Question 1. What is the difference an orthographic projection and a perspective projection? What is the difference between glFrustum() and glOrtho()? Between glFrustum() and gluPerspective()? If you were given the parameters to glFrustum(), how would you make an equivalent call using gluPerspective()? For the last question, you may assume that in the call to glFrustum(), top == -bottom and right == -left

Answer 1. An orthographic projection has projectors that are all parallel to each other. Orthographic projections are often used when we want to be able to accurately compare the size of two objects, no matter how far each object is from the viewpoint (this is useful, say, in architecture or computer-aided design). A perspective projection has projectors that converge to a single point (known as the perspective reference point, which is the location of the eye in OpenGL). Perspective projections are often used to reproduce what a human eye would see from a certain viewpoint in a 3-D scene, where objects appear smaller the further they are from the eye.

The function glFrustum() constructs the perspective projection matrix defined by its the arguments and multiplies this matrix to the right of the current OpenGL projection matrix. The left, right, top, and bottom boundaries of the frustum in the near plane are specified, as well as the z values of the near and far planes. The function glOrtho() constructs the orthographic projection matrix defined by its the arguments and multiplies this matrix to the right of the current OpenGL projection matrix. The left, right, top, and bottom boundaries of the frustum in the near plane are specified, as well as the z values of the near and far planes.

The functions glFrustum() and gluPerspective() are both used to multiply the current projection matrix on the right by a perspective projection. However, glFrustum() is called by specifying the top, bottom, left, and right bounds of the viewing frustum in the near plane and the values of zNear and zFar. gluPerspective() is called with the field of view in the y direction, the aspect ratio (x / y), and zNear and zFar. In many cases, gluPerspective() can perform the exact same task as glFrustum(), although it is called with a different sets of parameters. There are, however certain non-symmetric frustums that can be specified with glFrustum() that cannot be specified with gluPerspective().

Given the call

\[
\text{glFrustum(left, right, bottom, top, zNear, zFar)};
\]

where top == -bottom and right == -left, the same frustum can be specified with the call

\[
\text{gluPerspective(fovy, aspect, zNear, zFar)};
\]

where

\[
\text{fovy = (180 / PI) * (2 * arctan(top / zNear))};
\]

\[
\text{aspect = (right - left) / (top - bottom)};
\]

/* zNear and zFar are unchanged */
Question 2. What changes would be needed in order to express and then draw a leaf as a fractal? Provide some pseudocode and mention any changes that would be needed in the L-system.

Answer 2. The leaf can be designed as a fractal by modifying the program control flow so that when an actual leaf needs to be drawn in the plant L-system, the plant L-system invokes separate routines that handle the drawing of the leaf L-system. Currently, the control flow of drawPlant() handles just one L-system with one recursive depth. Since the leaf L-system is independent from the plant L-system, so it has its own recursive depth. The routines need to be set up so that an independent recursive depth can be passed in to determine how many iterations to refine a leaf.

One way of determining the drawing depth to pass to the leaf’s L-system is to use the value: \((\text{maximum drawing depth of the plant} - \text{current drawing depth})\) each time the leaf L-system is invoked. Using this technique, leaves that are on older branches of the tree (where the drawing depth is low) will appear more developed than leaves on younger branches (where the drawing depth is high).

Question 3. If you know a 3-D fruit has rotational symmetry about an axis and you were given a triangulation of the 2-D cross-section (this cross section is spanned by the axis of symmetry), how would you go about creating a triangulation for the surface of the 3-D fruit? Provide pseudocode. Assume that the triangulation of the 2-D cross-section is itself symmetric about the axis of symmetry and has one top and one bottom vertex that lies on the axis of symmetry.

Answer 3. Let \(n\) be the level of refinement of the 3-D fruit. Calculate which vertices of the 2-D triangulation are on the outer edge of the 2-D silhouette and label these vertices as outer edge vertices. Now, rotate only these outer edge vertices about the axis of symmetry \(n-1\) times, turning \(\pi / n\) radians each time. At each rotation, record the new location of the outer edge vertices. For the vertex at the very bottom and the very top of the outer edge, connect it in a triangle with the neighboring vertex at rotation \(i\) and the same neighboring vertex at rotation \(i-1\) for all \(i\) on \([1, n-1]\). Add this triangle to the set of faces. For all other vertices, for all \(i\) on \([1, n-1]\), calculate the quad formed by the pair of adjacent vertices on the outer edge at rotation \(i\) with the corresponding vertices at rotation \(i-1\) and store the quad in a temporary variable. Turn this quad into two triangles, and record these two triangles in the set of faces.

After the last iteration, connect the outer edge vertices from rotation \(n-1\) with the vertices from rotation 0. The set of triangles recorded form the faces of the 3-D fruit.

Coding Question: How does the plant change visually when you switch between a perspective projection and an orthographic projection? Why is this?

Coding Answer: The answer to this question will vary based on the exact parameters used when setting up either projection matrix. The main differences are that parts of the plant, under an orthographic projection, shrink during plant rotation only based on how narrow they become in the projection, rather than based on how far they are from the eye. In the case of a perspective projection, objects shrink the further away they are from the eye. This difference occurs because orthographic projectors are parallel with each other and do not originate from a point, while the opposite is true with perspective projectors.

Another difference you may have observed is that certain parts of a 3-D object, say, the z-component of an extrusion, may only be visible in the orthographic projection. This is again because the projectors in a perspective projection originate at a point, so they are somewhat constrained as to where they may go.

On the other hand, perspective projectors continue to radiate “outward” away from the center of the line of sight as they go further and further away from the eye, so in certain cases they may be able to capture more of a scene than orthographic projectors.