1. (5 pts) The world coordinate system is the system in which the entire scene is placed. Model coordinate systems are different for each object. For example, when drawing a sphere, the sphere’s model coordinate system would place the origin at the center of the sphere, regardless of where the sphere will appear in world coordinates. Through transformations, the various objects are placed in their positions in world coordinates.

2. (3 pts) The horizontal axis is the $x$-axis, the vertical axis is the $y$-axis, and the front-to-back axis is the $z$-axis.

3. (3 pts) Yellow is red and green combined, so write `glColor3f(1.0, 1.0, 0.0)`.

4. (5 pts) In a perspective projection, all the lines of projection converge on the viewpoint. In an orthogonal projection, the lines of projection are parallel to the viewing direction.

5. (5 pts) Clipping is when the parts of a scene that are outside the view frustum are removed before writing to the framebuffer.

6. (5 pts) The aspect ratio is the width of the window divided by the height of the window.

7. (8 pts) The default camera position is the origin and the default orientation is in the direction of the negative $z$-axis.

8. (8 pts) One good reason is because triangles are always convex. No complications arise from nonconvex shapes.

Another good reason is that triangles are always planar. No problems arise from nonplanar polygons.

Larger polygons must be tessellated and the particular tessellation can change while the scene is being viewed. If the programmer specifies the triangles (tessellation), then they will not vary.

Also, the exact shading (coloring) of larger polygons depends on the particular tessellation. This can create an undesirable effect. Of course, that effect will still be there if the programmer does the tessellation, but at least he has some control over it.

9. (8 pts) By using “triangle strip,” each vertex is passed only once. If we pass them as separate triangles, then many of the vertices will be passed multiple times, which is inefficient.

10. (8 pts) You might be drawing a house with a tree in the yard. You could push the current matrix, then draw the house, then pop the matrix. That would return you to the original point. Then push the matrix again, draw the tree,
then pop the matrix. The benefit is that the house and the tree can then be
drawn in either order.

The alternative is to draw the tree “off of” the house. That makes drawing the
tree dependent on drawing the house first. That may not be a big deal when
only two objects are being drawn, but if hundreds of objects are being drawn,
it would be impossible to maintain the program.

11. (8 pts) Without a finite far plane, we would waste time drawing distant objects
that would ultimately become only one pixel. Nobody will notice if that one
pixel is not there.

With the near plane, the problem is with objects that are in the plane of the
camera, perpendicular to the viewing direction. They will be projected to in-
finity. There is also the issue of objects that are behind the camera. They will
be “projected” in front of the camera.

12. (8 pts) Partition the wall into four rectangles. Two of the rectangles would be
the parts of the wall that are to the left and to the right of the window. The
other two rectangles would be the parts of the wall that are above and below
the window. The result would look like one large rectangle with a rectangular
hole in it.

13. (8 pts) The cylinder is first stretched to length 2 in the \( z \) direction. Then it
will be rotated to a vertical position along the positive \( y \)-axis. Then it will be
translated 3 units in the positive \( x \) direction. The final cylinder will have length
2, radius 1, and its axis of symmetry will go from (3, 0, 0) to (3, 2, 0).

14. (14 pts) It would be helpful to first compute the width and height of the window
in world coordinates:

\[
\text{width} = \text{xmax} - \text{xmin} \\
\text{height} = \text{ymax} - \text{ymin}
\]

Then use the ratios \( w / \text{screenWidth} \) and \( h / \text{screenHeight} \) to compute the new
values for \( \text{xmax} \) and \( \text{ymin} \):

\[
\text{newXmax} = \text{xmin} + \left( \frac{w}{\text{screenWidth}} \right) \text{width} \\
\text{newYmin} = \text{ymax} - \left( \frac{h}{\text{screenHeight}} \right) \text{height}
\]

Of course, since the upper-left corner is fixed,

\[
\text{newXmin} = \text{xmin} \\
\text{newYmax} = \text{ymax}
\]