1 Programming Log

11/16/23

I spent 1 hour installing GLUT and getting basic OpenGL programs to compile. When trying to compile and link via one call of gcc, I repeatedly got linking errors, so I set up a basic makefile for ease of development.

I then spent about 30 minutes designing the program.

I then spent about 3 hours writing the program. The largest difficulty encountered was with the Rubik's cube. Matrix multiplication was not happening in the order I expected it to happen, and it took me some time to figure out why my rotations were not working.

2 Program Design

```
Two modules
    data
        contain functions that draw all of the things from the requirements
    main
        handle selection of which thing to display, appropriate setup calls
        call displayFunc and mainLoop with the appropriate function from data

functions in data
    primary
        writeName()
            displays name and class info on output
        f1(type)
            draws the 2d function, via points or lines as according to type
        f2(type)
            draws the 3d function, quads/tris and wireframe/solid according to type
        f2_all()
            draws all four configurations of f2 in separate viewports
        rubik(gaps)
            draws a rubik's cube potentially with gaps
        rubik_grid()
            draws a grid of rubik cubes with gaps
    no-argument wrappers for display callbacks
        f1_points()
        f1_lines()
            wrappers that call f1 with the appropriate type
        f2_qw()
        f2_qs()
        f2_tw()
        f2_ts()
            wrappers that call f2 with quads/wire to tris/solid
        rubik_solid()
        rubik_gaps()
            wrappers that call rubik with appropriate arguments
```

3 Source Code

3.1 main.c

```
//main.c, Spencer Butler, CS324 OpenGL, 11/16/2023

#include <stdio.h>
#include <stdlib.h>

#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

#include "data.h"

int main(int argc, char **argv) {
    int item;

    if(argc < 2) {
        printf("Provide an argument to select which item to draw.\n");
        printf("\t1 -- 2D function (points)\n");
        printf("\t2 -- 2D function (lines)\n");
        printf("\t3 -- 3D function (wireframe tris)\n");
        printf("\t4 -- 3D function (all)\n");
        printf("\t5 -- Rubik's cube (no gaps)\n");
        printf("\t6 -- Rubik's cube (with gaps)\n");
        printf("\t7 -- Grid of Rubik's cubes\n");
        return 1;
    }
    item = atoi(argv[1]);

    glutInit(&argc, argv);
    glutCreateWindow("Basic OpenGL");
    glutReshapeWindow(720, 720);
    glClearColor(1.0, 1.0, 1.0, 1.0);

    if(item == 1 || item == 2) {
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-4, 10, -2, 2, -1, 1);
    }

    if(item == 3 || item == 4) {
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
    glOrtho(-8, 8, -8, 8, 0, 30);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(-5.5, -8.5, 6,
              0, 0, 0,
              0, 0, 1);
}

if(item == 5 || item == 6) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1, 1, -1, 1, 0, 10);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(2.5, 3, 2,
              0, 0, 0,
              0, 0, 1);
}

if(item == 7) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-25, 25, -1, 40, 0, 90);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(30, 24, -5,
              0, 0, 0,
              0, 0, 1);
}

switch(item) {
    case 1:
        glutDisplayFunc(f1_points);
        break;

    case 2:
        glutDisplayFunc(f1_lines);
        break;

    case 3:
        glutDisplayFunc(f2_tw);
        break;
```

```
        case 4:
            glutDisplayFunc(f2_all);
            break;

        case 5:
            glutDisplayFunc(rubik_solid);
            break;

        case 6:
            glutDisplayFunc(rubik_gaps);
            break;

        case 7:
            glutDisplayFunc(rubik_grid);
            break;

        default:
            printf("Unrecognized item.\n");
            return 1;
    }

    glutMainLoop();

    return 0;
}
```

3.2 data.h

```
//data.h, Spencer Butler, CS324 OpenGL, 11/16/2023

#ifndef data_h
#define data_h

//constants for f1 type information
#define SB_POINTS (1 << 0)
#define SB_LINES (0)

//constants for f2 type information
#define SB_QUADS (1 << 1)
#define SB_TRIS (0)
#define SB_WIREFRAME (1 << 0)
#define SB_SOLID (0)
```

```
//Draws name and class information
void writeName();

//Draws the 2D function according to type
//SB_POINTS/SB_LINES
void f1(int type);

//Draws the 3D function according to type
//SB_QUADS/SB_TRIS and SB_WIREFRAME/SB_SOLID
void f2(int type);

//Draws all four configurations of the 3D function
void f2_all();

//Draws a rubik's cube with gaps of input-specified width
void rubik(double gaps);

//Draws a grid of rubik's cubes with gaps
void rubik_grid();

//Wrapper for f1(SB_POINTS)
void f1_points();

//Wrapper for f1(SB_LINES)
void f1_lines();

//Wrapper for f2(SB_QUADS | SB_WIREFRAME)
void f2_qw();

//Wrapper for f2(SB_QUADS | SB_SOLID)
void f2_qs();

//Wrapper for f2(SB_TRIS | SB_WIREFRAME)
void f2_tw();

//Wrapper for f2(SB_TRIS | SB_SOLID)
void f2_ts();

//Wrapper for rubik(0.0)
void rubik_solid();

//Wrapper for rubik(0.05)
void rubik_gaps();
```

```
#endif
```

3.3 data.c

```
//data.c, Spencer Butler, CS324 OpenGL, 11/16/2023


#include <math.h>

#include <stdio.h>

#include <GL/glut.h>
#include <GL/gl.h>

#include "data.h"


//amount to scale characters by, since default size seems to be massive
#define CHAR_SCALE (1.0/(152 * 8))

#define PI (3.1415926)

//number of steps to take when drawing f1
#define F1_STEPS (200)

//number of steps per dimension when drawing f2
#define F2_STEPS (30)


//Draws name and class information
void writeName() {
    char name[] = "Spencer Butler";
    char class[] = "CS324 Fall 2023";
    char *p;

    glColor3f(0.0, 1.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0, 2, 0, 2, -1, 1);

    glMatrixMode(GL_MODELVIEW);
```

```
    glPushMatrix();
    glLoadIdentity();
    glTranslatef(0.1, 1.9, 0);
    glScalef(CHAR_SCALE, CHAR_SCALE, CHAR_SCALE);

    for(p = name; *p; p++) {
        glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
    }

    glScalef(1 / CHAR_SCALE, 1 / CHAR_SCALE, 1 / CHAR_SCALE);
    glTranslatef(-0.7, -0.1, 0);
    glScalef(CHAR_SCALE, CHAR_SCALE, CHAR_SCALE);

    for(p = class; *p; p++) {
        glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
    }


    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
}

//Draws the 2D function according to type
//SB_POINTS/SB_LINES
void f1(int type) {
    double x, y;

    writeName();

    if(type & SB_POINTS) {
        glBegin(GL_POINTS);
    } else {
        glBegin(GL_LINES);
    }


    glColor3f(0, 0, 0);

    for(x = -1 * PI; x <= 3 * PI; x += (4 * PI)/F1_STEPS) {
        y = exp(-0.2 * x) * sin(3 * x);
        glVertex2f(x, y);
    }

    glEnd();
```

```
    glFlush();
}

//helper function for f2 -- sets the color according to the z value
void setColor(double z) {
    z = z + 5;
    z = z / 10;
    glColor3f(z, 0, 1 - z);
}

//Draws the 3D function according to type
//SB_QUADS/SB_TRIS and SB_WIREFRAME/SB_SOLID
void f2(int type) {
    double x, y, z, stepSize;
    double a[4][3];
    int i;

    writeName();

    if(type & SB_WIREFRAME) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    } else {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    }

    if(type & SB_QUADS) {
        glBegin(GL_QUADS);
    } else {
        glBegin(GL_TRIANGLES);
    }

    glColor3f(0, 0, 0);

    stepSize = 10.0 / (F2_STEPS);
    for(x = -5; x < 5; x += stepSize) {
        for(y = -5; y < 5; y += stepSize) {
            //printf("(%lf, %lf) \n", x, y);
            a[0][0] = x;
            a[0][1] = y;

            a[1][0] = x;
            a[1][1] = y + stepSize;

            a[2][0] = x + stepSize;
            a[2][1] = y + stepSize;
```

```
            a[3][0] = x + stepSize;
            a[3][1] = y;

            for(i = 0; i < 4; i++) {
                a[i][2] = 5 * cos(2 * a[i][0] + 3 * sin(2 * a[i][1]));
                a[i][2] = a[i][2] * exp(-0.25 * (a[i][0] * a[i][0] + a[i][1] * a[i][1]))
            }

            if(type & SB_QUADS) {
                for(i = 0; i < 4; i++) {
                    setColor(a[i][2]);
                    glVertex3f(a[i][0], a[i][1], a[i][2]);
                }
            } else {
                for(i = 0; i < 4; i++) {
                    if(i == 2) { continue; }
                    setColor(a[i][2]);
                    glVertex3f(a[i][0], a[i][1], a[i][2]);
                }
                for(i = 1; i < 4; i++) {
                    setColor(a[i][2]);
                    glVertex3f(a[i][0], a[i][1], a[i][2]);
                }
            }
        }
    }

    glEnd();
    glFlush();


}

//Draws all four configurations of the 3D function
void f2_all() {
    GLint vp[4];
    glClear(GL_COLOR_BUFFER_BIT);
    glGetIntegerv(GL_VIEWPORT, &(vp[0]));

    glViewport(vp[0], vp[1], vp[2] / 2, vp[3] / 2);
    f2(SB_QUADS | SB_WIREFRAME);

    glViewport(vp[0] + vp[2] / 2, vp[1], vp[2] / 2, vp[3] / 2);
    f2(SB_QUADS | SB_SOLID);
```

```
    glViewport(vp[0] + vp[2] / 2, vp[1] + vp[3] / 2, vp[2] / 2, vp[3] / 2);
    f2(SB_TRIS | SB_WIREFRAME);

    glViewport(vp[0], vp[1] + vp[3] / 2, vp[2] / 2, vp[3] / 2);
    f2(SB_TRIS | SB_SOLID);

    glViewport(vp[0], vp[1], vp[2], vp[3]);

}

//Helper function to rotate among the rubik's cube colors
void incrementColor() {
    static int i = 0;
    static float colors[6][3] = {
        {1, 0, 0},
        {0, 1, 0},
        {0, 0, 1},
        {1, 0, 1},
        {0, 1, 1},
        {1, 0.3, 0}
    };

    glColor3fv(colors[i]);

    i++;
    i = i % 6;
}

//Helper function to draw one face of a rubik's cube
void rubikFace(double gaps) {
    int i, j;
    incrementColor();
    glBegin(GL_QUADS);
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            glVertex3f(i * 1.0/3 + gaps, j * 1.0/3 + gaps, 0);
            glVertex3f((i + 1) * 1.0/3 - gaps, j * 1.0/3 + gaps, 0);
            glVertex3f((i + 1) * 1.0/3 - gaps, (j + 1) * 1.0/3 - gaps, 0);
            glVertex3f(i * 1.0/3 + gaps, (j + 1) * 1.0/3 - gaps, 0);
        }
    }
    glEnd();
}
```

```
//Draws a rubik's cube with gaps of input-specified width
void rubik(double gaps) {
    int i;
    GLfloat active[16];
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    glMatrixMode(GL_MODELVIEW);



    //prepare a matrix that rotates around the center of the cube
    glPushMatrix();
    glLoadIdentity();
    glTranslatef(0.5, 0.5, 0.5);
    glRotatef(90, 1, 0, 0);
    glTranslatef(-0.5, -0.5, -0.5);
    glGetFloatv(GL_MODELVIEW_MATRIX, active);
    glPopMatrix();


    //4 easy faces
    for(i = 0; i < 4; i++) {
        rubikFace(gaps);
        glMultMatrixf(active);
    }

    //need to rotate about a different axis for last two faces
    glPushMatrix();
    glLoadIdentity();
    glTranslatef(0.5, 0.5, 0.5);
    glRotatef(90, 0, 1, 0);
    glTranslatef(-0.5, -0.5, -0.5);
    glGetFloatv(GL_MODELVIEW_MATRIX, active);
    glPopMatrix();

    glMultMatrixf(active);
    rubikFace(gaps);

    glMultMatrixf(active);
    glMultMatrixf(active);
    rubikFace(gaps);

    glMultMatrixf(active);

    glFlush();
```

```
}

//Draws a grid of rubik's cubes with gaps
void rubik_grid() {
    int i, j, k;
    GLfloat x_trans[16];
    GLfloat y_trans[16];
    GLfloat z_trans[16];

    glClear(GL_COLOR_BUFFER_BIT);
    writeName();


    glMatrixMode(GL_MODELVIEW);

    //prepare translation matrices
    glPushMatrix();
    glLoadIdentity();
    glTranslatef(1.2, 0, 0);
    glGetFloatv(GL_MODELVIEW_MATRIX, x_trans);

    glLoadIdentity();
    glTranslatef(0, 1.2, 0);
    glGetFloatv(GL_MODELVIEW_MATRIX, y_trans);

    glLoadIdentity();
    glTranslatef(0, 0, 1.2);
    glGetFloatv(GL_MODELVIEW_MATRIX, z_trans);
    glPopMatrix();

    glPushMatrix();
    for(i = 0; i < 20; i++) {
        glPushMatrix();
        for(j = 0; j < 20; j++) {
            glPushMatrix();
            for(k = 0; k < 20; k++) {
                rubik(0.05);
                glMultMatrixf(z_trans);
            }
            glPopMatrix();
            glMultMatrixf(y_trans);
        }
        glPopMatrix();
        glMultMatrixf(x_trans);
    }
```

```
    glPopMatrix();
}

//Wrapper for f1(SB_POINTS)
void f1_points() {
    glClear(GL_COLOR_BUFFER_BIT);
    f1(SB_POINTS);
}

//Wrapper for f1(SB_LINES)
void f1_lines() {
    glClear(GL_COLOR_BUFFER_BIT);
    f1(SB_LINES);
}

//Wrapper for f2(SB_QUADS | SB_WIREFRAME)
void f2_qw() {
    glClear(GL_COLOR_BUFFER_BIT);
    f2(SB_QUADS | SB_WIREFRAME);
}

//Wrapper for f2(SB_QUADS | SB_SOLID)
void f2_qs() {
    glClear(GL_COLOR_BUFFER_BIT);
    f2(SB_QUADS | SB_SOLID);
}

//Wrapper for f2(SB_TRIS | SB_WIREFRAME)
void f2_tw() {
    glClear(GL_COLOR_BUFFER_BIT);
    f2(SB_TRIS | SB_WIREFRAME);
}

//Wrapper for f2(SB_TRIS | SB_SOLID)
void f2_ts() {
    glClear(GL_COLOR_BUFFER_BIT);
    f2(SB_TRIS | SB_SOLID);
}

//Wrapper for rubik(0.0)
void rubik_solid() {
    glClear(GL_COLOR_BUFFER_BIT);
    writeName();
    rubik(0.0);
}
```

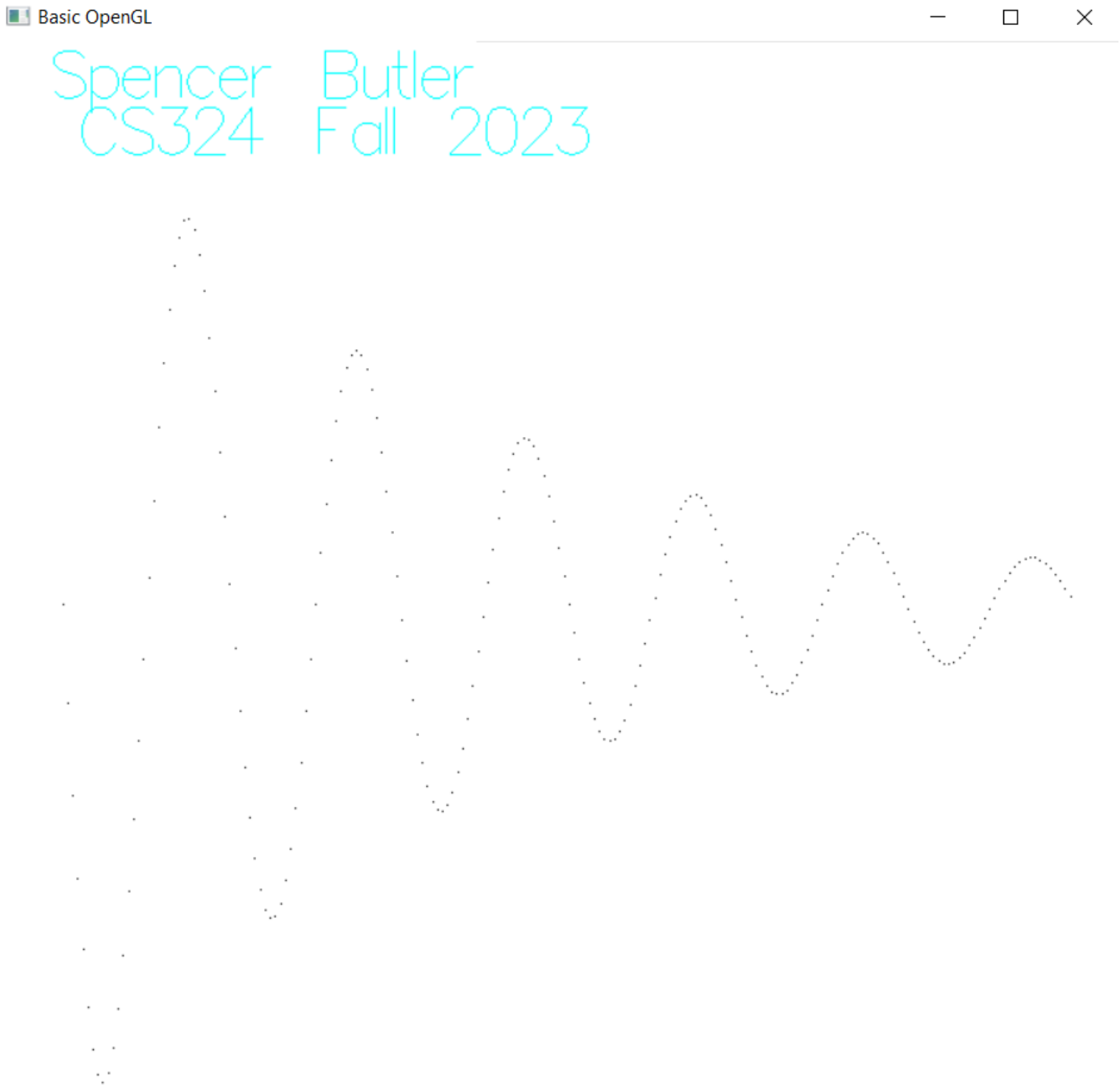
```
//Wrapper for rubik(0.05)
void rubik_gaps() {
    glClear(GL_COLOR_BUFFER_BIT);
    writeName();
    rubik(0.05);
}
```

3.4 makefile

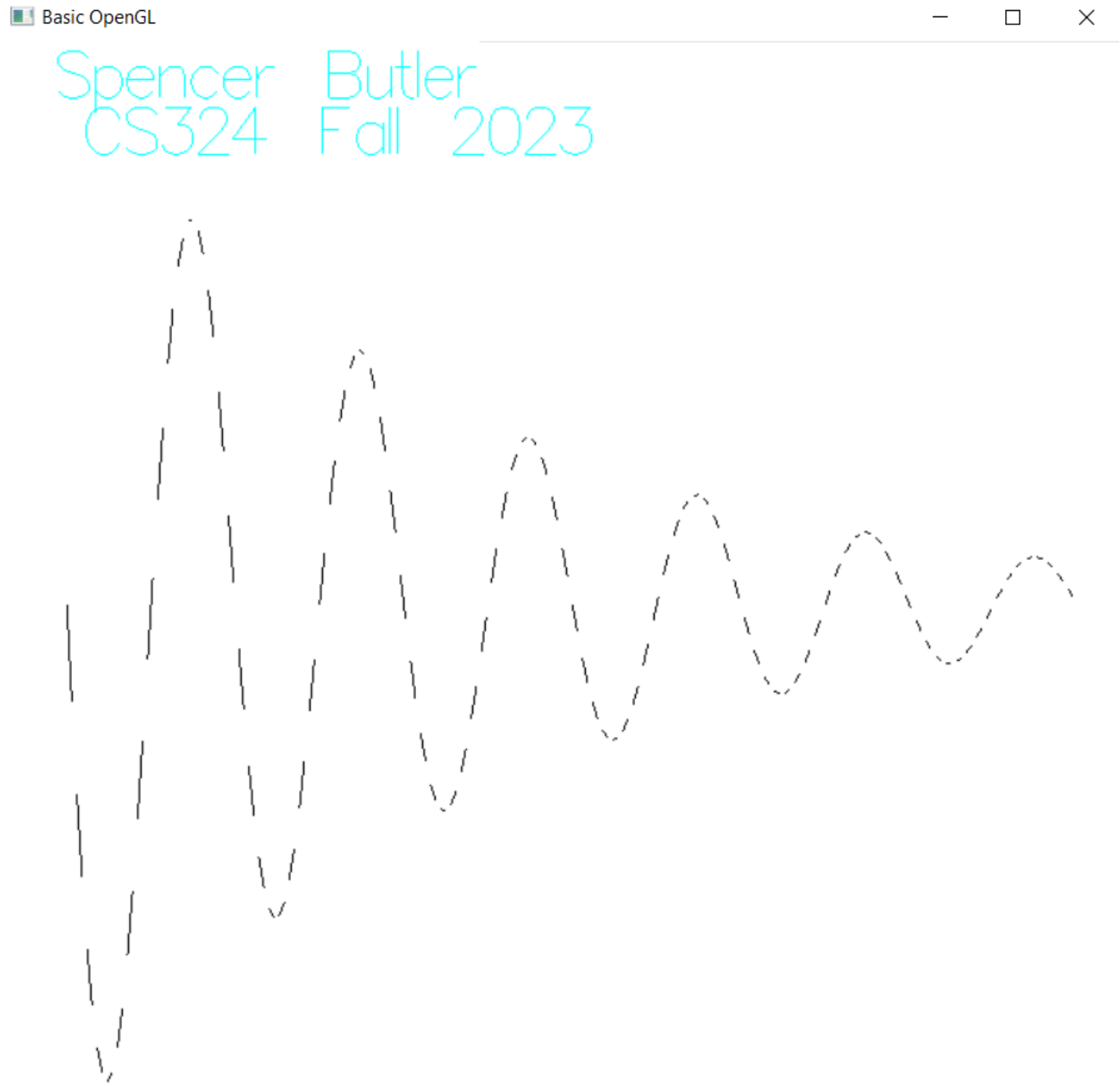
```
all: clean main


data.o: data.h
gcc -c data.c

main.o: data.h
gcc -c main.c

main: main.o data.o
gcc -o main main.o data.o -lfreeglut -lopengl32 -lglu32

clean:
rm *.o
```

4 Output

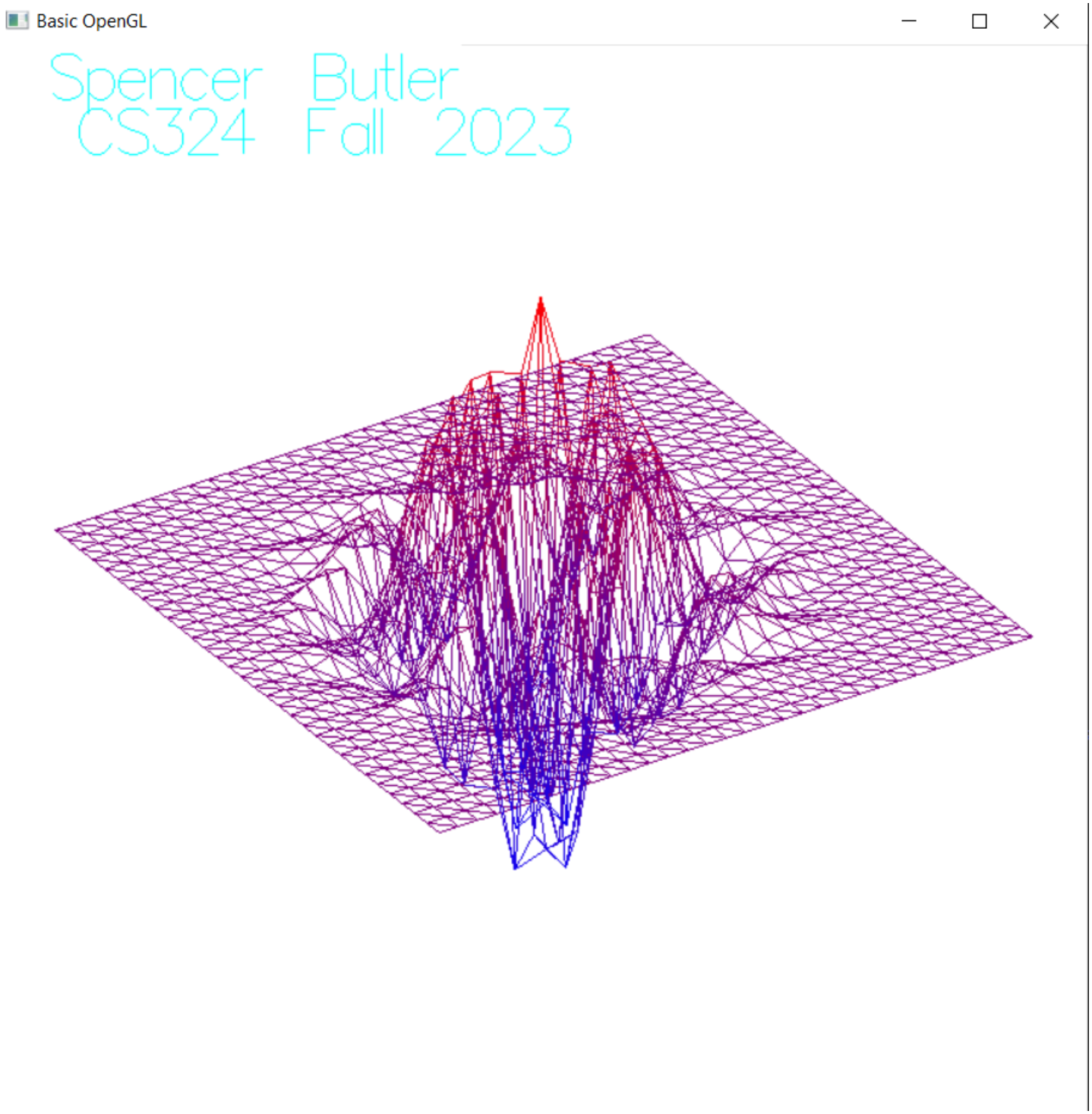
4.1 The 2D function, plotted as points:



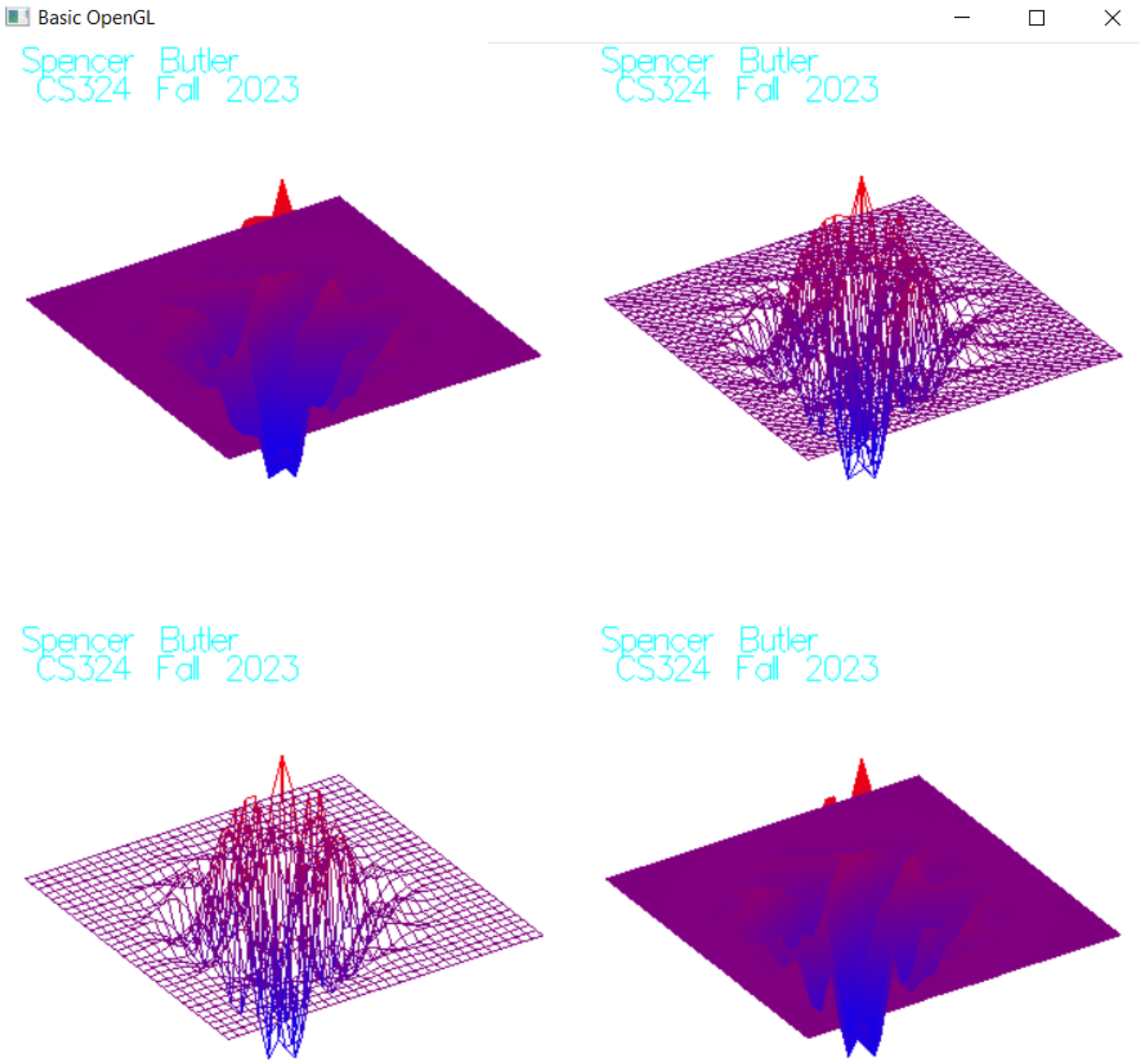
4.2 The 2D function, plotted as lines:



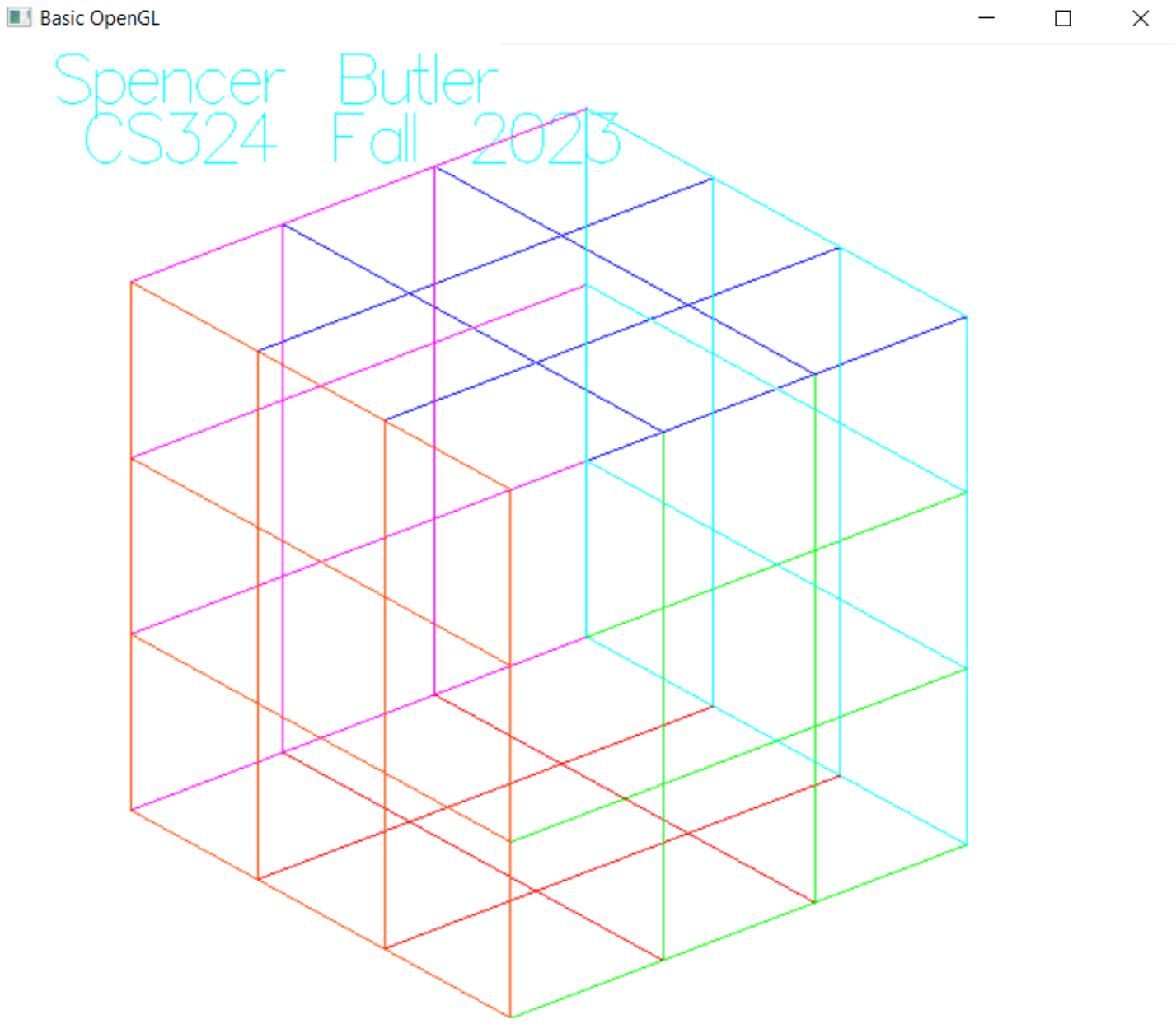
4.3 The 3D function, plotted via wireframe triangles:



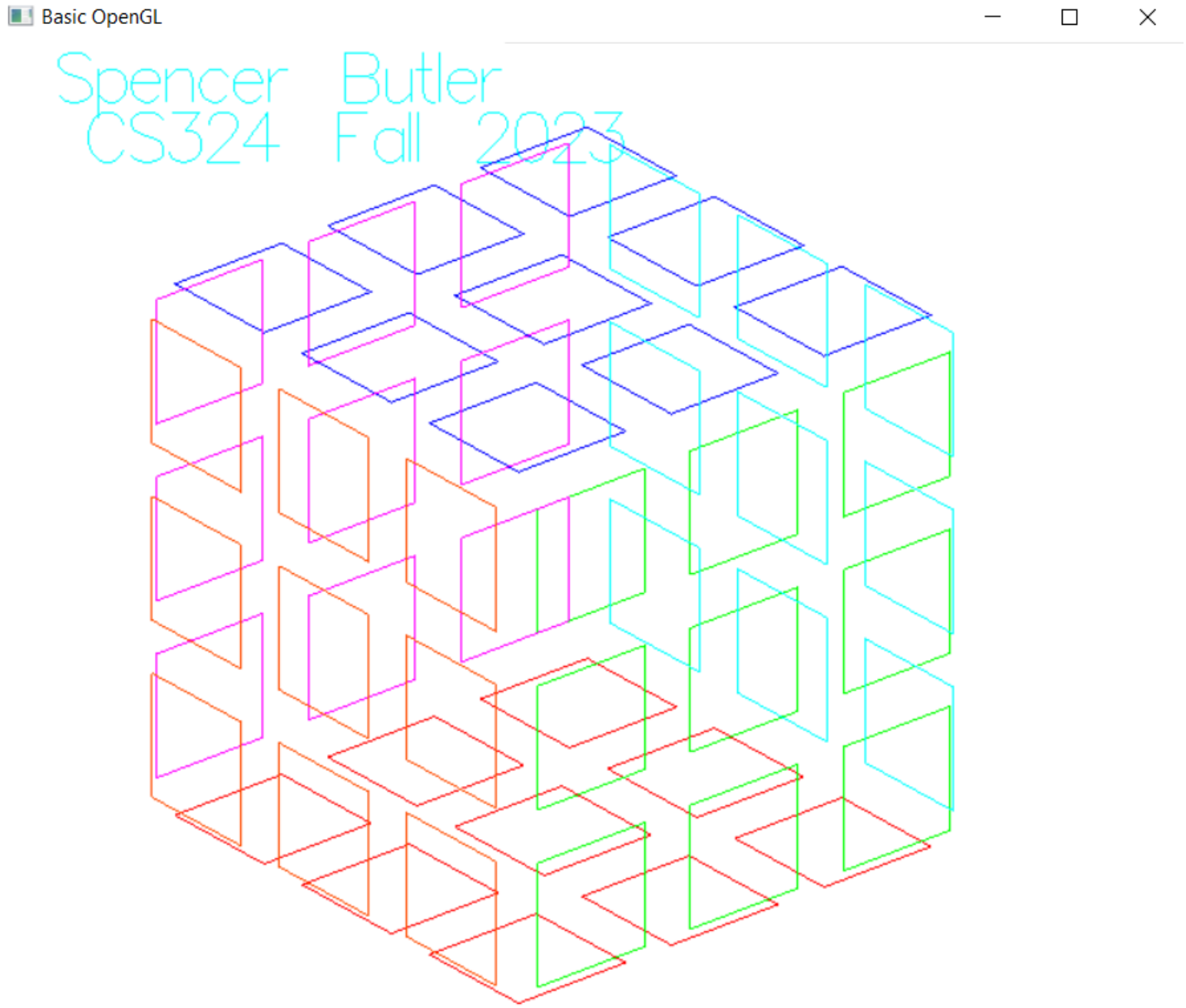
4.4 All 4 ways of plotting the 3D function:



4.5 Rubik's Cube without gaps:



4.6 Rubik's Cube with gaps:



4.7 8000 cubes, with gaps: