Daniel Meyer

CSE 420-01

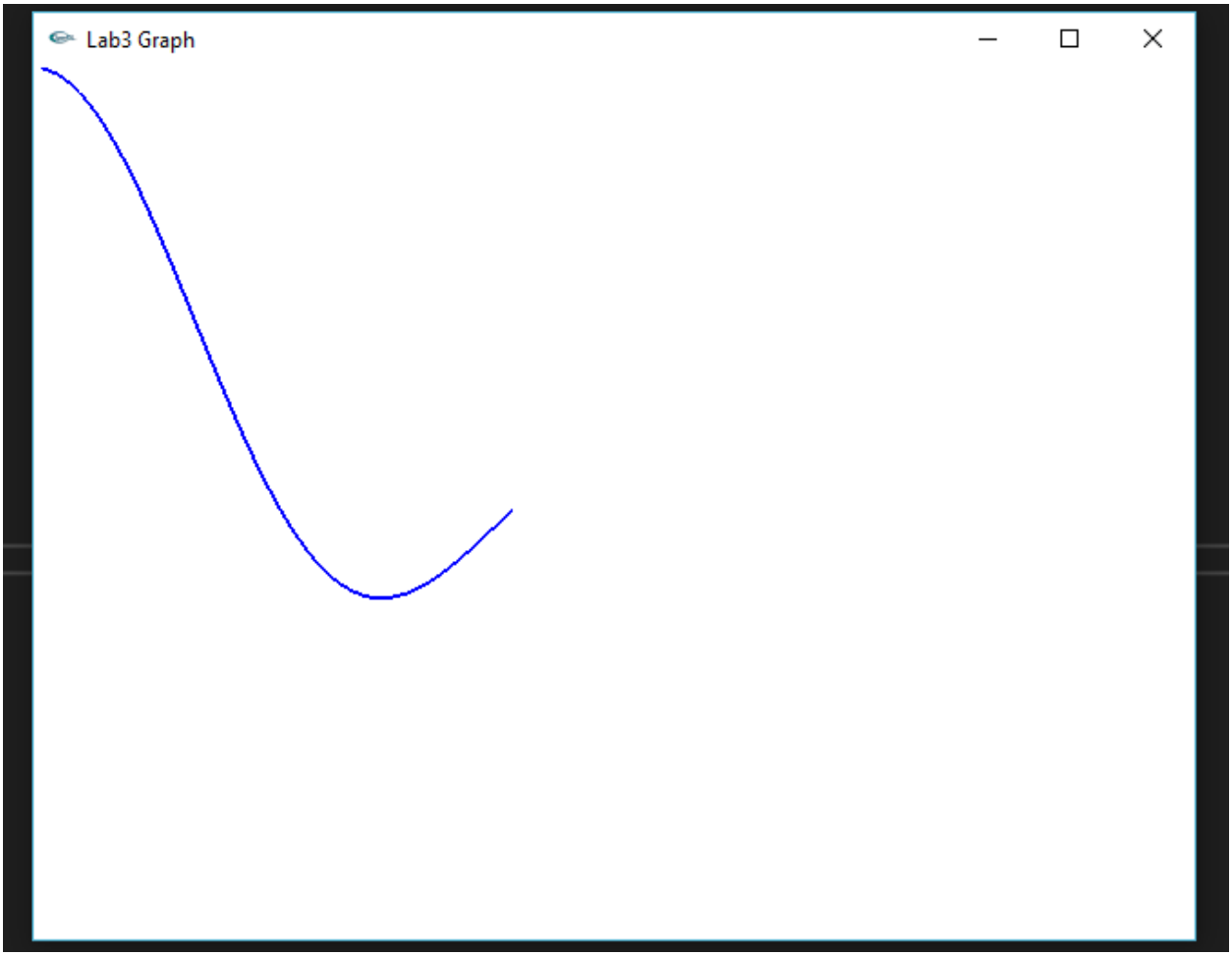
Lab 3

Scaling and Turtle Graphics

Part 1: (Success)

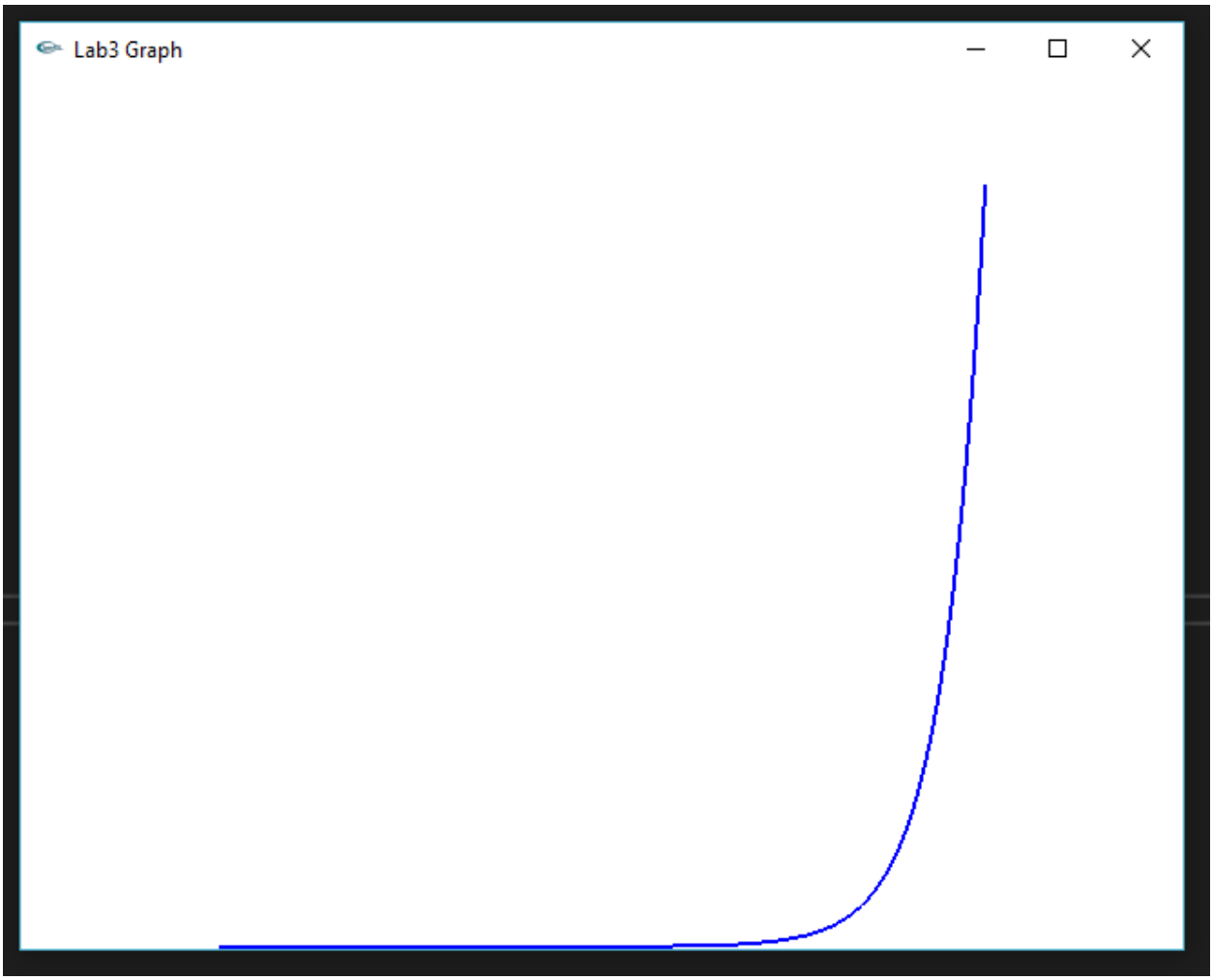
$f(x) = \sin(x) / x, \quad 0 < x \leq 2\pi$

```
void myDisplay(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();

	glBegin(GL_LINE_STRIP);


	for (float x = 0.0; x <= 2 * pi; x += 0.1)
	{
		glVertex2f(x, sin(x) / x);
	}

	glEnd();
	glFlush();
}


//In main

setWindow(0.0, 15.0, -1.0, 1.0);
```


Lab3 Graph

$f(x) = e^{x} / x^{2}, \quad 0 < x \leq 20$

```
void myDisplay(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();

	glBegin(GL_LINE_STRIP);


	for (float x = 0.0; x <= 20.0; x += 0.1)
	{
		glVertex2f(x, (pow(e, x) / pow(x, 2)));
	}


	glEnd();
	glFlush();

}

//In main

setWindow(-5.0, 25.0, -5000.0, 2000000.0);
```



Part 2: (Success)

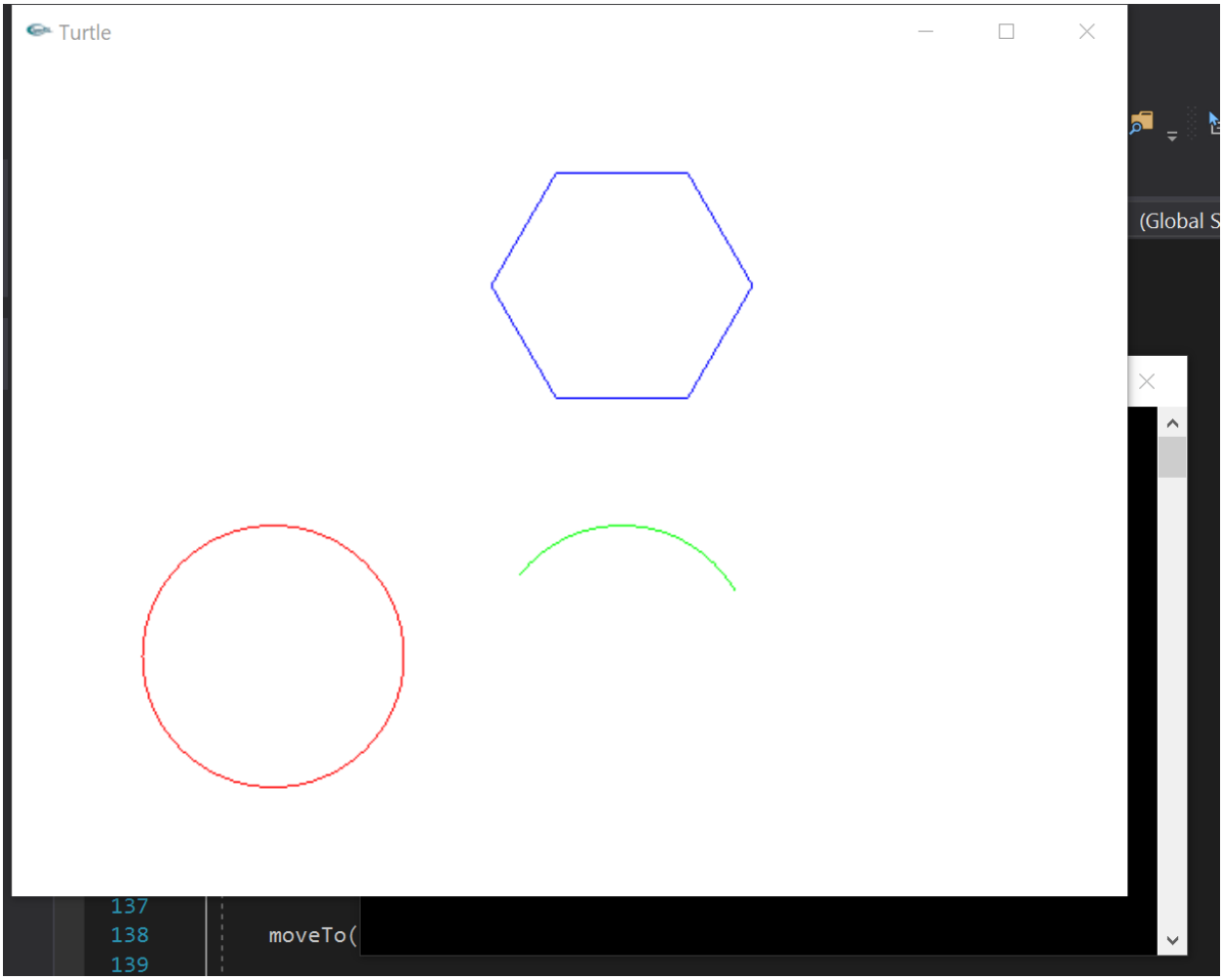
```
#include <Windows.h>
#include <iostream>
#include <math.h>
#include <GL/GL.h>
#include <GL/GLU.h>
#include <GL/GLUT.h>

class GLintPoint
{
public:
	GLint x, y;
};
```

```
class Point2
{
public:
	float x, y;
	void set(float dx, float dy) { x = dx; y = dy; }
	void set(Point2 &p) { x = p.x; y = p.y; }
	Point2(float xx, float yy) { x = xx; y = yy; }
	Point2() { x = y = 0; }
};

Point2 currPos;
Point2 CP;

const float pi = 3.14159265358979;

void moveTo(Point2 p)
{
	CP.set(p);
}

void moveTo(float x, float y)
{
	CP.set(x, y);
}

void lineTo(Point2 p)
{
	glBegin(GL_LINES);

	glVertex2f(CP.x, CP.y);
	glVertex2f(p.x, p.y);

	glEnd();
	glFlush();
	CP.set(p);
}

void lineTo(float x, float y)
{
	glBegin(GL_LINES);

	glVertex2f(CP.x, CP.y);
	glVertex2f(x, y);

	glEnd();
```

```
	glFlush();
	CP.set(x, y);
}

void myInit(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glColor3f(0.0, 0.0, 1.0);
}

void setWindow(float left, float right, float bottom, float top)
{
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D((GLdouble)left, (GLdouble)right, (GLdouble)bottom,
(GLdouble)top);
}

void setViewport(int left, int right, int bottom, int top)
{
	glViewport(left, bottom, right - left, top - bottom);
}

//draw an n-sided regular polygon
void draw_polygon(int N, float cx, float cy, float radius, float
rotAngle)
{
	if (N < 3) return;                    //bad number of sides

	double angle = rotAngle * pi / 180;   //initial angle
	double theta = 2 * pi / N;       //angle increment

	moveTo(radius * cos(angle) + cx, radius * sin(angle) + cy);

	for (int k = 0; k < N; k++) //repeat n times
	{
		angle += theta;
		lineTo(radius * cos(angle) + cx, radius * sin(angle) + cy);
	}
} //draw_polygon

void rosette(int N, float radius)
{
	Point2 *pointlist = new Point2[N];
	GLfloat theta = (2.0f * pi) / N;
```

```
	for (int c = 0; c < N; c++)
	{
		pointlist[c].set(radius * sin(theta * c), radius * cos(theta
* c));
	}

	for (int i = 0; i < N; i++)
	{
		for (int j = 0; j < N; j++)
		{
			moveTo(pointlist[i]);
			lineTo(pointlist[j]);
		}
	}
}

void draw_circle(float cx, float cy, float radius)
{
	glColor3f(1.0, 0.0, 0.0);

	const int numVerts = 100;
	draw_polygon(numVerts, cx, cy, radius, 0);
	glPointSize(3);

	glFlush();
}

void draw_arc(float cx, float cy, float radius, float sAngle, float
sweep)
{
	glColor3f(0.0, 1.0, 0.0);

	const int n = 30;
	float angle = sAngle * pi / 180;
	float theta = sweep * pi / (180 * n);

	moveTo(cx + radius * cos(angle), cy + radius * sin(angle));

	for (int i = 1; i < n; i++)
	{
		lineTo(cx + radius * cos(angle), cy + radius * sin(angle));
		angle += theta;
	}
}
```

```
void render()
{
	glClear(GL_COLOR_BUFFER_BIT);
	setWindow(-10.0, 10.0, -10.0, 10.0);
	setViewport(0, 500, 0, 500);
	//glViewport(10, 10, 640, 480);
	//rosette(5, 0.66f);
	glColor3f(0.0, 0.0, 1.0);
	draw_polygon(6, 4.0, 4.0, 3.0, 0);
	draw_circle(-4.0, -4.5, 3.0);
	draw_arc(4.0, -4.5, 3.0, 30, 120);
	glFlush();
}

void main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glutInitWindowSize(640, 480);
	glutCreateWindow("Turtle");
	glutDisplayFunc(render);

	myInit();
	glutMainLoop();
}
```



Summary:

At the start of this assignment I had encountered a problem using the code provided by the professor using SDL. The version of SDL used to create the code was the older SDL 1.2 which has since been updated to SDL 2.0. In the process of the update 2 key functions were removed and need to be updated within the code. These functions were SDL_Flip and SDL_SetVideoMode, as a result I instead used the polygon turtle drawing method given in the textbook on page 111 for drawing the hexagon, circle and arc. For drawing the polygons, I converted the code for using the Surface 'surf' to use the Point2 struct used in the textbook example and then proceded to convert the draw_polygon function in hook.cpp as well as convert the drawCircle and drawArc functions in canvas.cpp given in the lecture notes from the Canvas cvs to the Point2 struct. For drawing the graphs, I adapted the code given in the textbook on page 93 and changed the formula to the one found for doubly y in the provided plots.cpp to ensure the same results could be produced and upon success I entered the formulas shown above and produced the results seen in the screenshots. The results of these

conversions match the desired results asked for by the Lab and each program runs without error. As a result I believe I have earned the full 20 points for the assignment.