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. When is it preferable to use orthographic (parallel) projection? When is it preferable to use perspective projection? Explain why.

8. Draw a simple diagram that explains how to derive the projection formula for the x-coordinate.

9. Consider the following block of code.

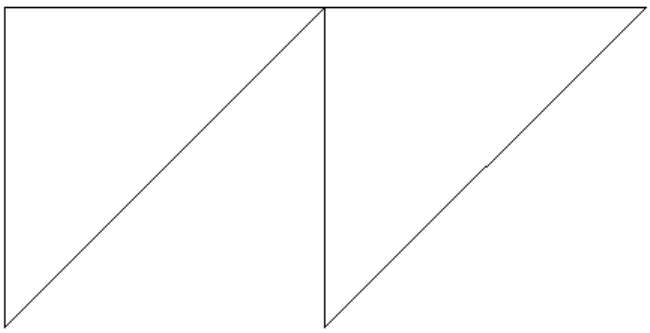
```
Model m = new Model("Problem 9");
m.addVertex(new Vertex( 0,  1, 0),
            new Vertex( 1,  0, 0),
            new Vertex(-1, -1, 0));
m.addPrimitive(new LineSegment(0, 1),
               new LineSegment(1, 2),
               new LineSegment(2, 0));
Position p = new Position(m, "p0", new Vector(0,0,0));
```

   (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 array objects. Draw a detailed picture of what the relationships between the objects "looks like" in the Java heap (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).

10. 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.



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

```
== Begin Rendering of Scene:  ==
-- Current Camera:
Camera:
perspective = true
==== Render position: Problem 11 ====
---- Translation vector = [x,y,z] = [   0.00000     1.00000    -1.00000]
====== Render model: Problem 11 ======
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   200.40005     0.00000)
3. Pixel-plane: vIndex =   1, (x,y,z) = ( 200.40005   200.40005     0.00000)
3. Pixel-plane: vIndex =   2, (x,y,z) = ( 100.45002   100.45002     0.00000)
3. Pixel-plane: LineSegment: [0, 2]
3. Pixel-plane: LineSegment: [1, 2]
4. Rasterize: LineSegment: [0, 2]
   vIndex =   0, (x,y,z) = (   0.50000   200.40005     0.00000)
   vIndex =   2, (x,y,z) = ( 100.45002   100.45002     0.00000)
4. Rasterize: LineSegment: [1, 2]
   vIndex =   1, (x,y,z) = ( 200.40005   200.40005     0.00000)
   vIndex =   2, (x,y,z) = ( 100.45002   100.45002     0.00000)
====== End model: Problem 11 ======
==== End position: Problem 11 ====
== End Rendering of Scene ==
```

(a) Draw a picture of what was drawn in the FrameBuffer (assume that the FrameBuffer is square). 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.

12. 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.

13. 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-array 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 upper left hand corner at `FrameBuffer` coordinate `(150, 150)`. Where in the pixel-array 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).

14. 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`.

15. 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);
```

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 15");
m.addVertex(new Vertex(-1, -1, -1),
            new Vertex( 1,  1, -1));
m.addPrimitive(new LineSegment(0, 1));
```

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

```
Scene scene = new Scene();
scene.addPosition(new Position(m));
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);
```

16. 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`.

```
var fb = new FrameBuffer(200, 200);
var scene = new Scene();
var model = new Model();
model.addVertex(
         new Vertex(-0.5, 0, -1),
         new Vertex(+0.5, 0, -1));
var ls = new LineSegment(0, 1);
model.addPrimitive(ls);
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");
```

```
var fb = new FrameBuffer(200, 200);
var scene = new Scene();
var model = new Model();
model.addVertex(
         new Vertex(-0.5, 0, -1),
         new Vertex(+0.5, 0, -1));
var ls = new LineSegment(0, 1);
model.addPrimitive(ls);
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");
```

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

```
final Scene scene = new Scene();
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));
m0.addPrimitive(new LineSegment(0, 1));
m1.addVertex(new Vertex(-1.0, -1.0, -1.0), new Vertex(-1.0, 1.0, -1.0));
m1.addPrimitive(new LineSegment(0, 1));
scene.addPosition(new Position(m0), new Position(m1));
final FrameBuffer fb = new FrameBuffer(512, 512, Color.darkGray);
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);
}
```

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.

```
FrameBuffer [w=6, h=8]
 r   g   b | 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|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|255 255 255|  0   0   0|  0   0   0|  0   0   0| 77 186 201|
  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| 64  64  64|
  0   0   0|  0   0   0|  0   0   0|  0 255 255|  0   0   0|  0   0   0|
```

19. 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 | 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|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  0   0   0|242  88 145|242  88 145|242  88 145|242  88 145|  0   0   0|
  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|
```