Intro to OpenGL

CS4620 Lecture 14

Guest Instructor: Nicolas Savva
What is OpenGL?

- **Open Graphics Library**
- A low level API for 2D/3D rendering with the Graphics Hardware (GPU)
- Cross-platform (Windows, OS X, Linux, iOS, Android, ...)

- Developed by SGI in 1992
  - 2014: OpenGL 4.5
  - 2008: OpenGL 3.0
  - 2006: OpenGL 2.1
- Main alternative: DirectX/3D
How does it fit in?

- Tap massive power of GPU hardware to render images
- Use GPU without caring about the exact details of the hardware
CPU and GPU memory

CPU

SYSTEM MEMORY

GPU

SHADERS

GPU MEMORY

Display

BUS
What OpenGL does for us

- Controls GPU
- Lets user specify resources...:
  - Geometry (vertices and primitives)
  - Textures
  - Shaders (programmable pieces of rendering pipeline)
  - Etc.
- ...and use them:
  - Rasterize and draw geometry
How we will use OpenGL

- **OpenGL version**
  - We use 2.x-3.x (plus extensions)
  - Code avoids older, deprecated parts of OpenGL standard

- **LWJGL**
  - Lightweight Java Game Library
  - Java bindings for OpenGL API

- **CS 4620/4621 Framework**
  - Simplifies creating and using OpenGL resources
LWJGL

- OpenGL originally written for C.
- LWJGL contains OpenGL binding for Java
  www.lwjgl.org/

- Gives Java interface to C OpenGL commands
- Manages framebuffer
  (framebuffer: a buffer that holds the image that is displayed on the monitor)
MainGame

• A window which can display GameScreens
• Initializes OpenGL context
• Forwards keyboard and mouse events to the event dispatcher

• Usage
  – Inherit from MainGame and implement methods
  – Create instance and call run method
@Override
protected void buildScreenList() {
    // Create objects inherited from GameScreen and
    // initialize screenList attribute
}

@Override
protected void fullInitialize() {
    // Code Executed Before Window Is Created
}

@Override
protected void fullLoad() {
    // Code Executed With An Active OpenGL Context
}
GameScreen

- Can display images created by OpenGL
- OpenGL “context”
  - Stores OpenGL state (geometry, buffers, etc.)

- Usage:
  - Inherit from class and implement methods
  - Create instance in MainGame.buildScreenList
@Override
public void update(GameTime gameTime) {
    // Animation: Update position of scene objects, camera
}

@Override
public void draw(GameTime gameTime) {
    // Drawing: Use LWJGL to draw to the screen
}

@Override
public void onEntry(GameTime gameTime) {
    // Initialization code
}

@Override
public void onExit(GameTime gameTime) {
    // Destruction, free allocated resources here
}
Events

• MainGame can trigger certain events
  – Something happens (e.g. user resizes window)
  – MainGame forwards event to the event dispatcher
    – KeyboardEventDispatcher
    – MouseEventDispatcher
  – Objects interested in the event can sign up as listeners
    – e.g. KeyboardEventDispatcher.OnKeyPressed.add(…)

• These events let us interact with OpenGL
OpenGL Commands and Resources
Demo: Hello World!
Example: Hello World's draw()

```java
@Override
public void draw(GameTime gameTime) {
    GL11.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    GL11.glClear(GL11.GL_COLOR_BUFFER_BIT);

    program.use();

    GLUniform.setST(program.getUniform("VP"),
                     new Matrix4(), false);
    GLUniform.set(program.getUniform("uGridColor"),
                 new Vector4(1, 1, 1, 1));

    vb.useAsAttrib(program.getAttribute("vPos"));
    ib.bind();
    GL11.glDrawElements(GL11.GL_TRIANGLES, indexCount,
                        GLType.UnsignedInt, 0);
    ib.unbind();

    GLProgram.unuse();
}
```
Framework Commands

- `program.use()`
  - Set which shader program the pipeline will use to draw geometry
- `GLUniform.setST(program.getUniform("VP"), …)`
  - Tell shader program to use the specified transformation as “VP”
- `GLUniform.set(program.getUniform("uGridColor"), …)`
  - Tell shader program to use the specified color as “uGridColor”
- `GLProgram.unuse()`
  - Tell OpenGL we are done drawing for now

- Each of these has OpenGL commands under the hood
Framework Commands

- vb.useAsAttrib(program.getAttribute("vPos"))
  - Tell shader program to use “vb” as vertex buffer and access vertex position using “vPos”

- ib.bind(), ib.unbind()
  - Bind (and unbind) the index buffer to tell OpenGL about how we use the vertices in the vertex buffer
Why Have a Framework?

• You write:

```
vb.useAsAttrib(program.getAttribute("vPos"));
```

• Framework does:

```
GL15.glBindBuffer(GL11.GL_ARRAY_BUFFER, vb.id);
GL15.glEnableVertexAttribArray(program.getAttribute("vPos"));
GL15.glVertexAttribPointer(program.getAttribute("vPos"), componentCount,
                          componentFormat, norm, elementByteSize, offset * elementByteSize);
```

• 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
  - GLProgram (shader programs)
  - GLBuffer (used to specify geometry)
- Constructing an object creates OpenGL resource
  - Object's data lives in GPU memory
  - Allows faster access while rendering
OpenGL Commands

• Get OpenGL context that is already initialized
• API calls: glxx.glSomeCommandName
• GL11.glClearColor(0.0f, 0.0f, 0.0f, 1.0f)
  - Set black as the color to use when clearing the screen
• GL11.glClear(GL11.GL_COLOR_BUFFER_BIT)
  - Clear the display buffer using the color given by glClearColor
• GL11.glDrawElements(…)
  - Draw primitives (now triangles)
Command Naming

• In C,
  • commands = functions
  • No two functions can have the same name
  • Some commands take different arguments but do the same thing
• All are commands of the form:

\[ \text{gl } \langle \text{name} \rangle \{1234\} \{b s i f d ub us ui\} \{v\} \]
### Argument Types in Command Names

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Data Type</th>
<th>Typical Corresponding C-Language Type</th>
<th>OpenGL Type Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>8-bit integer</td>
<td>signed char</td>
<td>GLbyte</td>
</tr>
<tr>
<td>s</td>
<td>16-bit integer</td>
<td>short</td>
<td>GLshort</td>
</tr>
<tr>
<td>i</td>
<td>32-bit integer</td>
<td>long</td>
<td>GLint, GLsizei</td>
</tr>
<tr>
<td>f</td>
<td>32-bit floating-point</td>
<td>float</td>
<td>GLfloat, GLclampf</td>
</tr>
<tr>
<td>d</td>
<td>64-bit floating-point</td>
<td>double</td>
<td>GLdouble, GLclampd</td>
</tr>
<tr>
<td>ub</td>
<td>8-bit unsigned integer</td>
<td>unsigned char</td>
<td>GLubyte, GLboolean</td>
</tr>
<tr>
<td>us</td>
<td>16-bit unsigned integer</td>
<td>unsigned short</td>
<td>GLushort</td>
</tr>
<tr>
<td>ui</td>
<td>32-bit unsigned integer</td>
<td>unsigned long</td>
<td>GLuint, GLenum, GLbitfield</td>
</tr>
</tbody>
</table>
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 GLBuffer
Primitives

- Basic shapes built from vertices; e.g. triangles, lines
  - Assemble to build more complicated shapes
- OpenGL: specify both with GLBuffer
GLBuffer

- OpenGL object to store arrays like vertex positions, vertex colors, indices

- We have to specify:
  - How many component per element
    - Color: 3D vector
    - Position: 2D/3D vector
    - Index: 1D
  - Type of stored element components (int, float, double, …)
  - Array of element components
    - The stored data itself
Specifying Vertices

- GLBuffer: store sequence of vertex positions
- Info needed:
  - Array of floats representing vertices
  - How many dimensions per vertex (2D? 3D?)

( [-0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5], 2 floats per vertex )

-0.5, -0.5
0.5, -0.5
0.5, 0.5
-0.5, 0.5

GLBuffer

\[
\begin{array}{cccc}
-0.5, -0.5 & 0.5, -0.5 & 0.5, 0.5 & -0.5, 0.5 \\
0 & 1 & 2 & 3 \\
\end{array}
\]

=  

\[
\begin{array}{cc}
\bullet & \bullet \\
3 & 2 \\
\bullet & \bullet \\
0 & 1 \\
\end{array}
\]
Specifying Vertices: TwoBoxes' init()

```java
@Override
public void onEntry(GameTime gameTime) {
    // 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
    };

    GLBuffer vertexBuffer = GLBuffer.createAsVertex(
        vertexPositions, 2, BufferUsageHint.StaticDraw);
    ...
```
Grouping Vertices into Primitives

- GLBuffer gives vertices in some order
  
<table>
<thead>
<tr>
<th>GLBuffer</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5, -0.5</td>
<td>0.5, -0.5</td>
<td>0.5, 0.5</td>
<td>-0.5, 0.5</td>
<td></td>
</tr>
</tbody>
</table>

  \[
  \begin{array}{c|c|c|c}
  0 & 1 & 2 & 3 \\
  \end{array}
  \]

- 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

- GLBuffer: 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]

GLBuffer
Ways to Group: GL Primitives

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

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

- GLBuffer 0: what the vertices are
- GLBuffer 1 (index array): in what order to put them
- Primitive Type: how to group ordered vertices into primitives

Together, fully describes geometry

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

- Bind buffer elements to vertex attributes
- Bind indices to buffer in OpenGL
- Draw using OpenGL
- Info needed:
  - GLBuffer with vertex attributes, GLBuffer with indices, primitive type

<table>
<thead>
<tr>
<th>GLBuffer 0</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>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

| GLBuffer 1   | 0, 1, 2, 0, 2, 3 |

Primitive Type: GL_TRIANGLES
Index arrays in TwoBoxes' onEntry()

... // earlier, filled GLBuffer vertexPositions

```java
int [] linesIndices = {
    0, 1,
    1, 2,
    2, 3,
    3, 0
};
int [] trianglesIndices = {
    0, 1, 2,
    0, 2, 3
};
// make index buffers
ibLines = GLBuffer.createAsIndex(linesIndices,
BufferUsageHint.StaticDraw);
indexCountLines = linesIndices.length;
ibTriangles = GLBuffer.createAsIndex(trianglesIndices,
BufferUsageHint.StaticDraw);
indexCountTriangles = trianglesIndices.length;
...```
TwoBoxes' draw()

...  
// Use box vertices we defined before  
vb.useAsAttrib(program.getAttribute("vPos"));  

// Setup transformations  
...

// Binding indices and drawing  
  ibTriangles.bind();  
  GL11.glDrawElements(GL11.GL_TRIANGLES,  
                      indexCountTriangles,  
                      GLType.UnsignedInt, 0);  
  ibTriangles.unbind();  

// Setup transformations  
...

// Binding indices and drawing  
  ibLines.bind();  
  GL11.glDrawElements(GL11.GL_LINES,  
                      indexCountLines,  
                      GLType.UnsignedInt, 0);  
  ibLines.unbind();
TwoBoxes' draw()

// Use box vertices we defined before
vb.useAsAttrib(program.getAttribute("vPos"));

// Setup transformations

// Binding indices and drawing
ibTriangles.bind();
GL11.glDrawElements(GL11.GL_TRIANGLES,
                   indexCountTriangles,
                   GLType.UnsignedInt, 0);
ibTriangles.unbind();

// Setup transformations

// Binding indices and drawing
ibLines.bind();
GL11.glDrawElements(GL11.GL_LINES,
                    indexCountLines,
                    GLType.UnsignedInt, 0);
ibLines.unbind();

- Bind index array and say what primitives we will build
- One will make triangles, the other lines
TwoBoxes' draw()

// Use box vertices we defined before
vb.useAsAttrib(program.getAttribute("vPos"));

// Setup transformations

// Binding indices and drawing
  ibTriangles.bind();
  GL11.glDrawElements(GL11.GL_TRIANGLES,
                      indexCountTriangles,
                      GLType.UnsignedInt, 0);
  ibTriangles.unbind();

// Setup transformations

// Binding indices and drawing
  ibLines.bind();
  GL11.glDrawElements(GL11.GL_LINES,
                      indexCountLines,
                      GLType.UnsignedInt, 0);
  ibLines.unbind();

- Tell OpenGL how to order the vertices when building primitives
- Triangles and lines will need different vertex orders
TwoBoxes' draw()

// Use box vertices we defined before
vb.useAsAttrib(program.getAttribute("vPos"));

// Setup transformations
...

// Binding indices and drawing
ibTriangles.bind();
GL11.glDrawElements(GL11.GL_TRIANGLES,
indexCountTriangles,
GLType.UnsignedInt, 0);
ibTriangles.unbind();

// Setup transformations
...

// Binding indices and drawing
ibLines.bind();
GL11.glDrawElements(GL11.GL_LINES,
indexCountLines,
GLType.UnsignedInt, 0);
ibLines.unbind();

• Both boxes use the same buffer with four vertices
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 GLBuffers to store this info
Multiple Vertex Attributes

• We can bind a GLBuffer to a vertex attribute
• We can use multiple attributes per vertex
Attribute Bindings

- Shader program draws vertices using array of attributes
- Program declares a variable (vPos, vColor) for each attribute

<table>
<thead>
<tr>
<th>Array of Vertex Attributes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>1</td>
<td>1, 0, 0</td>
<td>0, 1, 0</td>
<td>0, 0, 1</td>
<td>1, 1, 0</td>
</tr>
</tbody>
</table>

Index Array: 0, 1, 2, 0, 2, 3

Primitive Type: GL_TRIANGLES

Program:
0 = vPos
1 = vColor
2

Vertex Attributes Wanted
Demo: Color Boxes
ColorBoxes' onEntry()

@Override
public void onEntry(GameTime gameTime) {

  ...
  // 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
  };

  vbColor = GLBuffer.createAsVertex(vertexColors, 3,
                                    BufferUsageHint.StaticDraw);

  ...

ColorBoxes' draw()

...  
// Use box vertex positions and colors as we defined before  
// Bind attribute array values to a variable in the shader

    vb.useAsAttrib(program.getAttribute("vPos"));
    vbColor.useAsAttrib(program.getAttribute("vColor"));

    // Draw the two boxes as we did before

    ...

...
Transformations
Representing Transformations

- We will use Matrix3, Matrix4 classes
  - Set matrix contents:
    
    ```java
    Matrix3 translate = new Matrix3(1.0f, 0.0f, 3.0f,
                                   0.0f, 1.0f, -4.0f,
                                   0.0f, 0.0f, 1.0f);
    ```

  - Copy matrices:
    
    ```java
    Matrix3 translateAgain = new Matrix3(translate);
    ```

  - Multiply matrices:
    
    ```java
    translate.mulBefore(translateAgain);
    // translate = translate * translateAgain
    ```

  - Transform points:
    
    ```java
    Vector3 vert = new Vector3(2.0f, 0.0f, 1.0f);
    translate.mulPos(vert);  // vert = translate * vert
    ```
Matrix3/4 Class

- Static functions provide various useful transformation matrices
  - Identity matrices
  - Translations, scales, rotations
  - Projection matrices
Transforming a Vertex

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

\[
\begin{bmatrix}
\text{Screen position}
\end{bmatrix}
= \begin{bmatrix}
\text{Transformation}
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
1
\end{bmatrix}
\]
Transforming a Vertex

- Applies two transforms: Projection and ModelView
- Very basic description now (we have previously discussed this topic)

\[
\text{Screen position} = \begin{bmatrix}
\text{Projection} & \text{ModelView}
\end{bmatrix} \begin{bmatrix}
px \\
py \\
pz \\
1
\end{bmatrix}
\]
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 draw()
Setting Matrices

• You tell the shader program what matrices to use
  - GLUniform.setST(program.getUniform("VP"), tr, false);
  - You can bind a matrix to a variable in the shader, similarly to vertex attributes

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

• Common use pattern:
  - GLUniform.setST(program.getUniform("tr"), transformForThisObject, false);
  - GL11.glDrawElements(…)
  - Define other transformation
  - Draw something else
Demo: Sierpinski Triangle
@Override
public void draw(GameTime gameTime) {
    GL11.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    GL11.glClear(GL11.GL_COLOR_BUFFER_BIT);

    program.use();

    GLUniform.set(program.getUniform("uGridColor"),
                 new Vector4(1, 1, 1, 1));

    // Use box vertices
    vb.useAsAttrib(program.getAttribute("vPos"));

    // Transformation
    Matrix4 tr = new Matrix4();
    tr.mulAfter(Matrix4.createTranslation(
                new Vector3(0.0f, -(float)Math.sqrt(3)/6, 0.0f)));

    sierpinski(ibLines, tr, 10);

    GLProgram.unuse();
}
Recursively Drawing Fractal

- sierpinski(gl, tr, k):
  - Draw a level-k Sierpinski triangle, and
  - transform by tr
  - $k = 0$: draw triangle to the right

- Recursively: at level k, draw three $k-1$ Sierpinski triangles, transforming them to the three corners of the triangle
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
    if (k == 0) {
        GLUniform.setST(program.getUniform("VP"), tr, false);

        // Draw the triangle
        ibLines.bind();
        GL11.glDrawElements(GL11.GL_LINES, indexCountLines, GLType.UINT, 0);
        ibLines.unbind();
    } else {
        Matrix4 next;

        //draw the up triangle
        next = new Matrix4(tr);
        next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
        next.mulAfter(Matrix4.createTranslation(new Vector3(0.0f, 0.5f / (float)Math.sqrt(3.0f), 0.0f)));
        sierpinski(lines, next, k-1);

        //draw the right triangle
        next = new Matrix4(tr);
        next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
        next.mulAfter(Matrix4.createTranslation(new Vector3(0.25f, -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
        sierpinski(lines, next, k-1);

        //draw the left triangle
        next = new Matrix4(tr);
        next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
        next.mulAfter(Matrix4.createTranslation(new Vector3(-0.25f, -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
        sierpinski(lines, next, k-1);
    }
}
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
    if (k == 0) {
        GLUniform.setST(program.getUniform("VP"), tr, false);

        // Draw the triangle
        ibLines.bind();
        GL11.glDrawElements(GL11.GL_LINES, indexCountLines, GLType.UnsignedInt, 0);
        ibLines.unbind();
    } else {
        [...]
    }
}
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
    [...]
} else {
    Matrix4 next;

    // draw the up triangle
    next = new Matrix4(tr);
    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
    next.mulAfter(Matrix4.createTranslation(new Vector3(0.0f, 0.5f / (float)Math.sqrt(3.0f), 0.0f)));
    sierpinski(lines, next, k-1);
}
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
    [...] 
} else {
    Matrix4 next;
    //draw the up triangle
    next = new Matrix4(tr);
    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
    next.mulAfter(Matrix4.createTranslation(new Vector3(0.0f, 0.5f / (float)Math.sqrt(3.0f), 0.0f)));
    sierpinski(lines, next, k-1);
    [...] 
}
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {

    [...]

} else {
    Matrix4 next;

    //draw the up triangle
    next = new Matrix4(tr);
    next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
    next.mulAfter(Matrix4.createTranslation(new Vector3(0.0f, 0.5f / (float)Math.sqrt(3.0f), 0.0f)));
    sierpinski(lines, next, k-1);

    [...]
}

tr

up triangle
public void sierpinski(GLBuffer lines, Matrix4 tr, int k) {
   if (k == 0) {
      GLUniform.setST(program.getUniform("VP"), tr, false);
      // Draw the triangle
      ibLines.bind();
      GL11.glDrawElements(GL11.GL_LINES, indexCountLines, GLType.UnsignedInt, 0);
      ibLines.unbind();
   } else {
      Matrix4 next;
      // draw the up triangle
      next = new Matrix4(tr);
      next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
      next.mulAfter(Matrix4.createTranslation(new Vector3(0.0f,
            0.5f / (float)Math.sqrt(3.0f), 0.0f)));
      sierpinski(lines, next, k-1);
      // draw the right triangle
      next = new Matrix4(tr);
      next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
      next.mulAfter(Matrix4.createTranslation(new Vector3(0.25f,
            -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
      sierpinski(lines, next, k-1);
      // draw the left triangle
      next = new Matrix4(tr);
      next.mulAfter(Matrix4.createScale(new Vector3(0.5f, 0.5f, 0.5f)));
      next.mulAfter(Matrix4.createTranslation(new Vector3(-0.25f,
            -0.25f / (float)Math.sqrt(3.0f), 0.0f)));
      sierpinski(lines, next, k-1);
   }
}
Books and resources