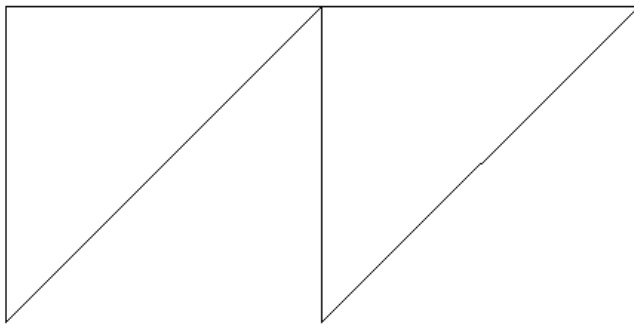


1. What is a **Framebuffer** data structure? What does it contain? What does it represent? How is it used in a graphics rendering pipeline?
2. What is a **Model** data structure? What does it contain? What does it represent? How is it used in a graphics rendering pipeline?
3. What is a **Position** data structure? What does it contain? What does it represent? How is it used in a graphics rendering pipeline?
4. What is a **Scene** data structure? What does it contain? What does it represent? How is it used in a graphics rendering pipeline?
5. Briefly describe the contents of a PPM file. In what ways is a PPM file similar to a **Framebuffer** object? In what ways is a PPM file different from a **Framebuffer** object?
6. Briefly describe each of the following coordinate systems.
 - (a) model coordinates
 - (b) camera coordinates
 - (c) image-plane coordinates
 - (d) pixel-plane coordinates
 - (e) viewport coordinates (in a framebuffer)
 - (f) framebuffer coordinates
7. Draw a simple diagram that explains how to derive the projection formula for the x-coordinate.

8. Consider the following block of code.

```
Model m = new Model("Problem 8");
m.addVertex(new Vertex( 0,  1, 0),
             new Vertex( 1,  0, 0),
             new Vertex(-1, -1, 0));
m.addColor(Color.black);
m.addPrimitive(new LineSegment(0, 1, 0), // vertex, vertex, color
               new LineSegment(1, 2, 0),
               new LineSegment(2, 0, 0));
Position p = new Position(m, "p0", new Vector(0,0,0));
```

- (a) Draw a picture of what would be drawn if we rendered the position **p** built by this code using an orthographic projection looking down the z -axis. In your picture, label the vertices in order from vertex 0 to vertex 2.
 - (b) How many Java objects does this code instantiate, counting all the objects that are composed in other objects? Be sure to include things like **List**, **String**, and **Integer** objects. Draw a detailed picture of the tree relationships between the objects (which objects refer to which objects?).
 - (c) Write a minimal Java program that will compile and run and draw the above position (by “draw” I mean write the **FrameBuffer** to a file that can be viewed).
9. Write code similar to the code in the previous problem that will define the **Model** shown in the following picture. Write a minimal Java program that will compile and run and draw the model.



10. Suppose that we get the following logging output from rendering a Scene.

```

== Begin Rendering of Scene: Problem 10
-- Current Camera:
Camera:
perspective = true
==== Render position: p0
----- Translation vector = [x,y,z] = [ 0.00000 1.00000 -1.00000]
===== Render model: Problem-10
0. Model      : vIndex = 0, (x,y,z) = ( -1.00000 0.00000 0.00000)
0. Model      : vIndex = 1, (x,y,z) = ( 1.00000 0.00000 0.00000)
0. Model      : vIndex = 2, (x,y,z) = ( 0.00000 -1.00000 0.00000)
1. Camera     : vIndex = 0, (x,y,z) = ( -1.00000 1.00000 -1.00000)
1. Camera     : vIndex = 1, (x,y,z) = ( 1.00000 1.00000 -1.00000)
1. Camera     : vIndex = 2, (x,y,z) = ( 0.00000 0.00000 -1.00000)
2. Projected  : vIndex = 0, (x,y,z) = ( -1.00000 1.00000 -1.00000)
2. Projected  : vIndex = 1, (x,y,z) = ( 1.00000 1.00000 -1.00000)
2. Projected  : vIndex = 2, (x,y,z) = ( 0.00000 0.00000 -1.00000)
3. Pixel-plane: vIndex = 0, (x,y,z) = ( 0.50000 200.50000 0.00000)
3. Pixel-plane: vIndex = 1, (x,y,z) = ( 200.50000 200.50000 0.00000)
3. Pixel-plane: vIndex = 2, (x,y,z) = ( 100.50000 100.50000 0.00000)
3. Pixel-plane: LineSegment: ([0, 2], [0, 0])
3. Pixel-plane: LineSegment: ([1, 2], [0, 0])
3. Pixel-plane: cIndex = 0, java.awt.Color[r=255,g=0,b=255]
4. Rasterize: LineSegment: ([0, 2], [0, 0])
    vIndex = 0, (x,y,z) = ( 0.50000 200.50000 0.00000)
    vIndex = 2, (x,y,z) = ( 100.50000 100.50000 0.00000)
    cIndex = 0, java.awt.Color[r=255,g=0,b=255]
    cIndex = 0, java.awt.Color[r=255,g=0,b=255]
4. Rasterize: LineSegment: ([1, 2], [0, 0])
    vIndex = 1, (x,y,z) = ( 200.50000 200.50000 0.00000)
    vIndex = 2, (x,y,z) = ( 100.50000 100.50000 0.00000)
    cIndex = 0, java.awt.Color[r=255,g=0,b=255]
    cIndex = 0, java.awt.Color[r=255,g=0,b=255]
===== End model: Problem-10
==== End position: p0
== End Rendering of Scene: Problem 10

```

- (a) Draw a picture of what was drawn in the `Framebuffer` (assume that the `Framebuffer` is square, that the default `Viewport` is the whole `Framebuffer`, and the `Framebuffer`'s background color is white). In your picture, label the vertices in order from vertex 0 to vertex 2.
- (b) Write a minimal Java program that will compile and run and produce this logging output.

11. Suppose we have a **Framebuffer** object that is 200 pixels by 200 pixels. Suppose we define a 100 pixel high by 200 pixel wide **Viewport** within the **Framebuffer** so that the **Viewport** is centered in the **Framebuffer**.

Let $v = (4, -3, -6)$ be a vertex in the camera coordinate system.

Use the formulas in the accompanying formula sheet to answer these questions.

- (a) What vertex in the image-plane $z = -1$ does the vertex v project to?
- (b) What point in the pixel-plane is the projected v transformed to by the pixel-plane transformation?
- (c) Which pixel in the **Framebuffer**'s **Viewport** represents the vertex v ? (Be sure to use the correct **Viewport** pixel coordinate system.)
- (d) Which pixel in the **Framebuffer** represents the vertex v ? (Be sure to use **Framebuffer** pixel coordinates.)
- (e) Which index in the **Framebuffer**'s pixel-array represents the vertex v ?
- (f) Write a minimal Java program that will compile and run and use its debugging output to verify your calculations.

12. Suppose that a **Framebuffer** is 600 pixels wide by 400 pixels tall and it is stored in row-major form with a single `int` per pixel.

- (a) Where in the `pixel.buffer` is the pixel with **Framebuffer** coordinates (257, 188)? Show your calculation.
- (b) Write a brief, but complete, program to verify your result from part (a).
- (c) Suppose that the **Framebuffer** has a **Viewport** that is 200 pixels wide by 200 pixels tall with its upper left hand corner at **Framebuffer** coordinate (150, 150). Where in the **Framebuffer**'s `pixel.buffer` is the pixel with **Viewport** coordinates (88, 165)? Show your calculation.
- (d) Write a brief, but complete, program to verify your result from part (c).

13. Let $v_1 = (0.4, 0.8, -1)$ and $v_2 = (2.2, -0.3, -1)$ be two vertices in the image-plane $z = -1$ and suppose that they define a **LineSegment**.

- (a) Draw on a piece of paper a simple (2D) picture of the image-plane, the view rectangle, and this line segment.
- (b) Write a program that renders the line segment into an 800 by 800 pixel **Framebuffer**.
- (c) Use your program to determine, from the debugging information, the exact pixel where the line segment leaves the **Framebuffer**. (Hint: Be sure to enable clipping.)

14. Suppose we have a `Framebuffer` object `fb` that is 10 pixels wide by 3 pixels high and this `Framebuffer`'s `Viewport` is set to be all of the `Framebuffer`.

```
Framebuffer fb = new FrameBuffer(10, 3, Color.black);
```

Suppose we have a `Model` object `m` and this `Model` contains a single `LineSegment` that is a diagonal across the view rectangle in the image-plane.

```
Model m = new Model("Problem-14");
m.addVertex(new Vertex(-1, -1, -1),
            new Vertex( 1,  1, -1));
m.addColor(Color.white);
m.addPrimitive(new LineSegment(0, 1, 0));
```

Suppose we render this `Model` into the `Framebuffer`.

```
Scene scene = new Scene("Problem 14");
scene.addPosition(new Position(m, "p0"));
Pipeline.render(scene, fb);
```

- (a) Which logical pixel in the pixel-plane is each `Vertex` of the `LineSegment` rounded to? Show your calculations.
- (b) Which pixels in the `Framebuffer` are turned on (that is, not black) by the rasterizer when it draws this line? Hint: Find the slope, m , of the line in the pixel-plane. Then, in the pixel-plane, use “over 1, up m ” to move from the line’s left endpoint to its right endpoint (exactly how the renderer does it).

Hint: Notice that the above code is a complete program. You can check your answer by using `renderer_1` with the following options set in the program.

```
scene.debug = true;
Rasterize.debug = true;
Pipeline.render(scene, fb);
System.out.println(fb);
```

15. Consider the following two blocks of code. Both blocks draw three line segments in a `Framebuffer` but there are only two `Vertex` objects and one `LineSegment` object. Carefully explain how each block of code manages to render the single `LineSegment` object in three different places in each `Framebuffer`.

<pre> var fb = new FrameBuffer(200, 200); var scene = new Scene("Problem 15a"); var model = new Model("Problem-15a"); model.addVertex(new Vertex(-0.5, 0, -1), new Vertex(+0.5, 0, -1)); model.addColor(Color.white); model.addPrimitive(new LineSegment(0, 1, 0)); scene.addPosition(new Position(model), new Position(model), new Position(model)); scene.getPosition(1) .translate(0, 0.5, 0); scene.getPosition(2) .translation(0, -0.5, 0); Pipeline.render(scene, fb); fb.dumpFB2File("Problem_16a.ppm"); </pre>	<pre> var fb = new FrameBuffer(200, 200); var scene = new Scene("Problem 15b"); var model = new Model("Problem-15b"); model.addVertex(new Vertex(-0.5, 0, -1), new Vertex(+0.5, 0, -1)); model.addColor(Color.white); model.addPrimitive(new LineSegment(0, 1, 0)); scene.addPosition(new Position(model)); Pipeline.render(scene, fb); scene.getPosition(0) .translate(0, 0.5, 0); Pipeline.render(scene, fb); scene.getPosition(0) .translate(0, -0.5, 0); Pipeline.render(scene, fb); fb.dumpFB2File("Problem_16b.ppm"); </pre>
---	---

16. Describe the animation created by the frames that this code generates.

```

final Scene scene = new Scene("Problem 16");
final Model m0 = new Model("horizontal"), m1 = new Model("vertical");
m0.addVertex(new Vertex(-1.0, -1.0, -1.0), new Vertex(1.0, -1.0, -1.0));
m1.addVertex(new Vertex(-1.0, -1.0, -1.0), new Vertex(-1.0, 1.0, -1.0));
m0.addColor(Color.black); m1.addColor(Color.black);
m0.addPrimitive(new LineSegment(0, 1, 0));
m1.addPrimitive(new LineSegment(0, 1, 0));
scene.addPosition(new Position(m0, "p0"), new Position(m1, "p1"));
final FrameBuffer fb = new FrameBuffer(512, 512, Color.white);
for (int j = 0; j <= 50; ++j) {
    fb.clearFB();
    Pipeline.render(scene, fb);
    fb.dumpFB2File(String.format("Review_Problem_17_Frame%03d.ppm", j));
    final Vector t0 = scene.getPosition(0).getTranslation();
    final Vector t1 = scene.getPosition(1).getTranslation();
    scene.getPosition(0).translate(t0.x, t0.y + 0.04, t0.z);
    scene.getPosition(1).translate(t1.x + 0.04, t1.y, t1.z);
}

```

17. The following is the `toString()` output from printing a (very small) `FrameBuffer` object to the console window. Write a Java program that will compile and run and produce this output.

```
FrameBuffer [w=6, h=8]
```

```

r   g   b | r   g   b | r   g   b | r   g   b | r   g   b |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 255 255 0 |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 255 255 255 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0 255   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 0   0   0 | 0   0   0 | 0 255 255 | 0   0   0 |

```

18. The following is the `toString()` output from printing a (very small) `FrameBuffer` object to the console window. Write a Java program that will compile and run and produce this output. (Hint: Think viewport.)

```
FrameBuffer [w=6, h=12]
```

```

r   g   b | r   g   b | r   g   b | r   g   b | r   g   b |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 242  88 145 | 242  88 145 | 242  88 145 | 242  88 145 |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |
0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 | 0   0   0 |

```

19. Suppose we have the following folder structure.

```
\problem_19
|   clean_class_files.cmd
|   tree -F.cmd
\---one
|       One.java
\---two
        Three.java
```

The `One.java` file looks like this.

```
import two.Three;
public class One {
    public static void printMessage() {
        System.out.println("Hello from One.");
    }
    public static void main(String[] args) {
        printMessage();
        Three.printMessage();
    }
}
```

The `Three.java` file looks like this.

```
package two;
public class Three {
    public static void printMessage() {
        System.out.println("Hello from Three.");
    }
}
```

- (a) With a command-line prompt open in the folder `problem_19`, what is the simplest command-line that will compile the file `One.java`? What command-line will then run class `One`? Both command-lines must be run from within folder `problem_19`.
- (b) With a command-line prompt open in the folder `problem_19\one`, what is the simplest command-line that will compile the file `One.java`? What command-line will then run class `One`? Both command-lines must be run from within folder `problem_19\one`.
- (c) With a command-line prompt open in the folder `problem_19\two`, what is the simplest command-line that will compile the file `One.java`? What command-line will then run class `One`? Both command-lines must be run from within folder `problem_19\two`.

20. Suppose we have the following folder structure.

```
\problem_20
|   clean_class_files.cmd
|   tree -F.cmd
\---one
|       One.java
\---two
|       \---three
|           Three.java
```

The `One.java` file looks like this.

```
import three.Three;
public class One {
    public static void printMessage() {
        System.out.println("Hello from One.");
    }
    public static void main(String[] args) {
        printMessage();
        Three.printMessage();
    }
}
```

The `Three.java` file looks like this.

```
package three;
public class Three {
    public static void printMessage() {
        System.out.println("Hello from Three.");
    }
}
```

- (a) With a command-line prompt open in the folder `problem_20`, what is the simplest command-line that will compile the file `One.java`? What command-line will then run class `One`? Both command-lines must be run from within folder `problem_20`.
- (b) With a command-line prompt open in the folder `problem_20\one`, what is the simplest command-line that will compile the file `One.java`? What command-line will then run class `One`? Both command-lines must be run from within folder `problem_20\one`.
- (c) With a command-line prompt open in the folder `problem_20\two`, what is the simplest command-line that will compile the file `One.java`? What command-line will then run class `One`? Both command-lines must be run from within folder `problem_20\two`.