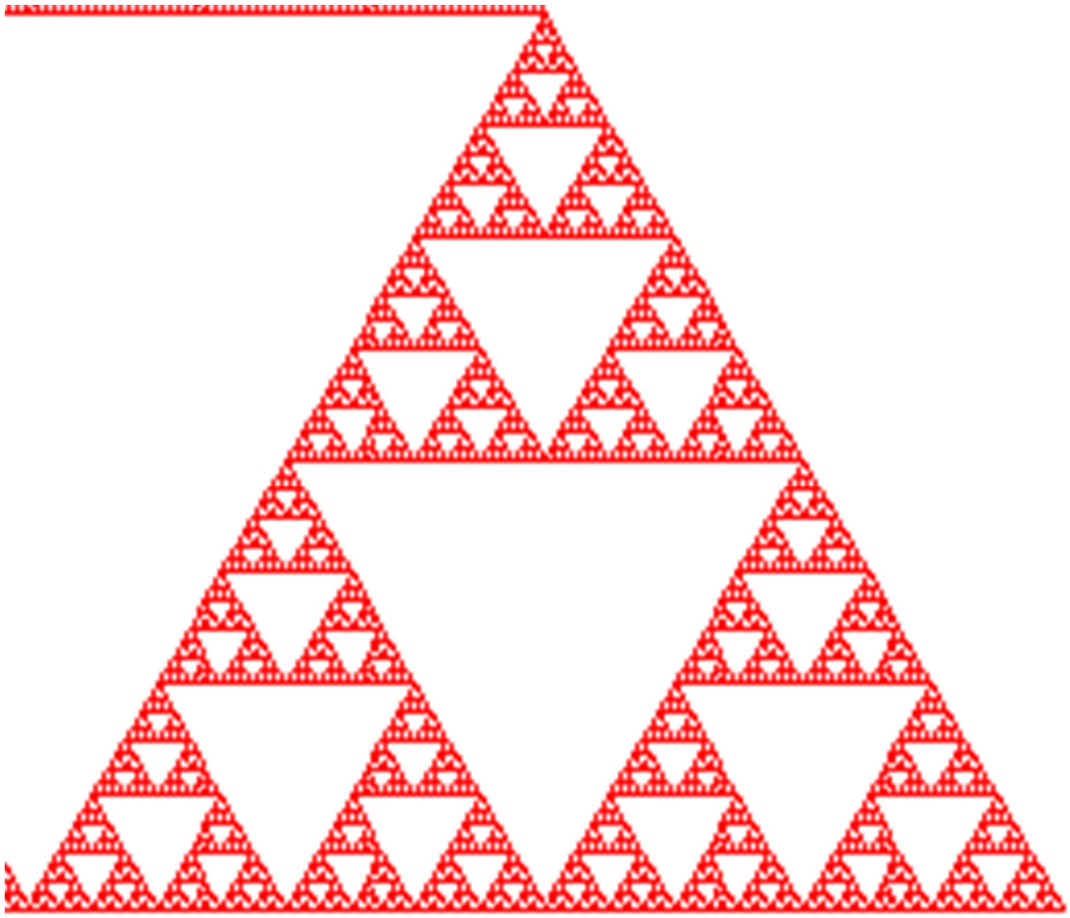
Sierpinski Triangle

- min length
  = start length / 128



Close up - Self Similar

- Close up of bottom, right triangle from minLength = startLength / 128



Implementation of Sierpinski

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
       RenderingHints.VALUE_ANTIALIAS_ON);
    // g2.setStroke(new BasicStroke(2));
    g2.setColor(Color.RED);
    double x = (SIZE - START_LENGTH) / 2;
    double y = SIZE - x;
    drawTriangles(g2, x, y, START_LENGTH);
}
```

Implementation of Sierpinski

```
private static final int SIZE = 600;
private static final double START_LENGTH = 500;
private static final double Y3_FACTOR = Math.sqrt(3) / 4;
private static final double Y3_FINAL = Math.sqrt(3) / 2;

private double minLength = START_LENGTH / 128;
```

```
private void drawTriangles(Graphics2D g2, double x1, double y1,
        double currentLength) {
    if(currentLength <= minLength)
        drawOneTriangle(g2, x1, y1, currentLength);
    else {
        double x2 = x1 + currentLength / 2;
        double x3 = x1 + currentLength / 4;
        double y3 = y1 - Y3_FACTOR * currentLength;
        double newLength = currentLength / 2;
        drawTriangles(g2, x1, y1, newLength);
        drawTriangles(g2, x2, y1, newLength);
        drawTriangles(g2, x3, y3, newLength);
    }
```

Base Case

Recursive Case

Implementation of Sierpinski

Base Case - Draw One Triangle

```
private void drawOneTriangle(Graphics2D g2, double x1,
        double y1, double currentLength) {
    double x2 = x1 + currentLength;
    double x3 = x1 + currentLength / 2;
    double y3 = y1 - Y3_FINAL * currentLength;
    g2.drawLine((int) x1, (int) y1, (int) x2, (int) y1);
    g2.drawLine((int) x1, (int) y1, (int) x3, (int) y3);
    g2.drawLine((int) x3, (int) y3, (int) x2, (int) y1);
}
```

Fractal Land

- Example from KGPJ chapter 26
- Create a mesh of quads
  - example 256 x 256 tiles
- allow the height of points in the quad to vary from one to the next
- split quads into one of 5 categories based on height
  - water, sand, grass, dry earth, stone
  - appearance based on texture (image file)

Fractal Mesh

- Generate varied height of quads using a "Diamond - Square" algorithm
- looking down on mesh
- heights of points A, B, C, D

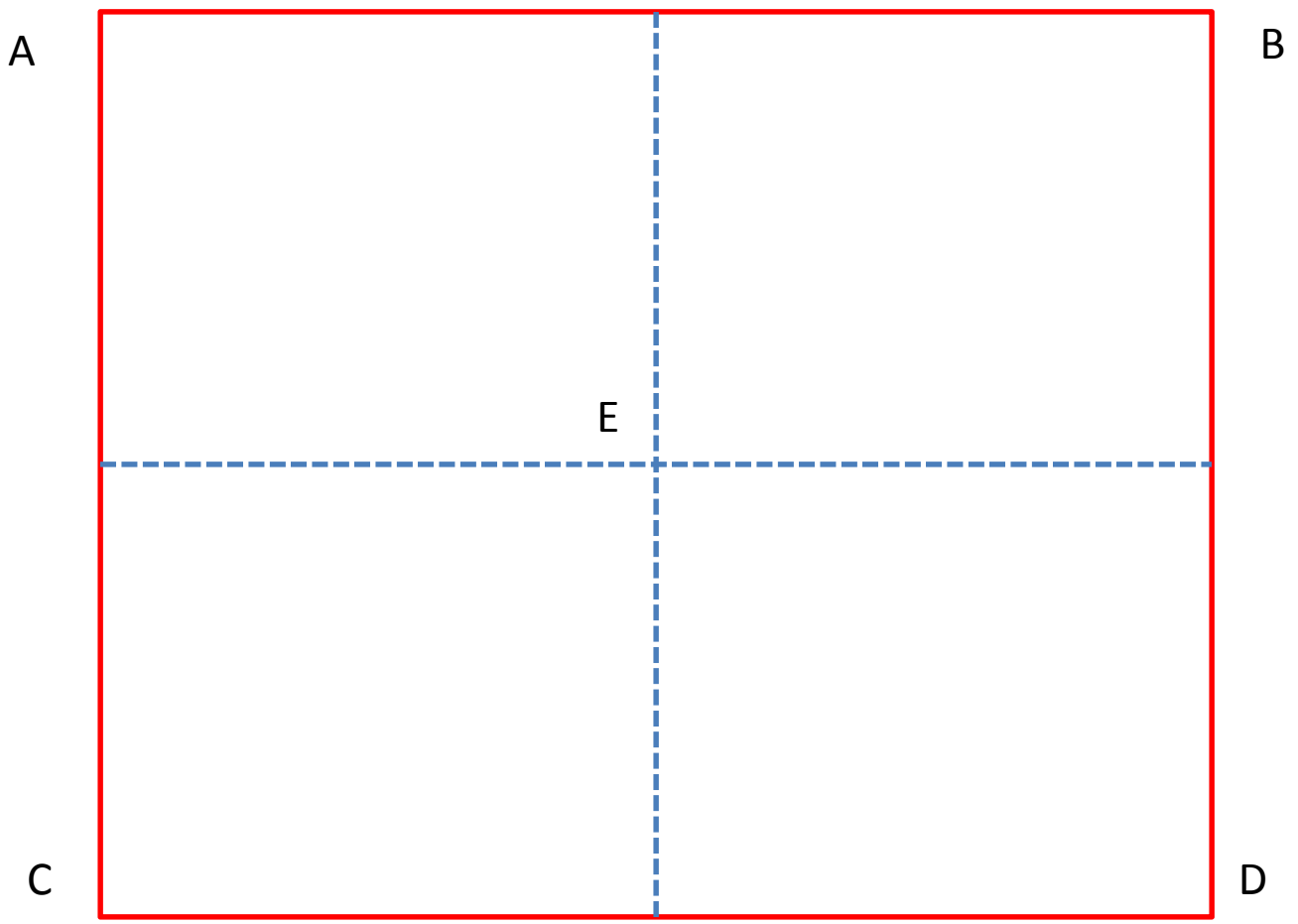
Generate Fractal Mesh - Diamond Step

dHeight = max height
- min height

Height of Point E =

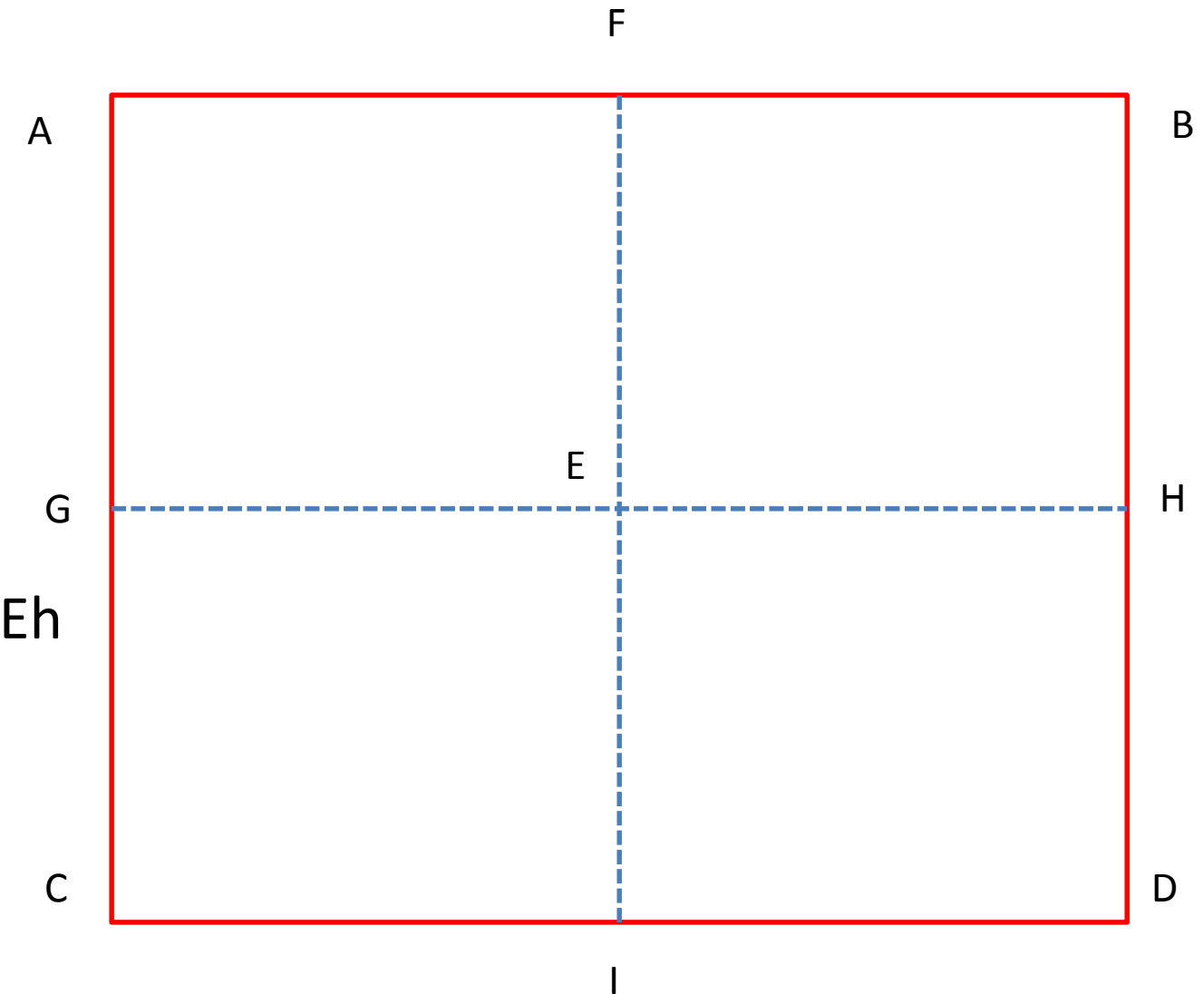
(Ah + Bh + Ch + Dh) / 4
+ random(-dHeight/2,
+dHeight/2)

average of 4 corner points
plus some random value
in range of max and min
allowed height

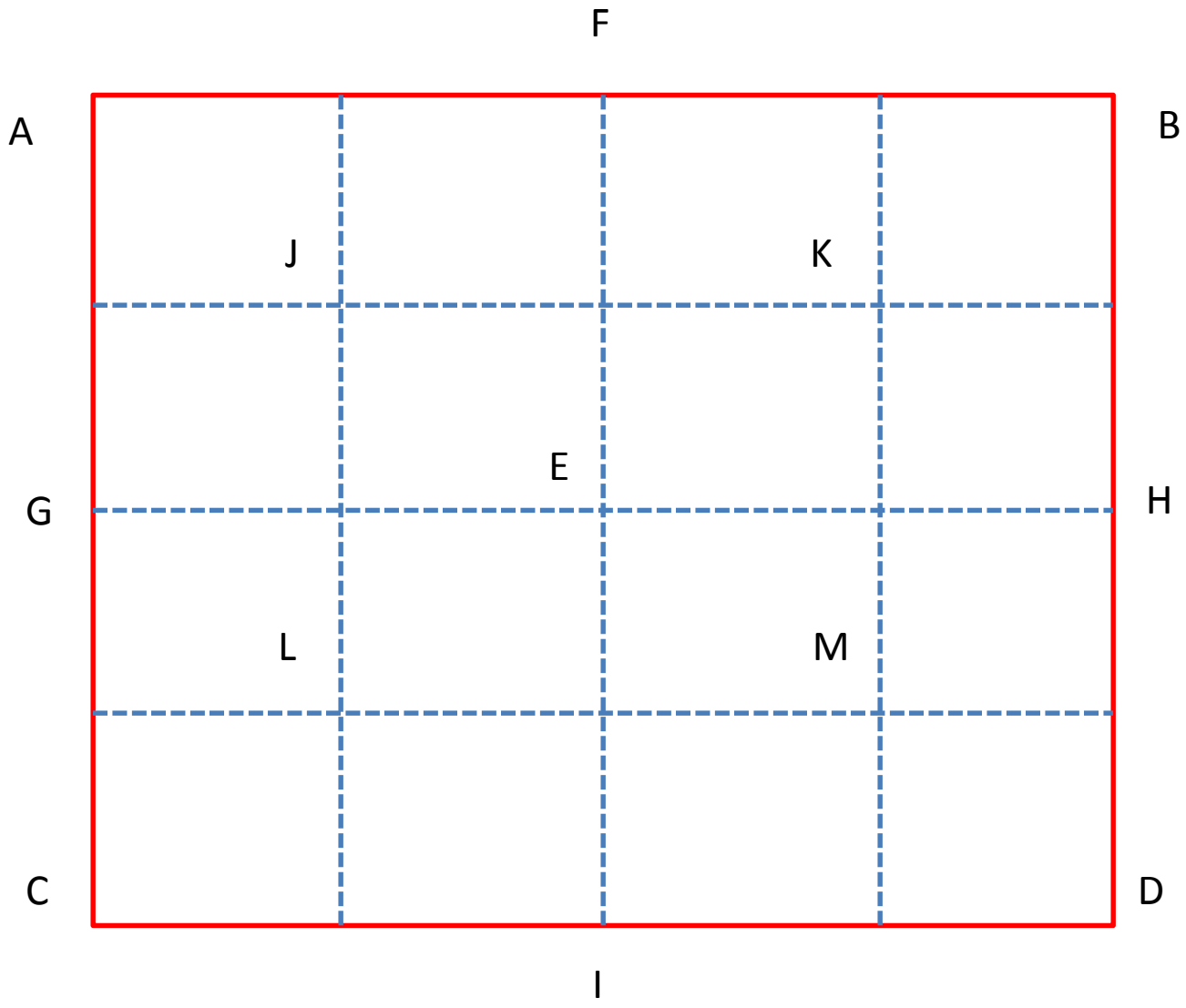


Square Step

- With height of E set generate height of 4 points around E
- Gh = (Ah + Eh + Eh + Ch) / 4 + random(-dHeight/2, +dHeight/2)



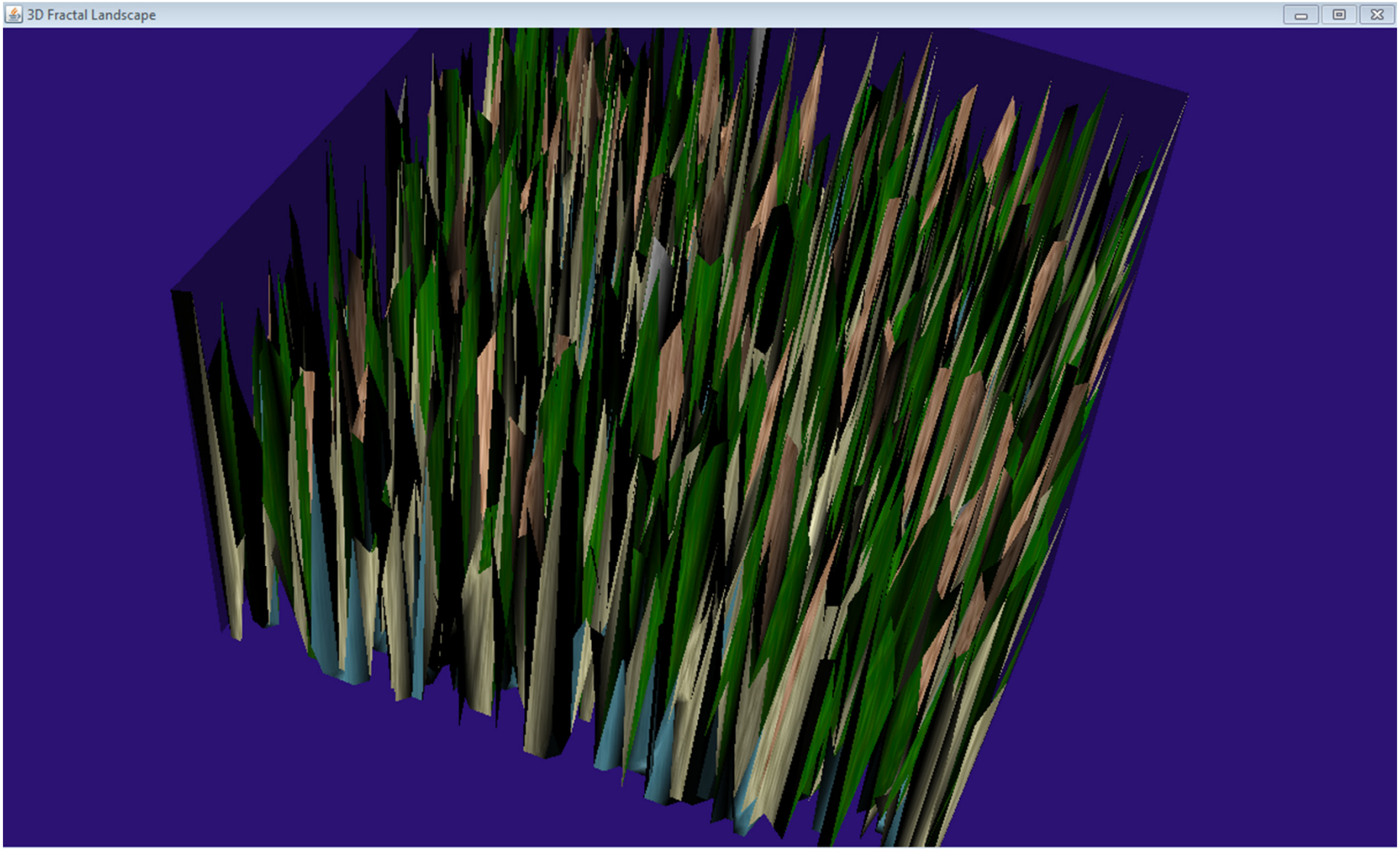
Repeat Diamond Step



Completing Mesh

- Continuing alternating diamond and square step until the size of the quad is 1.
- Each point of quad at a fixed x and z coordinate, but the y has been generated randomly
- Problem: if the height is allowed to vary between the min and max height for every point how different can points on a quad be?

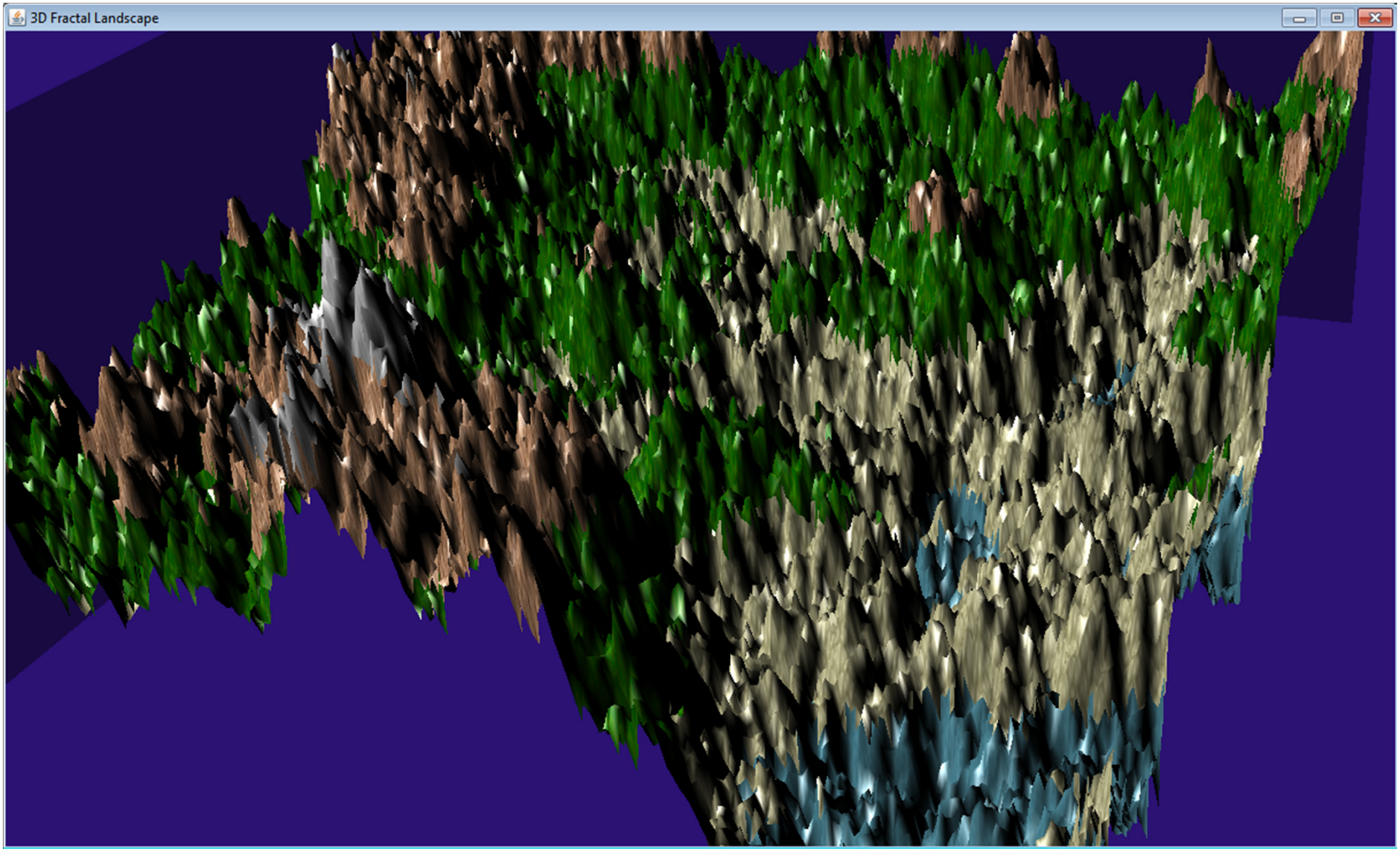
A Spear Trap



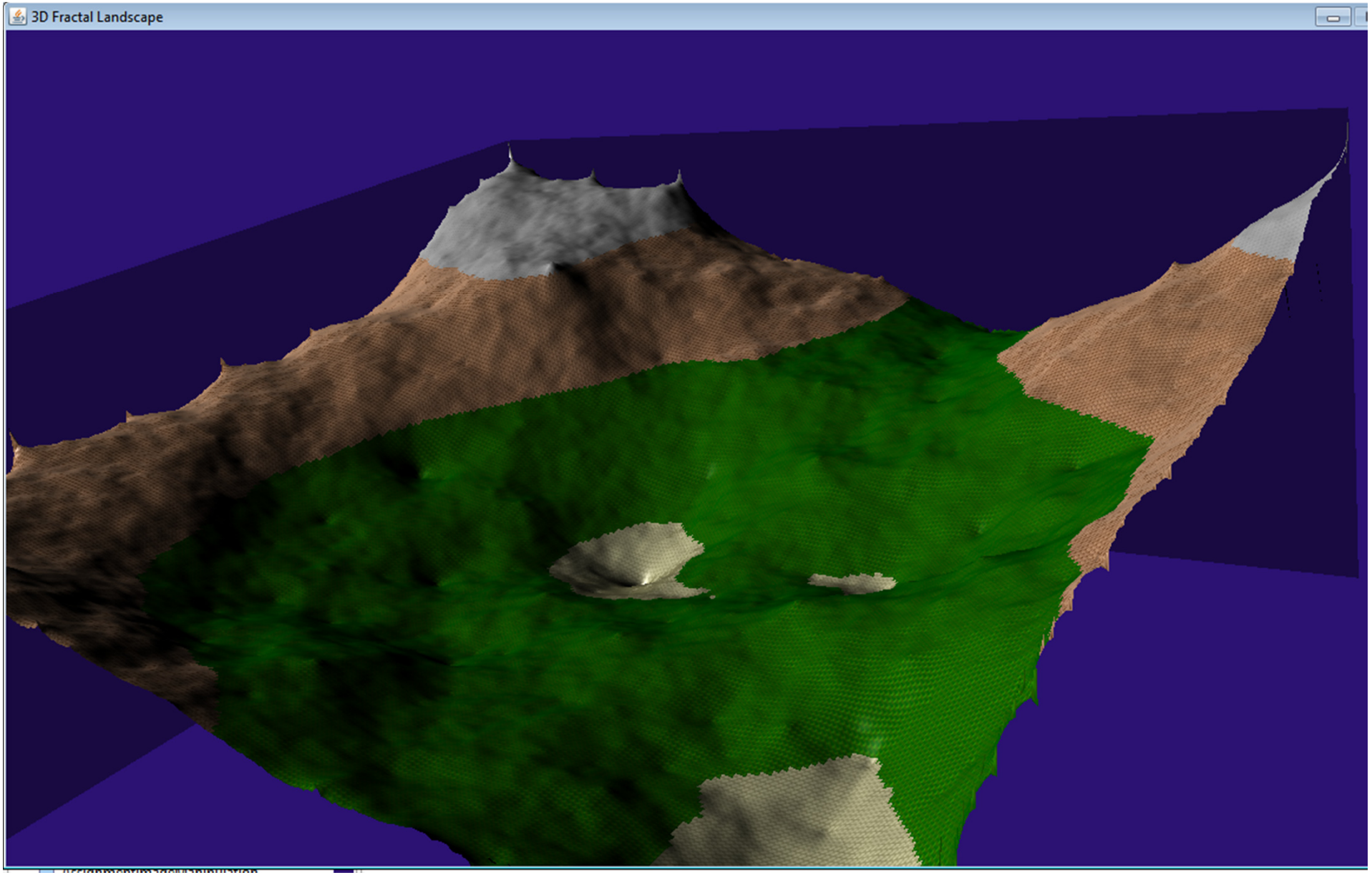
What is the Problem?

- By allowing the points that form a single quad to vary anywhere in the range from min to max height we get vast differences
- Solution: after a pair of diamond - square steps reduce the range of the random value
- referred to as the *flatness* factor
- range = range / flatness

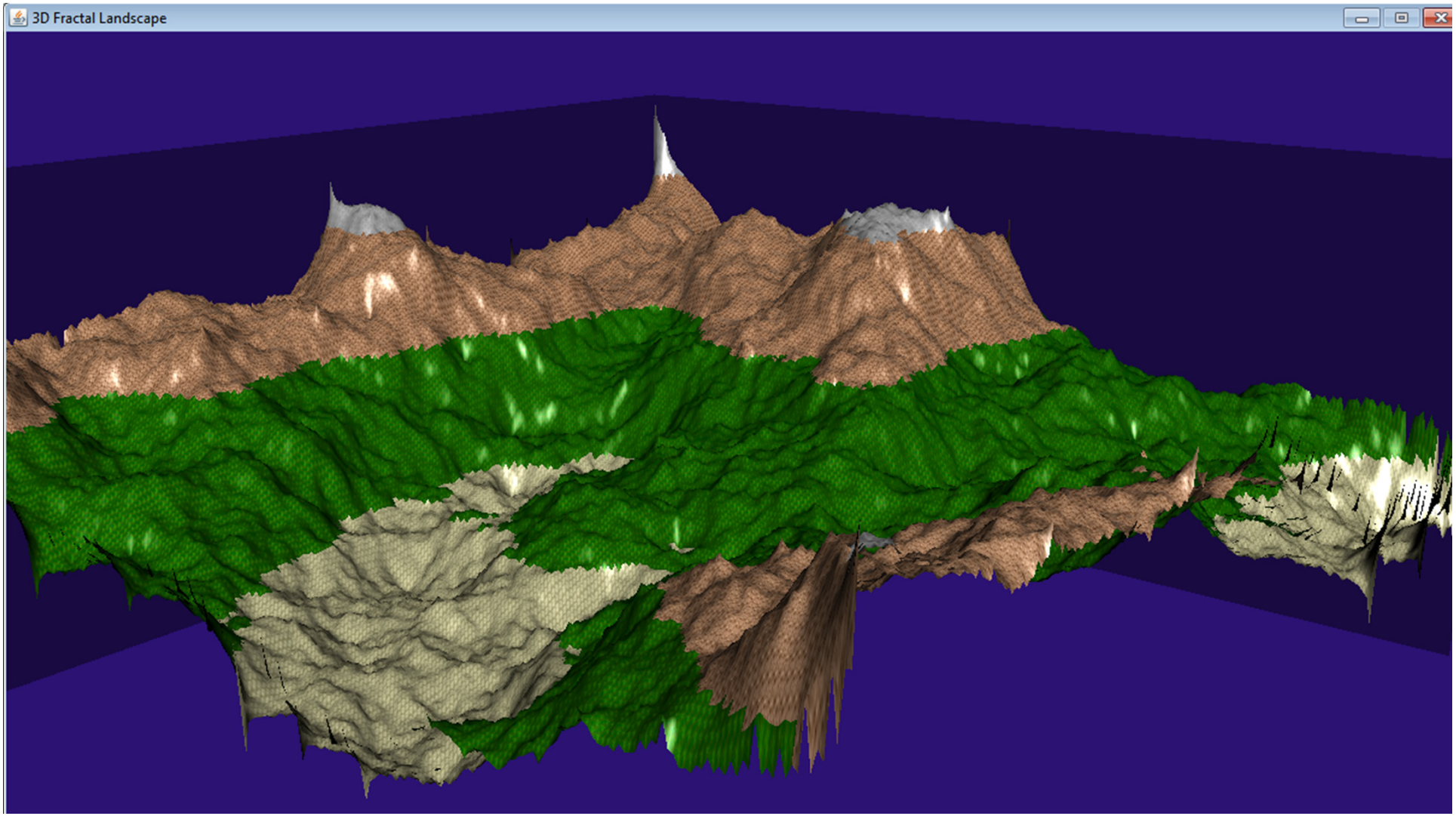
Flatness of 1.5



Flatness of 2.5



Flatness of 2.0



Textures

- Shapes in Java3D may be wrapped in a texture
- In FractalLand the mesh is simply a QuadArray
- Each texture is an image file
- Quad Array created and texture coordinates generated for each quad
  - how does image map to quad

Simple Textures

- Texture can also be applied to the primitive shapes: box, cone, sphere, cylinder
- From the interpolator example
- When creating box must add primFlag to generate texture coordinates

```
private void addPillar() {
    Appearance ap = getApp();
    Box b = new Box(5, 10f, 7f, Box.GENERATE_NORMALS
            | Box.GENERATE_TEXTURE_COORDS , ap);

    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(5, 5, 10));
    TransformGroup positionTG = new TransformGroup(t3d);
```

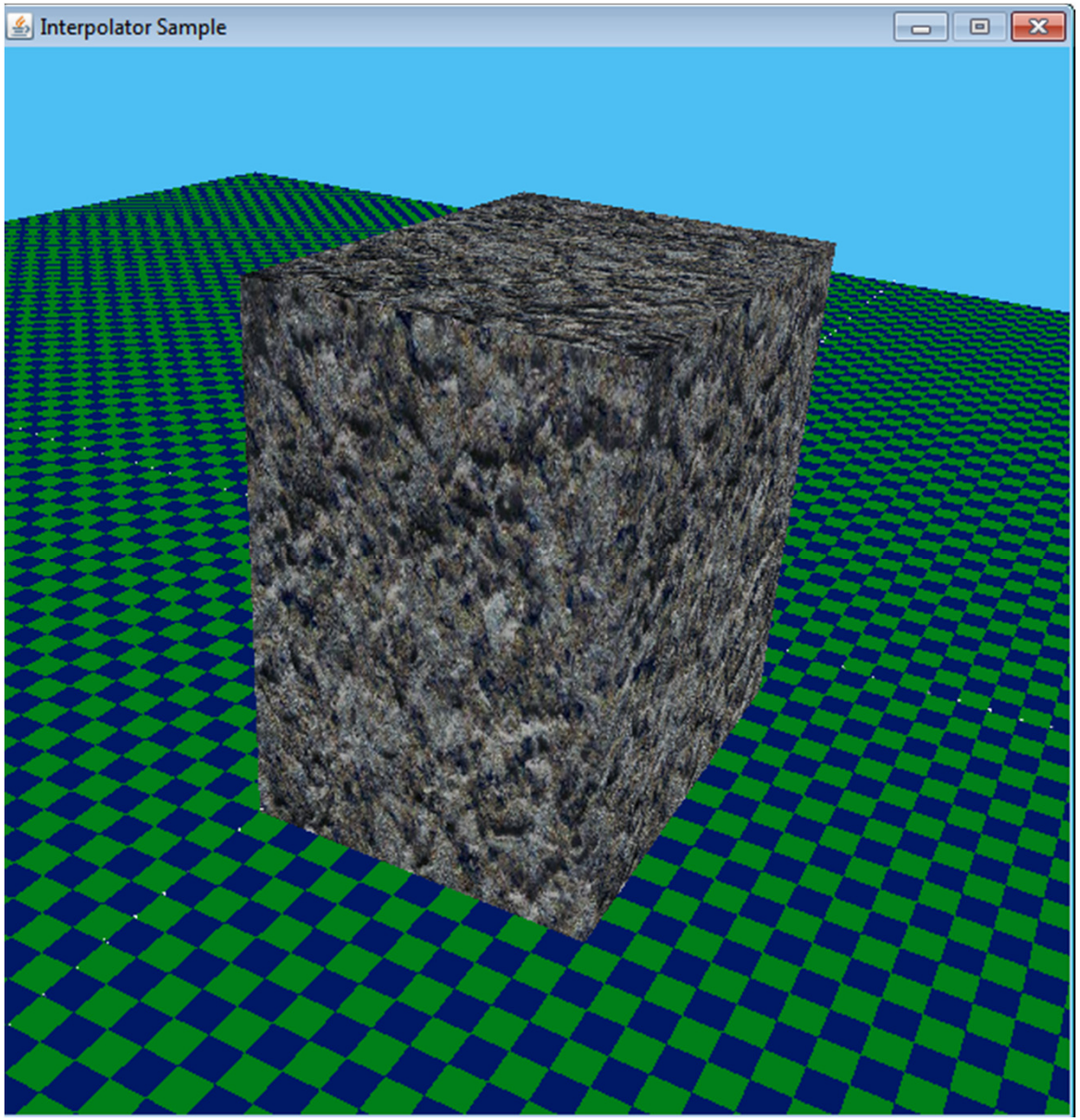
Creating Appearance

- Appearance for shape based on texture not material

```
private Appearance getApp() {
    Texture texture =
        new TextureLoader("images\\stone.jpg", null).getTexture()

    Appearance result = new Appearance();
    result.setTexture(texture);
    return result;
}
```

Result

- texture wrapped and repeated as necessary
- can lead to odd seams, creases, and stretching

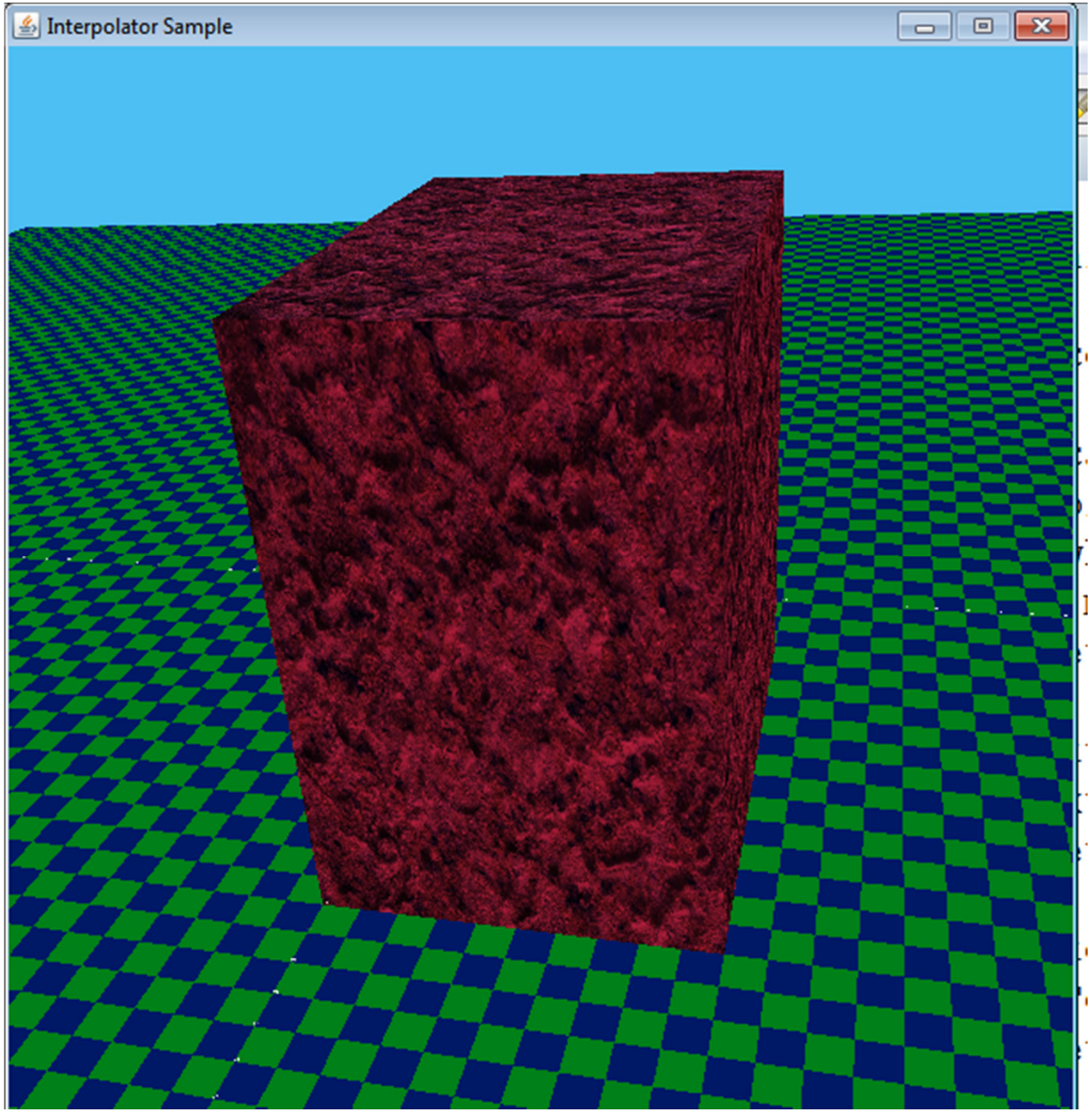


Combining Texture and Material

- can combine material and texture to create *modulated* material

```
private Appearance getApp() {

    Appearance result = new Appearance();

    Color3f card = new Color3f(.77f, .12f, .24f);
    Color3f black = new Color3f(0, 0, 0);
    Color3f whiteish = new Color3f(.8f, .8f, .8f);
    Material mat = new Material(card, black, card, whiteish, 64);
    result.setMaterial(mat);

    TextureAttributes ta = new TextureAttributes();
    ta.setTextureMode(TextureAttributes.MODULATE);
    result.setTextureAttributes(ta);

    Texture texture =
        new TextureLoader("images\\stone.jpg", null).getTexture();
    result.setTexture(texture);

    return result;
}
```

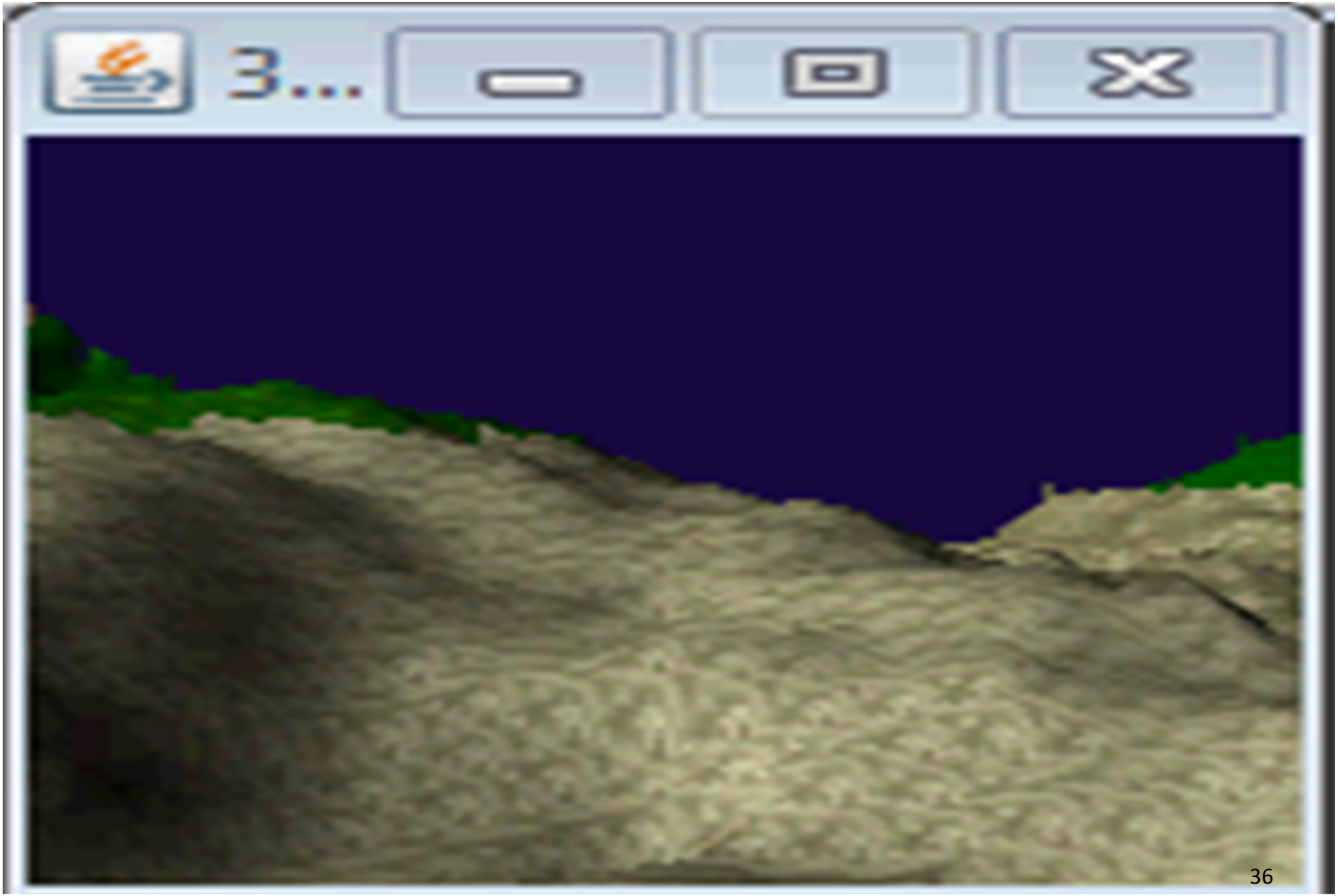
Result



Controls

- Version of FractalLand shown had orbit controls to allow movement of camera anywhere in scene
- program includes method to add key controls and keeps camera close to the ground
  - as if moving across the landscape

Result



Adding Fog

- Java3D includes ability to add fog
- LinearFog
- ExponentialFog
- density of fog as function of distance from the camera
- fog has color and parameters to determine density of fog

LinearFog

```
public class LinearFog
extends Fog
```

The LinearFog leaf node defines fog distance parameters for linear fog. LinearFog extends the Fog node by adding a pair of distance values, in Z, at which the fog should start obscuring the scene and should maximally obscure the scene.

The front and back fog distances are defined in the local coordinate system of the node, but the actual fog equation will ideally take place in eye coordinates.

The linear fog blending factor, `f`, is computed as follows:

```
f = (backDistance - z) / (backDistance - frontDistance)
```

where:

`z` is the distance from the viewpoint.
`frontDistance` is the distance at which fog starts obscuring objects.
`backDistance` is the distance at which fog totally obscurs objects.

```
private void addFog() {
    // linear fog
    // skyColour = new Color3f(0.17f, 0.07f, 0.45f);
    LinearFog fogLinear = new LinearFog(skyColor, 5.0f, 40.0f);
    fogLinear.setInfluencingBounds(bounds);  // same as background
    sceneBG.addChild(fogLinear);
}  // end of addFog()
```

Result