Introduction

- Show how to produce graphics using OpenGL
- Introduce our framework for OpenGL programming
OpenGL

- **Open Graphics Library**
- API for GPU-accelerated 2D / 3D rendering
- Cross-platform (Windows, OS X, Linux, iOS, Android, ...)
- Developed by SGI in 1992
  - 2009: OpenGL 3.2
  - 2010: OpenGL 4.0
  - 2011: OpenGL 4.2
- Main alternative: Direct3D (Windows / XBox)
Why Use OpenGL?

- Tap massive power of GPU hardware to render images
- Use GPU without worrying about exact details of hardware
What OpenGL Does For Us

• Controls GPU
• Lets user specify resources...:
  – Geometry (vertices and primitives)
  – Textures
  – Shaders (programs that run in the graphics pipeline)
  – Etc.
• ...and use them:
  – Rasterize and draw geometry
How We'll Use OpenGL

- **OpenGL version**
  - We use 2.1, plus extensions
  - Some features in 2.1 are no longer commonly used
  - Code uses subset of features used in 3.x and 4.x

- **JOGL**
  - Java bindings for OpenGL API

- **CS 4620 Framework**
  - Simplifies creating and using OpenGL resources
Creating an OpenGL Program
GLCanvas

- Java UI component
- Can display images created by OpenGL
- OpenGL “context”
  - Stores OpenGL state (geometry, buffers, etc.)
  - Provides the framebuffer that we draw to

- Usage: create and put into a Java window
Events

- GL canvas can trigger certain events
  - Something happens (e.g. user resizes window)
  - Canvas broadcasts a message
  - Objects interested in the event can sign up as listeners
  - When event happens, OpenGL calls method in listener

- These events let us interact with OpenGL
GLEventListener

- Interface for classes that can listen for OpenGL events
- Implementations can perform OpenGL commands

Usage:
- Create class implementing GLEventListener
- Implement four methods
- Call GLCanvas.addGLListener
GLEventListener Methods

• init
  - Called once when OpenGL context is created

• display
  - Called when GLCanvas wants to update the image it shows

• resize
  - Called when canvas is resized

• dispose
  - Called when OpenGL context is destroyed
GLEventListener Methods

- **init**
  - Use to create GL resources (geometry, textures, etc.)

- **display**
  - Use to draw to the canvas

- **resize**
  - Use to update information based on new window size

- **dispose**
  - Use to free up GL resources
Demo: Hello World!
OpenGL Commands and Resources
@Override

public void display(GLAutoDrawable drawable) {
    final GL2 gl = drawable.getGL().getGL2();

    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT);

    Program.use(gl, program);

    program.setColor(gl, 1.0f, 1.0f, 1.0f);
    boxArray.draw(gl);

    Program.unuse(gl);
}

Example: Hello World's display()
OpenGL Commands

- final GL2 gl = drawable.getGL().getGL2()
  - Get OpenGL context for canvas
  - GL2 object exposes underlying C API for OpenGL
  - API calls: gl.glSomeCommandName

- gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f)
  - Set black as the color to use when clearing the screen

- gl.glClear(GL2.GL_COLOR_BUFFER_BIT)
  - Clear the display buffer using the color given by glClearColor
Framework Commands

- `Program.use(gl, program)`
  - Set which shader program the pipeline will use to draw geometry
  - More about this on another day...
- `program.setColor(gl, 1.0f, 1.0f, 1.0f);`
  - Tell shader program to draw geometry in white
- `boxArray.draw(gl)`
  - Draw the geometry described by `boxArray`
- `Program.unuse(gl)`
  - Tell OpenGL we are done drawing for now

Each of these has OpenGL commands under the hood
Framework Commands

- Program.use(gl, program)
  - Set which shader program the pipeline will use to draw geometry
  - More about this on another day...
- program.setColor(gl, 1.0f, 1.0f, 1.0f);
  - Tell shader program to draw geometry in white
- boxArray.draw(gl)
  - Draw the geometry described by boxArray
- Program.unuse(gl)
  - Tell OpenGL we are done drawing for now

Each of these has OpenGL commands under the hood

Framework commands that need access to OpenGL take the gl object as their first argument
Why Have a Framework?

- You write:
  ```java
  boxArray.draw(gl)
  ```

- Framework does:
  ```java
  gl.glBindVertexArray(boxArray.id);
  gl.glBindBuffer(GL2.GL_ELEMENT_ARRAY_BUFFER, boxArray.indexBuffer.id);
  gl.glDrawElements(boxArray.geometryType, boxArray.indexBuffer.numElements, boxArray.indexBuffer.format, 0);
  gl.glBindBuffer(GL2.GL_ELEMENT_ARRAY_BUFFER, 0);
  gl.glBindVertexArray(0);
  ```

- Annoying to retype full sequence of commands for every draw
Framework and GL Resources

- OpenGL API has “objects” that hold rendering resources
  - Geometry, textures, shader programs, etc.
- Framework represents these with Java classes
  - Program (shader programs)
  - VertexBuffer, VertexArray (used to specify geometry)
- Constructing an object creates OpenGL resource
  - Object's data lives in GPU memory
  - Allows faster access while rendering
OpenGL Resources:
Geometry
Demo: Two Boxes
What We're Seeing

- Box on left: two triangles
- Box on right: four lines
Vertices

- Foundation for all geometry
- OpenGL: specify with VertexBuffer
Primitives

- Basic shapes built from vertices; e.g. triangles, lines
  - Assemble to build more complicated shapes
- OpenGL: specify with VertexArray and IndexBuffer
Specifying Vertices: VertexBuffer

- OpenGL object to store vertices
- Info needed:
  - Array of floats representing vertices
  - How many dimensions per vertex (2D? 3D?)

\[
\begin{bmatrix}
-0.5, -0.5 & 0.5, -0.5 & 0.5, 0.5 & -0.5, 0.5, \\
\end{bmatrix}, \quad 2 \text{ floats per vertex }
\]

```
VertexBuffer
-0.5, -0.5  0.5, -0.5  0.5, 0.5  -0.5, 0.5
0 1 2 3
```

\[
= \begin{bmatrix}
3 & 2 \\
\end{bmatrix}
\begin{bmatrix}
0 & 1 \\
\end{bmatrix}
\]
Specifying Vertices: TwoBoxes' `init()`

```java
@Override
public void init(GLAutoDrawable drawable) {
    if(initialized)
        return;

    GL2 gl = drawable.getGL().getGL2();

    // define vertex positions
    float [] vertexPositions = {
        -0.5f, -0.5f, // vertex 0
        0.5f, -0.5f,  // vertex 1
        0.5f,  0.5f,  // vertex 2
        -0.5f,  0.5f  // vertex 3
    };

    VertexBuffer vertexBuffer =
        new VertexBuffer(gl, vertexPositions, 2);

    ...
```
Grouping Vertices into Primitives

- VertexBuffer gives vertices in some order

<table>
<thead>
<tr>
<th>-0.5, -0.5</th>
<th>0.5, -0.5</th>
<th>0.5, 0.5</th>
<th>-0.5, 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- Can re-order vertices to form primitives, so that:
  - Every three vertices form a triangle
    
    \[ [0, 1, 2, \ 0, 2, 3] \]

  - Every two vertices form a line
    
    \[ [0, 1, \ 1, 2, \ 2, 3, \ 3, 0] \]
Grouping Vertices: IndexBuffer

- OpenGL object to store sequence of vertex indices
- Info needed:
  - List of integer indices

\[
[0, 1, 2, \ 0, 2, 3]
\]

\[
[0, 1, \ 1, 2, \ 2, 3, \ 3, 0]
\]
Ways to Group: GL Primitives

- OpenGL declares several primitive types
- Determine how to group a sequence of verts into primitives

(adapted from http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html)
Ways to Group: GL Primitives

- OpenGL declares several primitive types
- Determine how to group a sequence of verts into primitives

(adapted from http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html)
Putting it Together

- VertexBuffer: what the vertices are
- IndexBuffer: in what order to put them
- Primitive Type: how to group ordered vertices into primitives

Together, fully describes geometry

<table>
<thead>
<tr>
<th>VertexBuffer</th>
<th>-0.5, -0.5</th>
<th>0.5, -0.5</th>
<th>0.5, 0.5</th>
<th>-0.5, 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndexBuffer</td>
<td>0, 1, 2, 0, 2, 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primitive Type</td>
<td>GL_TRIANGLES</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Putting it Together: VertexArray

- OpenGL object to fully describe a piece of geometry
- Can be drawn by OpenGL
- Info needed:
  - VertexBuffer, IndexBuffer, primitive type

VertexArray

<table>
<thead>
<tr>
<th>VertexBuffer</th>
<th>-0.5, -0.5</th>
<th>0.5, -0.5</th>
<th>0.5, 0.5</th>
<th>-0.5, 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

IndexBuffer: 0, 1, 2, 0, 2, 3

Primitive Type: GL_TRIANGLES
VertexArray in TwoBoxes' init()

...  // earlier, filled VertexBuffer vertexPositions

int [] linesIndices = {
    0, 1,
    1, 2,
    2, 3,
    3, 0
};
int [] trianglesIndices = {
    0, 1, 2,
    0, 2, 3
};
// make index buffers
linesBuffer = new IndexBuffer(gl, linesIndices);
trianglesBuffer = new IndexBuffer(gl, trianglesIndices);
...

VertexArray in TwoBoxes' init()

...  
// earlier: made VertexBuffer vertexBuffer  
// filled IndexBuffer linesBuffer, trianglesBuffer

boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);  
boxArray.setIndexBuffer(gl, trianglesBuffer);  
boxArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);  
boxLinesArray.setIndexBuffer(gl, linesBuffer);  
boxLinesArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);
VertexArray in TwoBoxes' init()

...  
// earlier: made VertexBuffer vertexBuffer  
// filled IndexBuffer linesBuffer, trianglesBuffer

boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);
boxArray.setIndexBuffer(gl, trianglesBuffer);
boxArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);
boxLinesArray.setIndexBuffer(gl, linesBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

• Construct VertexArray and say what primitives we will build
• One will make triangles, the other lines
VertexArray in TwoBoxes' init()

... 
// earlier: made VertexBuffer vertexBuffer
// filled IndexBuffer linesBuffer, trianglesBuffer

boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);
boxArray.setIndexBuffer(gl, trianglesBuffer);
boxArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);
boxLinesArray.setIndexBuffer(gl, linesBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

• Tell array how to order the vertices when building primitives
• Triangles and lines will need different vertex orders
VertexArray in TwoBoxes' init()

...  
// earlier: made VertexBuffer vertexBuffer  
// filled IndexBuffer linesBuffer, trianglesBuffer

boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);  
boxArray.setIndexBuffer(gl, trianglesBuffer);  
boxArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);  
boxLinesArray.setIndexBuffer(gl, linesBuffer);  
boxLinesArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

• Tell arrays what vertex data to use  
• Both boxes use the same buffer of four vertices
VertexArray in TwoBoxes' init()

...  
// earlier: made VertexBuffer vertexBuffer  
// filled IndexBuffer linesBuffer, trianglesBuffer

boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);  
boxArray.setIndexBuffer(gl, trianglesBuffer);  
boxArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);  
boxLinesArray.setIndexBuffer(gl, linesBuffer);  
boxLinesArray.setAttributeBuffer(gl, TwoDimProgram.VERTEX_INDEX, vertexBuffer);

• What is this about?
Vertices are More than Positions

- A vertex has a position
- But it can also have other attributes:
  - Normal vector
  - Color
  - Texture coordinate
  - etc.
- Use multiple VertexBuffers to store this info

<table>
<thead>
<tr>
<th>Positions</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -0.5, -0.5</td>
<td>1, 0, 0</td>
</tr>
<tr>
<td>1 0.5, -0.5</td>
<td>0, 1, 0</td>
</tr>
<tr>
<td>2 0.5, 0.5</td>
<td>0, 0, 1</td>
</tr>
<tr>
<td>3 -0.5, 0.5</td>
<td>1, 1, 0</td>
</tr>
</tbody>
</table>
### The Full Power of VertexArray

- Accepts multiple VertexBuffers specifying different attributes
- Each buffer mapped to some index in array of attributes

<table>
<thead>
<tr>
<th>VertexArray</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array of Vertex Attributes</td>
<td>-0.5, -0.5</td>
<td>0.5, -0.5</td>
<td>0.5, 0.5</td>
<td>-0.5, 0.5</td>
</tr>
<tr>
<td>IndexBuffer</td>
<td>0, 1, 0</td>
<td>0, 1, 0</td>
<td>0, 0, 1</td>
<td>1, 1, 0</td>
</tr>
<tr>
<td>Primitive Type</td>
<td>GL_TRIANGLES</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
0, 1, 2, 0, 2, 3
```
Attribute Indices

- Shader program draws vertices using same array of attributes
- Program declares where it wants each attribute to be

<table>
<thead>
<tr>
<th>VertexArray</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array of Vertex Attributes</strong></td>
<td><strong>0</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-0.5, -0.5</td>
</tr>
<tr>
<td>1</td>
<td>1, 0, 0</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td><strong>IndexBuffer</strong></td>
<td><strong>0, 1, 2, 0, 2, 3</strong></td>
</tr>
<tr>
<td><strong>Primitive Type</strong></td>
<td><strong>GL_TRIANGLES</strong></td>
</tr>
<tr>
<td><strong>0</strong> = Position</td>
<td></td>
</tr>
<tr>
<td><strong>1</strong> = Color</td>
<td></td>
</tr>
<tr>
<td><strong>2</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Vertex Attributes Wanted</strong></td>
<td></td>
</tr>
</tbody>
</table>
Demo: Color Boxes
@Override
public void init(GLAutoDrawable drawable) {

... // fill vertexBuffer as before

float [] vertexColors = {
    1.0f, 0.0f, 0.0f, // vertex 0
    0.0f, 1.0f, 0.0f, // vertex 1
    0.0f, 0.0f, 1.0f, // vertex 2
    1.0f, 1.0f, 0.0f // vertex 3
};

VertexBuffer colorsBuffer =
    new VertexBuffer(gl, vertexColors, 3);
...

ColorBoxes' init()

... // fill IndexBuffers trianglesBuffer and linesBuffer
boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);
boxArray.setIndexBuffer(gl, trianglesBuffer);
boxArray.setAttributeBuffer(gl, TwoDimColorProgram.VERTEX_INDEX, vertexBuffer);
boxArray.setAttributeBuffer(gl, TwoDimColorProgram.COLOR_INDEX, colorsBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);
boxLinesArray.setIndexBuffer(gl, linesBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimColorProgram.VERTEX_INDEX, vertexBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimColorProgram.COLOR_INDEX, colorsBuffer);
ColorBoxes' init()

```javascript
... // fill IndexBuffers trianglesBuffer and linesBuffer
boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);
boxArray.setIndexBuffer(gl, trianglesBuffer);
boxArray.setAttributeBuffer(gl, TwoDimColorProgram.VERTEX_INDEX, vertexBuffer);
boxArray.setAttributeBuffer(gl, TwoDimColorProgram.COLOR_INDEX, colorsBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);
boxLinesArray.setIndexBuffer(gl, linesBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimColorProgram.VERTEX_INDEX, vertexBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimColorProgram.COLOR_INDEX, colorsBuffer);

• Put vertex positions at index where shader program wants them
```
ColorBoxes' init()

```java
... // fill IndexBuffers trianglesBuffer and linesBuffer
boxArray = new VertexArray(gl, GL2.GL_TRIANGLES);
boxArray.setIndexBuffer(gl, trianglesBuffer);
boxArray.setAttributeBuffer(gl, TwoDimColorProgram.VERTEX_INDEX, vertexBuffer);
boxArray.setAttributeBuffer(gl, TwoDimColorProgram.COLOR_INDEX, colorsBuffer);

boxLinesArray = new VertexArray(gl, GL2.GL_LINES);
boxLinesArray.setIndexBuffer(gl, linesBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimColorProgram.VERTEX_INDEX, vertexBuffer);
boxLinesArray.setAttributeBuffer(gl, TwoDimColorProgram.COLOR_INDEX, colorsBuffer);

• Put vertex colors at index where shader program wants them
```
Transformations
Representing Transformations

- We will use Matrix3f, Matrix4f classes
  - Set matrix contents:
    ```java
    Matrix3f translate = new Matrix3f(1.0f, 0.0f, 3.0f,
                                      0.0f, 1.0f, -4.0f,
                                      0.0f, 0.0f, 1.0f);
    ```
  - Copy matrices:
    ```java
    Matrix3f translateAgain = new Matrix3f(translate);
    ```
  - Multiply matrices:
    ```java
    translate.mul(translateAgain);
    // translate = translate * translateAgain
    ```
  - Transform points:
    ```java
    Vertex3f vert = new Vertex3f(2.0f, 0.0f, 1.0f);
    translate.transform(vert); // vert = translate * vert
    ```
Transforms Class

- Provides various useful transformation matrices
  - Identity matrices
  -Translations, scales, rotations
  -Projection matrices
- Command names indicate dimension:
  - `translate3DH()`: 4x4 matrix, expects 3D homogeneous coords
  - `identity3D()`: 3x3 matrix, expects 3D non-homogeneous coords
Transforming a Vertex

- VertexArray gives vertices to shader program
- Program transforms them onto screen before drawing

\[
\text{Screen position} = \begin{bmatrix}
\text{Transformation} \\
px \\
py \\
pz \\
1
\end{bmatrix}
\]
Transforming a Vertex

- Applies two transforms: Projection and ModelView
- Very basic description now; will discuss topic more later

\[
\text{Screen position} = \begin{bmatrix}
\text{Projection} & \text{ModelView}
\end{bmatrix} \begin{bmatrix}
px \\
py \\
pz \\
1
\end{bmatrix}
\]
ModelView matrix

- Transforms vertex into coordinates of the viewer
- Modify matrix to transform objects to different places
Projection matrix

- Projects point down onto camera plane
- Usually set matrix once at beginning of display()
Setting Matrices

- You tell the shader program what matrices to use
  - program.setProjection(projectionMatrix);
  - program.setModelView(modelViewMatrix);

- Program keeps drawing with the same matrices until you change them

- Common use pattern:
  - program.setModelView(transformForThisObject);
  - thisObject.draw()
Demo: Sierpinski Triangle
@Override
public void display(GLAutoDrawable drawable) {
    final GL2 gl = drawable.getGL().getGL2();

    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT);

    Program.use(gl, program);

    program.setProjection(gl,
        Transforms.ortho2DH(-1.0f, 1.0f, -1.0f, 1.0f));
    program.setColor(gl, 1.0f, 1.0f, 1.0f);

    sierpinski(gl, Transforms.identity2DH(), 8);

    Program.unuse(gl);
}
Recursively Drawing Fractal

• Define “levels” of Sierpinski triangles

Recursive function sierpinski(gl, modelView, k)
  – Level 0: a single triangle
  – Build level k recursively from three copies of level k-1
  – Use modelView matrix to transform the three copies

• Example of drawing a hierarchy
void sierpinski(GL2 gl, Matrix3f modelView, int k)
{
    if (k == 0)
        triangle(gl, modelView);
    else
    {
        Matrix3f next;

        // make next = modelView * (transform to top corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(0.0f, 0.5f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);

        // make next = modelView * (transform to bottom right corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(0.25f, -0.25f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);

        // make next = modelView * (transform to bottom left corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(-0.25f, -0.25f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);
    }
}

void sierpinski(GL2 gl, Matrix3f modelView, int k)
{
    if (k == 0)
        triangle(gl, modelView);
}
```cpp
void sierpinski(GL2 gl, Matrix3f modelView, int k) {
    // base case
    if (k == 0) {
        // do nothing
    } else {
        Matrix3f next;
        // make next = modelView * (transform to top corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(0.0f, 0.5f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);
    }
}
```
void sierpinski(GL2 gl, Matrix3f modelView, int k) {
    
    else {
        Matrix3f next;
        
        // make next = modelView * (transform to top corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(0.0f, 0.5f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);
    }
}
void sierpinski(GL2 gl, Matrix3f modelView, int k) {

    else {
        Matrix3f next;

        // make next = modelView * (transform to top corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(0.0f, 0.5f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);
    }
}

}
void sierpinski(GL2 gl, Matrix3f modelView, int k)
{
    if (k == 0)
        triangle(gl, modelView);
    else
    {
        Matrix3f next;

        // make next = modelView * (transform to top corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(0.0f, 0.5f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);

        // make next = modelView * (transform to bottom right corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(0.25f, -0.25f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);

        // make next = modelView * (transform to bottom left corner)
        next = new Matrix3f(modelView);
        next.mul(Transforms.translate2DH(-0.25f, -0.25f / sqrt(3.0f)));
        next.mul(Transforms.scale2DH(0.5f));
        sierpinski(gl, next, k-1);
    }
}