1. (10 points) There are several different coordinate systems used in the process of generating a display scene. What are these? Discuss how each of these coordinates is used in the process.

2. (10 points) Consider a rotation $R_1$ about an axis $(1,0,0)$ by angle $120^\circ$ and another rotation $R_2$ about an axis $(0,1,0)$ by angle $120^\circ$. What is the axis and angle of the composite rotation $R_2R_1$?

3. (10 points) To a plane $\hat{n} = (a, b, c, d)^T$ that represents $ax + by + cz + d = 0$, we apply a transformation $M$ which sends each point $p = (x, y, z, 1)^T$ to $Mp$. Show that the transformed plane can be represented as $(M^{-1})^T\hat{n}$.

4. (15 points) Consider a triangle with three corners $(0,0), (1,0), (0,1)$. Discuss how to modify the Cohen-Sutherland line clipping algorithm so that it can compute the squared distance from a point $(x_0, y_0)$ to the triangle.

5. (15 points) Figure 3-36 shows the result of Bresenham’s line-drawing algorithm when the pixel coordinates are addressed at the lower-left corner. What modifications are needed in the corresponding pseudo-code of the algorithm for this change?

6. (20 points) Consider a line clipping against a perspective-projection frustum view volume as shown in Figure 7-46. There is a simple way of combining a one-dimensional Cohen-Sutherland line-clipping algorithm and a two-dimensional NLN line-clipping algorithm as outlined below.

   (a) (10 points) Using the affine transformation from $R^3$ to $R^1$ (as discussed in Quiz #2) that transforms the near clipping plane to 0 and the far clipping plane to 1, discuss how to modify the Cohen-Sutherland algorithm to a 1-dimensional version that clips lines against the near/far clipping planes.

   (b) (10 points) Discuss how to reduce the remaining 3D line-clipping problem to a 2D line-clipping problem with respect to a clipping window.
7. (20 points) Fill in the blanks in the following OpenGL program that generates the Voronoi diagram for line segments in the plane.

```c
#define NUMLINES 7

double lines[NUMLINES][4] = {
    { 0, 0, 0.3, 0 }, { 0.2, -0.1, 0.3, 0.1 }, ...
};
double colors[NUMLINES][3] = {
    { 0.0, 0.0, 1.0 }, { 0.0, 1.0, 0.0 }, ...
};

void init() {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    // Projection Matrix
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1, 1, -1, 1, -1, 1);
    // ModelView Matrix
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // Depth Test
    glEnable(GL_DEPTH_TEST);
}

void drawline(int index) {
    glColor3d(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex3d(lines[index][0], lines[index][1], 0.1);
    glVertex3d(lines[index][2], lines[index][3], 0.1);
    glEnd();
}

void get_normal(int index, double normal[2]) {
    double len;
    normal[0] = lines[index][1] - lines[index][3];
    normal[1] = lines[index][2] - lines[index][0];
    len = sqrt(normal[0] * normal[0] + normal[1] * normal[1]);
    normal[0] /= len;
    normal[1] /= len;
}

void drawvoronoi(int index) {
    double normal[2];
    const double kSize = 2.0;
    get_normal(index, normal);

    // Voronoi
    // ...
glColor3dv(colors[index]);

// (5)

glTranslated(lines[index][0], lines[index][1], (6);

// (7)

// (5)

glTranslated(lines[index][2], lines[index][3], (6);

// (7)

// (8)

glBegin((8);

// 1. draw voronoi
for (int i=0; i<NUMLINES; ++i) 
    drawvoronoi(i);

// 2. draw line
for (int i=0; i<NUMLINES; ++i) 
    drawline(i);

glutSwapBuffers();

}

// Depth Buffer
void display() {
    // (9)
    // (10)

    // for (int i=0; i<NUMLINES; ++i)
    //     drawvoronoi(i);

    // for (int i=0; i<NUMLINES; ++i)
    //     drawline(i);

    glClear((9) | (10);

    glutSwapBuffers();
}

void main(int argc, char **argv) {
    glutInit(&argc, argv);
    ...
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}