

Name: Shaikh Ubaid
Roll no: 180001050
Date: 11 May, 2021
Assignment no: 4

Code:

```
#include <GL/glut.h> /* including the glut library */

#include <cmath>
#include <cstdio>

#include "imageio.h"

#define WINDOW_WIDTH 1200
#define WINDOW_HEIGHT 700
#define ROT_AMT 10.0
#define LIGHT_INTENSITY_AMT 0.2

int texImageWidth;
int texImageHeight;
char texture_file[] = "wall.png";
char backface[] = "Back Face";
char frontface[] = "Front Face";
int initial_x, initial_y, flag;

static GLuint texture_handle[1]; //texture names

float light_intensity = 1.0f;
GLfloat lightColor[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat lightPos[] = {100.0, 100.0, 100.0, 1.0}; /* Infinite light location. */

/* for defining a structure for representing a position vector */
struct Position {
    float x, y, z;
};

Position cameraPos = {0.0, 0.0, 1.0};
Position viewUpVector = {0.0, 1.0, 0.0};

/* this function calculates the square of the given number */
float sqr(float x) {
    return ((x) * (x));
}

/* for drawing given text at the given location */
void renderStrokeFontString(float x, float y, float z, char *string) {
    char *c;
    glPushMatrix();
    glTranslatef((10 * z) * x, y, z);
    glScalef((10 * z) * 0.00025, 0.00025, 1);

    for (c = string; *c != '\0'; c++) {
        glutStrokeCharacter(GLUT_STROKE_ROMAN, *c);
    }
    glPopMatrix();
}

/* load texture image */
GLubyte *makeTexImage(char *loadfile) {
    int i, j, c, width, height;
```

```

GLubyte *texImage;
/* Only works for .png or .tif images.  NULL is returned if errors occurred.
loadImageRGA() is from imageio library downloaded from Internet. */
texImage = loadImageRGBA((char *)loadfile, &width, &height);
texImageWidth = width;
texImageHeight = height;
return texImage;
}

void drawCube() {
    float x0 = -0.1, y0 = -0.1, x1 = 0.1, y1 = 0.1, z0 = 0.1;
    float face[6][4][3] = {
        {{x0, y0, z0}, {x1, y0, z0}, {x1, y1, z0}, {x0, y1, z0}}, /* front */
        {{x0, y1, -z0}, {x1, y1, -z0}, {x1, y0, -z0}, {x0, y0, -z0}}, /* back */
        {{x1, y0, z0}, {x1, y0, -z0}, {x1, y1, -z0}, {x1, y1, z0}}, /* right */
        {{x0, y0, z0}, {x0, y1, z0}, {x0, y1, -z0}, {x0, y0, -z0}}, /* left */
        {{x0, y1, z0}, {x1, y1, z0}, {x1, y1, -z0}, {x0, y1, -z0}}, /* top */
        {{x0, y0, z0}, {x0, y0, -z0}, {x1, y0, -z0}, {x1, y0, z0}} /* bottom */
    };
    for (int i = 0; i < 6; ++i) { /* draw cube with texture images */
        if (i == 0) { /* for front face */
            glBindTexture(GL_TEXTURE_2D, 0); /* do apply texture for this face */
            glColor3f(0.0, 0.0, 0.0);
            renderStrokeFontString(-0.09, 0.0, 0.1, frontface);
            glColor3f(0.4, 0.5, 0.2);
            glBegin(GL_QUADS);
            glVertex3fv(face[i][0]);
            glVertex3fv(face[i][1]);
            glVertex3fv(face[i][2]);
            glVertex3fv(face[i][3]);
            glEnd();
        } else if (i == 1) { /* for back face */
            glBindTexture(GL_TEXTURE_2D, 0); /* do apply texture for this face */
            glColor3f(0.4, 0.8, 0.1);
            renderStrokeFontString(-0.09, 0.0, -0.1, backface);
            glColor3f(1, 1, 1);
            glBegin(GL_QUADS);
            glVertex3fv(face[i][0]);
            glVertex3fv(face[i][1]);
            glVertex3fv(face[i][2]);
            glVertex3fv(face[i][3]);
            glEnd();
        } else {
            glBindTexture(GL_TEXTURE_2D, texture_handle[0]);
            glBegin(GL_QUADS);
            glTexCoord2f(0.0, 0.0);
            glVertex3fv(face[i][0]);
            glTexCoord2f(1.0, 0.0);
            glVertex3fv(face[i][1]);
            glTexCoord2f(1.0, 1.0);
            glVertex3fv(face[i][2]);
            glTexCoord2f(0.0, 1.0);
            glVertex3fv(face[i][3]);
            glEnd();
        }
    }
}

/* Initialize OpenGL Graphics */
void initGL() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); /* Set background color to black and opaque */
    glClearDepth(1.0f); /* Set background depth to farthest */
}

```

```

glGenTextures(1, texture_handle);
GLubyte *texImage = makeTexImage(texture_file);
if (!texImage) {
    fprintf(stderr, "\nError reading %s \n", texture_file);
}
glBindTexture(GL_TEXTURE_2D, texture_handle[0]); /* now we work on handles */
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, texImageWidth,
             texImageHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, texImage);

delete texImage; /* free memory holding texture image */

/* specifying the viewing frustum into the world coordinate system. */
glMatrixMode(GL_PROJECTION);
gluPerspective(/* field of view in degree */ 40.0,
              /* aspect ratio */ 1.0,
              /* Z near */ 1.0, /* Z far */ 10.0);
}

/* Handler for window-repaint event. Called back when the window first appears and
whenever the window needs to be re-painted. */
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear color and depth buffers

    /* Enabling required features of OpenGL */
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glEnable(GL_BLEND);
    glEnable(GL_LINE_SMOOTH);
    glEnable(GL_COLOR_MATERIAL);

    glLineWidth(2.0); /* setting line width for glutStrokeCharacter */

    glShadeModel(GL_FLAT); /* setting shading model to flat */

    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    glCullFace(GL_BACK); /* back-facing polygons are culled */

    GLfloat ambientLight[] = {light_intensity, light_intensity, light_intensity, 1.0f};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightColor); /* Diffuse (non-shiny) light component */
    glLightfv(GL_LIGHT0, GL_SPECULAR, lightColor); /* Specular (shiny) light component */

    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

    glMatrixMode(GL_MODELVIEW); /* To operate on model-view matrix */
    glLoadIdentity();          /* Reset the model-view matrix */

    /* set camera position (PRP) and viewing point (VRP) */
    gluLookAt(cameraPos.x, cameraPos.y, cameraPos.z, 0, 0, 0, viewUpVector.x, viewUpVector.y, viewUpVector.z);

    drawCube(); /* draw our cube */

    glutSwapBuffers(); /* Swap the front and back frame buffers (double buffering) */
}

```

```

    glDisable(GL_TEXTURE_2D);
}

/* this function maps keyboard keys to actions */
GLvoid windowKey(unsigned char key, int x, int y) {
    switch (key) {
        case 27: /* 27 is for esc key, press escape to exit */
            exit(0);
        case 't': /* t is for reset */
            light_intensity = 1.0;
            cameraPos.x = 0.0;
            cameraPos.y = 0.0;
            cameraPos.z = 1.0;
            viewUpVector.x = 0.0;
            viewUpVector.y = 1.0;
            viewUpVector.z = 0.0;
            break;
        case '+': /* increase the light intensity */
            if (light_intensity < 1.0)
                light_intensity += LIGHT_INTENSITY_AMT;
            break;
        case '-': /* decrease the light intensity */
            if (light_intensity > 0.4)
                light_intensity -= LIGHT_INTENSITY_AMT;
            break;
        default:
            printf("Key %d has no action assigned.\n", key);
            break;
    };
    glutPostRedisplay(); /* repaint the window after performing the updation */
}

/* this function is used to normalize the given point/vector */
Position normalize(Position point) {
    Position normalized_point;
    float x = point.x, y = point.y, z = point.z;
    float norm = sqrt(sqr(x) + sqr(y) + sqr(z));
    normalized_point.x = x / norm;
    normalized_point.y = y / norm;
    normalized_point.z = z / norm;
    return normalized_point;
}

/* this function returns the crossproduct of viewUpVector and CameraPosition and thus returns the
viewRightVector */
Position crossProduct() {
    float a, b, c, d, e, f;
    a = viewUpVector.x;
    b = viewUpVector.y;
    c = viewUpVector.z;
    d = cameraPos.x;
    e = cameraPos.y;
    f = cameraPos.z;
    Position viewRightVector = {b * f - e * c, d * c - a * f, a * e - b * d};
    return normalize(viewRightVector);
}

/* this function rotates the given point about the given axis by the given angle */
Position rotateAboutAxis(Position point, float angle, Position axis) {
    float c = cos(angle), s = sin(angle), x = axis.x, y = axis.y, z = axis.z;
    float Mr[4][4] = {
        {(1 - c) * sqr(x) + c, (1 - c) * x * y - s * z, (1 - c) * x * z + s * y, 0},
        {(1 - c) * x * y + s * z, (1 - c) * sqr(y) + c, (1 - c) * y * z - s * x, 0},
    };
}

```

```

        {(1 - c) * x * z - s * y, (1 - c) * y * z + s * x, (1 - c) * sqrt(x) + c, 0},
        {0, 0, 0, 1}};
    Position new_point;
    new_point.x = Mr[0][0] * point.x + Mr[0][1] * point.y + Mr[0][2] * point.z;
    new_point.y = Mr[1][0] * point.x + Mr[1][1] * point.y + Mr[1][2] * point.z;
    new_point.z = Mr[2][0] * point.x + Mr[2][1] * point.y + Mr[2][2] * point.z;
    return normalize(new_point);
}

GLvoid onMouseMotion(int x, int y) {
    /* Calculate the amount of rotation given the mouse movement. */
    float deltaAngleX = (2 * M_PI / glutGet(GLUT_WINDOW_WIDTH)); /* a movement from left to right = 2*PI = 360
    deg */
    float deltaAngleY = (M_PI / glutGet(GLUT_WINDOW_HEIGHT)); /* a movement from top to bottom = PI = 180
    deg */
    float xAngle = (initial_x - x) * deltaAngleX;
    float yAngle = (initial_y - y) * deltaAngleY;

    Position viewRightVector = crossProduct();
    /* Rotate the camera around the pivot point on the first axis */
    Position new_point = rotateAboutAxis(cameraPos, yAngle, viewRightVector);

    /* Rotate the camera around the pivot point on the second axis */
    cameraPos = rotateAboutAxis(new_point, xAngle, viewUpVector);

    viewUpVector = rotateAboutAxis(viewUpVector, yAngle, viewRightVector);

    initial_x = x;
    initial_y = y;
    glutPostRedisplay();
}

/* function which gets activated whenever user presses a mouse button. It sets the initial x and y for use by
onMouseMotion()*/
GLvoid mouseButtonPressed(GLint pressedButton, GLint cur_state, GLint x, GLint y) {
    if (pressedButton != GLUT_LEFT_BUTTON)
        return;

    if (cur_state == GLUT_UP) {
        flag = 0;
    } else {
        flag = 1;
        initial_x = x;
        initial_y = y;
    }
}

/* Handler for window re-size event. Called back when the window first appears and
whenever the window is re-sized with its new width and height */
void reshape(GLsizei width, GLsizei height) { // GLsizei for non-negative integer
    /* Compute aspect ratio of the new window */
    if (height == 0) height = 1; /* To prevent divide by 0 */
    GLfloat aspect = (GLfloat)width / (GLfloat)height;
    glViewport(0, 0, width, height); /* Set the viewport to cover the new window */

    /* Set the aspect ratio of the clipping volume to match the viewport */
    glMatrixMode(GL_PROJECTION); /* To operate on the Projection matrix */
    glLoadIdentity(); /* Reset */
    /* Enable perspective projection with fovy, aspect, zNear and zFar */
    gluPerspective(45.0f, aspect, 0.1f, 100.0f);
}

/* Main function: GLUT runs as a console application starting at main() */

```

```

int main(int argc, char **argv) {
    glutInit(&argc, argv); // Initialize GLUT
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH); // Enable double buffered mode
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT); // Set the window's initial width & height
    glutInitWindowPosition(0, 0); // Position the window's initial top-left
corner
    glutCreateWindow("Cube"); // Create window with the given title
    glutDisplayFunc(display); // Register callback handler for window
re-paint event
    glutReshapeFunc(reshape); // Register callback handler for window re-size
event
    initGL(); // Our own OpenGL initialization
    glutKeyboardFunc(&windowKey); // Our key pressed handler function
    glutMouseFunc(&mouseButtonPressed); // Registering mouse button handler function
    glutMotionFunc(&onMouseMotion); // Registering mouse motion handler function
    glutMainLoop(); // Enter the infinite event-processing loop
    return 0;
}

```

Output:











