R E P O R T

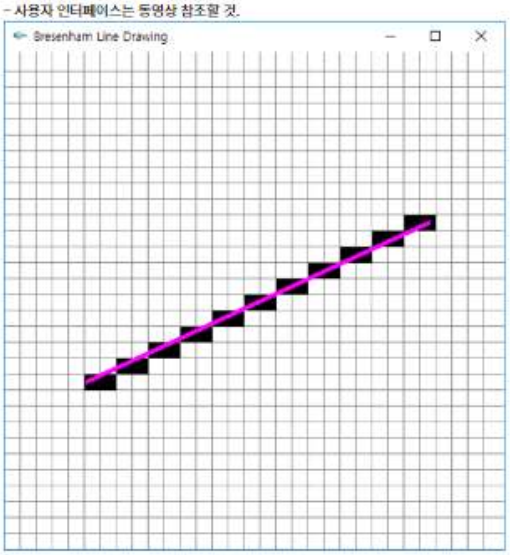
실습과제 10

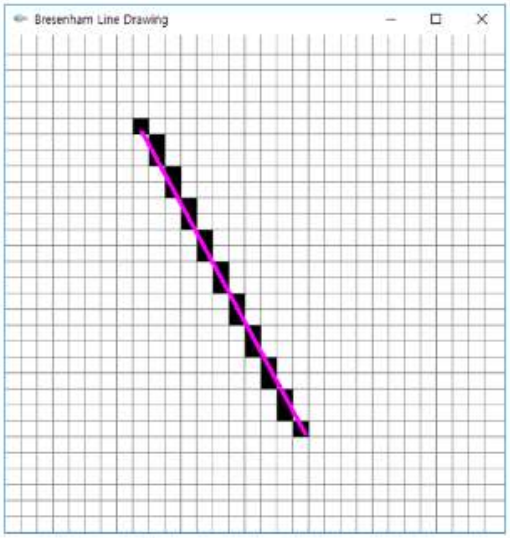
내용

과제 명세

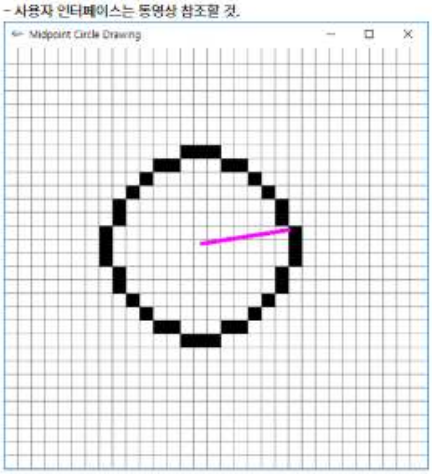
실습과제 10

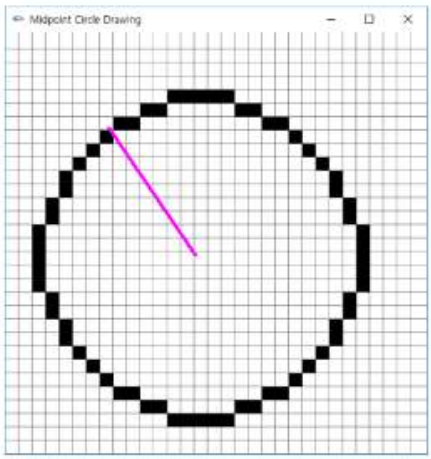
• Bresenham의 선분 그리기 알고리즘을 모든 입력 상황에 대해 처리할 수 있도록 확장하라.
- 다음은 실행 결과의 예이다.
- 사용자 인터페이스는 동영상 참조할 것.





• Midpoint Circle 알고리즘을 모든 입력 상황에 대해 처리할 수 있도록 확장하라. 특히 drawCirclePoint(x,y) 함수에서는 8 화소에 대해 그려야 할 것이다.
- 다음은 실행 결과의 예이다.
- 사용자 인터페이스는 동영상 참조할 것.





기본 구현하기

- 예시 코드

```
#include <iostream>
#include <gl/glut.h>

inline int abs(int k) { return (k >= 0) ? k : -k; }      // 절대값 구하기
inline int max(int i, int j) { return (i > j) ? i : j; }     // 최대값 구하기
inline int ROUND(double v) { return (int)(v + 0.5); }            // 반올림값 구하기
inline double dist(double dx, double dy) { return sqrt(dx * dx + dy * dy); }// 반지름값( dx²+ dy의 1/2승)구하기
/*inline void swap(int* a, int* b) {                    // swap 값 구하기 => std::swap으로 대체
            int* buf = a;
            a = b;
            b = buf;
}*/

static int winW = 500, winH = 500;                  // 너비 높이 => 각각 500
static int nGrid = 30;                                                  // 가로 세로 그리드 => 30 * 30
static double pixelW, pixelH;                       // pixel 단위 너비와 높이를 나타내는 변수
static int px, py, qx, qy;                                  // 점 p와 q에 대한 좌표 (px, py), (qx, qy)
static int mode = 0;                                        // 키보드 숫자 1, 2, 3, 4에 따라 모드 전환

// 그리드 라인을 그리는 함수
void draw2DGrid()
{
            glLineWidth(1);                                                     // grid 크기 1을 갖는 라인 그리기
            glBegin(GL_LINES);
            for (int i = 0; i <= nGrid; i++) {          // 행라인과 열라인을 nGrid 크기에 의존하여 그린다.
                        glVertex2d(i * pixelW, 0);
                        glVertex2d(i * pixelW, winH);
                        glVertex2d(0, i * pixelH);
                        glVertex2d(winW, i * pixelH);
            }
            glEnd();
}

// 픽셀 단위(행렬 기준 1*1칸)로 그리는 함수
void drawPixel(GLint x, GLint y) {
            int xi = x * pixelW;
            int yi = y * pixelH;
            glRectd(xi, yi, xi + pixelW, yi + pixelH);
}

// mode에 의존하는 함수 1,2,3,4 //
void lineBasic(GLint x1, GLint y1, GLint x2, GLint y2);// mode 1
void lineDDA(GLint x1, GLint y1, GLint x2, GLint y2); // mode 2
void lineBreaseham(GLint x1, GLint y1, GLint x2, GLint y2);       // mode 3
void circleMidPoint(GLint radius);              // mode 4

// 기본 출력 함수
void display()
```

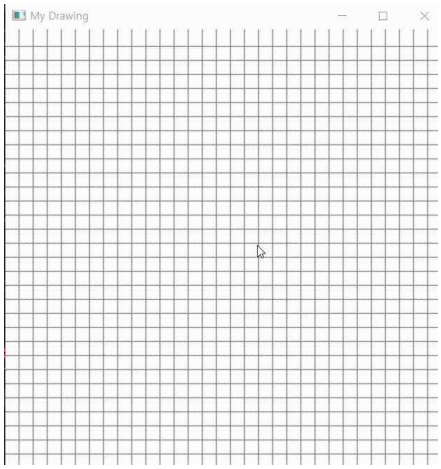
```
{
	glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.5, 0.5, 0.5);
	draw2DGrid(); // 화소 그리드

	glColor3f(0, 0, 0);	// Breaseham 선분
	switch (mode) {
	case 1: lineBasic(px / pixelW, py / pixelH, qx / pixelW, qy / pixelH);
		break;
	case 2:	lineDDA(px / pixelW, py / pixelH, qx / pixelW, qy / pixelH);
		break;
	case 3: lineBreaseham(px / pixelW, py / pixelH, qx / pixelW, qy / pixelH);
		break;
	case 4: int radius = ROUND(dist(px / pixelW - qx / pixelW, py / pixelH - qy / pixelH));
		circleMidPoint(radius);
		break;
	}

	glColor3f(1.0, 0.0, 1.0);		//원래의 선분
	glLineWidth(5);
	glBegin(GL_LINES);
		glVertex2f(px, py);
		glVertex2f(qx, qy);
		glEnd();
		glFlush();
}

// 마우스 클릭(시작-끝 점 좌표) 함수
void mouseClick(GLint button, GLint state, GLint x, GLint y) {
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
		px = x;
		py = winH - y;
	}
}

// 마우스 모션()함수(선 내부의 점 좌포) 함수
void mouseMotion(GLint x, GLint y) {
	qx = x;
	qy = (winH - y);
	glutPostRedisplay();
}

//================================================================================
// 픽셀을 좌표에 맞게 그리는 함수
void displayPixel(int x, int y){
	drawPixel( x + (qx - px) / winW, y + (qy - py) / winH);
}

//================================================================================
// mode 1. lineBasic 함수 => 선분의 기울기를 이용해서 그리는 함수
```

```
// 직선의 방정식을 이용(y = m(x - x1)+y1=> – 1보다 크면 y 좌표를 증가, 1보다 작으면 x 좌표를 증가
// 간단하지만 부동 소수점 곱셈이 반복되어 속도가 느린 단점이 있다.
// std::swap과 steep 변수를 이용해서 실행 범위를 전범위로 확장시켰다.
void lineBasic(GLint x1, GLint y1, GLint x2, GLint y2) {
	const bool steep = (abs(y2 - y1) > abs(x2 - x1));
	if (steep)
	{
		std::swap(x1, y1);
		std::swap(x2, y2);
	}

	if (x1 > x2)
	{
		std::swap(x1, x2);
		std::swap(y1, y2);
	}
	double m = (double)(y2 - y1) / (x2 - x1);
	for (int x = x1; x <= x2; x++) {
		double y = m * (x - x1) + y1;
		if (steep)
		{
			displayPixel(y, x);
		}
		else
		{
			displayPixel(x, y);
		}
	}
}
//================================================================================
// mode2. DDA 알고리즘을 이용한 선분 그리기=> 증가분을 이용해서 선분을 구하는 방식
// Digital Differential Analyzer를 의미한다.
// 증가분 m을 계산하고 증가분을 반복적으로 더하는 식으로 구현한다.
// 곱셈을 반복적으로 사용하지 않는다는 점에서 mode1에 비해 장점을 가진다.
// 하지만 mode1처럼 부동 소수점 연산 오버헤드가 존재하고, 이상적인 m과 부동소수점으로 표현하는 m의 차이
// 즉, 긴 선분을 그리는 경우 에러가 점점 쌓이게 된다는 문제가 있다.
void lineDDA(GLint x1, GLint y1, GLint x2, GLint y2) {
	int dX = abs(x2 - x1);
	int dY = abs(y2 - y1);
	int steps = max(dX, dY);			// (dX > dY) ? dX : dY;
	double incX = (double)(x2 - x1) / steps;
	double incY = (double)(y2 - y1) / steps;
	double x = x1;
	double y = y1;

	for (int i = 0; i <= steps; i++, x += incX, y += incY)
		displayPixel(ROUND(x), ROUND(y));
}
. . .
void keyboard(unsigned char key, int x, int y) {
	if (key >= '0' && key <= '9')
```

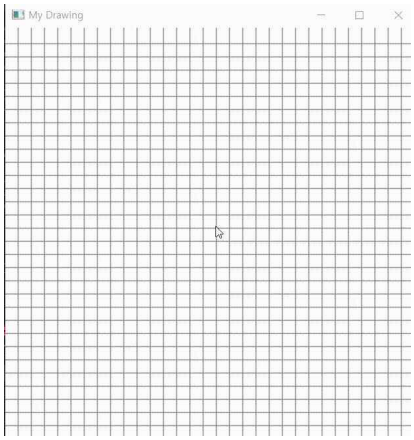
```
                mode = (int)(key - '0');
        else if (key == 'q') exit(0);
        glutPostRedisplay();
}

int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitWindowSize(winW, winH);
        glutCreateWindow("My Drawing");
        pixelW = winW / nGrid;
        pixelH = winH / nGrid;
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, winW, 0, winH, -1, 1);

        glutDisplayFunc(display);
        glutMouseFunc(mouseClick);
        glutMotionFunc(mouseMotion);
        glutKeyboardFunc(keyboard);

        glutMainLoop();
        return 0;
}
```

- 실행 결과

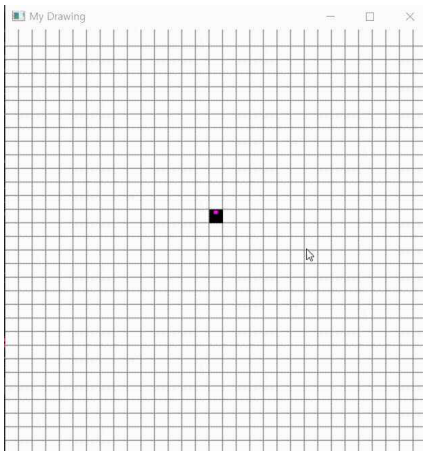
기본 출력 => 기본적으로 생성되는 보라색 라인만 출력하는 모습입니다.



mode 1. Basic Line 출력 => 기울기를 이용해서 선을 출력하는 함수를 모든 사분면에 대해 정규화한 모습입니다.



mode 2. DDA Line 출력=> 부동소수점과 절대값 연산을 이용해 선을 출력하는 모습입니다.



Bresenham의 선분 그리기 알고리즘 확장하기

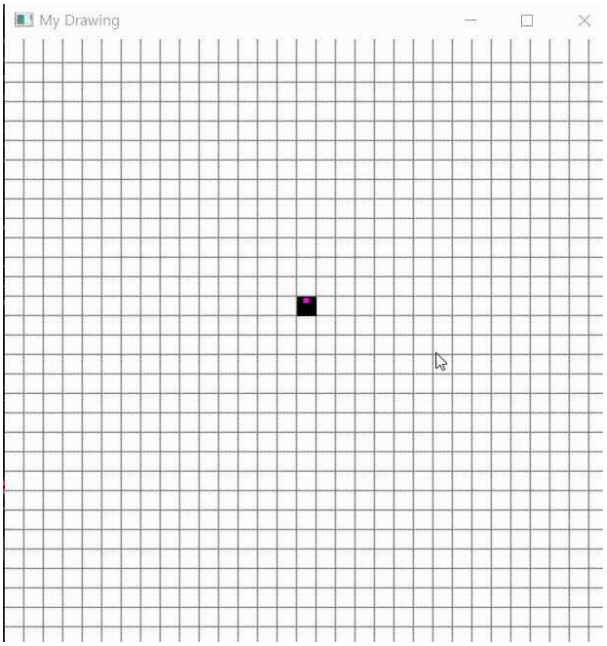
- 예시 코드

```
// mode 3. Breaseham 알고리즘을 이용한 선분 그리기
// 본래 선분의 기울기 0~1 사이에서 구현됨 (0~45도)
//          P(i+1)을 구하기 위해 P(i)가 0보다 작을 때 P(i) + 2dY(dF_1)
//          그 외에 경우 P(i) = 2dY - dX(dF_2 )
//          P(i) = 2dY - dX를 만족한다.

// 모든 경우를 만족시키기 위해
// ① 증가분이 큰 것을 밑변으로 바꾸고
// ② 양수의 범위를 적용시키기 위해 x2 - x1 > 0, y2-y1 > 0인 조건을 추가해준다.
// ③ 기울기를 기준으로 1보다 크면 y 좌표를 증가 1보다 작으면 x 좌표를 증가해준다.
void lineBreaseham(GLint x1, GLint y1, GLint x2, GLint y2) {
          // Bresenham's line algorithm
          // steep과 swap을 이용한 Breaseham 알고리즘 확장
          const bool steep = (abs(y2 - y1) > abs(x2 - x1));
          if (steep)
          {
                    std::swap(x1, y1);
                    std::swap(x2, y2);
          }

          if (x1 > x2)
          {
                    std::swap(x1, x2);
                    std::swap(y1, y2);
          }

          const float dx = x2 - x1;
          const float dy = abs(y2 - y1);

          float error = dx / 2.0f;
          const int ystep = (y1 < y2) ? 1 : -1;
          int y = (int)y1;

          const int maxX = (int)x2;

          for (int x = (int)x1; x <= maxX; x++)
          {
                    if (steep)
                    {
                              displayPixel(y, x);
                    }
                    else
                    {
                              displayPixel(x, y);
                    }

                    error -= dy;
                    if (error < 0)
```

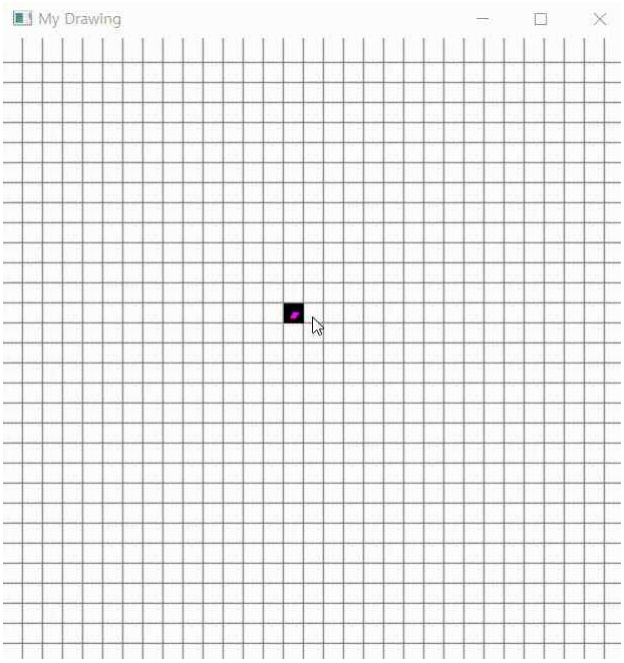
```
                {
                        y += ystep;
                        error += dx;
                }
        }
        /*
        일반화 이전 코드
        int dX = x2 - x1;
        int dY = y2 - y1;
        int dF_1 = 2 * dY;                          // 판별식 1
        int dF_2 = 2 * (dY - dX);       // 판별식 2
        int F = 2 * dY - dX;            // 초기값
        int y = y1;

        for (int x = x1 + 1; x < x2; x++) {
                if (F < 0) F += dF_1;
                else {
                        F += dF_2;
                        y++;
                }
                displayPixel(x, y);
        }
        displayPixel(x1, y1);
        displayPixel(x2, y2);
        */
}
```

- 실행 결과

정규화한 출력 결과 => 0~360도에 해당하는 모든 경우에 정상적으로 출력됨을 보여주고 있다.



정규화 이전 출력 결과 => 0~45도 사이일 때만 정상적으로 출력됨을 보여주고 있다.

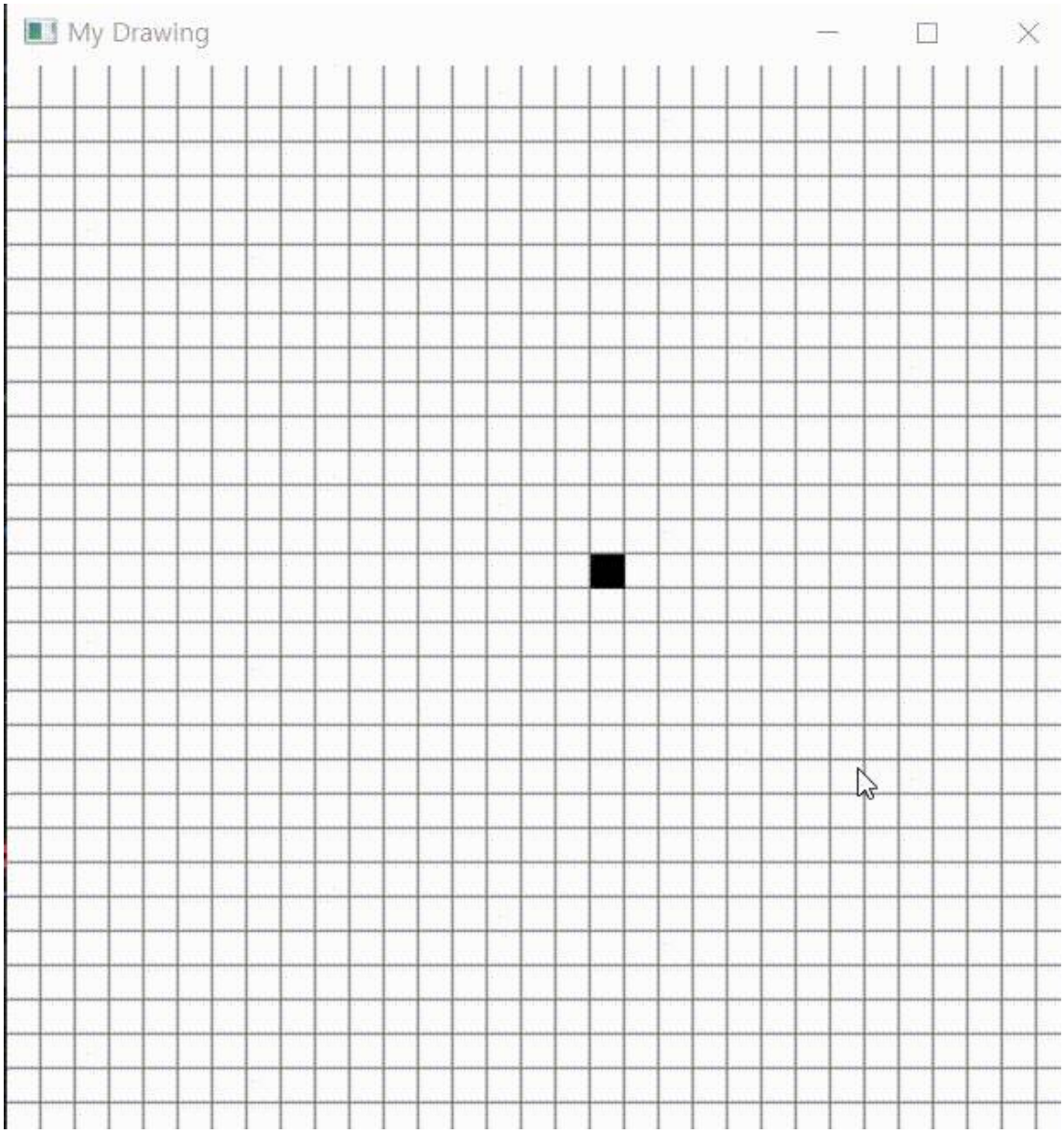


Midpoint Circle 알고리즘 확장하기

- 예시 코드

```
// mode 4 Midpoint Circle 알고리즘을 이용한 원그리기
// d(i+1)을 구하기 위해
// d(i) < 0 인 경우 d(i) + (2x(i) + 3)
// 그 외의 경우 d(i) + (2x(i) + 2y(i) + 5)
// d(0) = 1 - R (단, R은 반지름)
// 을 만족한다.
// 부동 소수점 연산이 필요없고, 2의 배수는 shift연산으로 처리되며
// 각각의 원의 1/8(0~45도)을 그리기에 대칭을 이용해 다른 원의 좌표도 구해준다.
// px, py를 중심으로 하고 주변의 x, y를 그려주는 형태로 구현된다.
void drawCirclePoint(int x, int y) {
        // 8좌표에 대한 그리기
        displayPixel(x + px / pixelW, y + py / pixelH);
        displayPixel(x + px / pixelW, -y + py / pixelH);
        displayPixel(-x + px / pixelW, y + py / pixelH);
        displayPixel(-x + px / pixelW, -y + py / pixelH);
        displayPixel(y + px / pixelW, x + py / pixelH);
        displayPixel(y + px / pixelW, -x + py / pixelH);
        displayPixel(-y + px / pixelW, x + py / pixelH);
        displayPixel(-y + px / pixelW, -x + py / pixelH);
}

void circleMidPoint(GLint radius) {          // 중심이 (0,0)이고 반지름이 R인 원
        int x = 0;
        int y = radius;
        int d = 1 - radius;
        drawCirclePoint(x, y);
        for (; y > x; x++) {
                if (d < 0)
                        d = d + 2 * x + 3;
                else {
                        d = d + 2 * (x - y) + 5;
                        y = y - 1;
                }
                drawCirclePoint(x, y);
        }
}
```

- 실행 결과 => 8분면에 대해 정상적으로 원을 출력하는 모습입니다.



-

고찰 및 느낀점

이번 과제를 진행함에 있어서 선분 그리기 알고리즘 3개와 원을 그리는 알고리즘의 구현을 실습해보았습니다. 선분 그리는 알고리즘을 각각 살펴보았을 때 전체적으로 똑같이 그려지는 것처럼 생겼었지만 자세히 보았을 때 픽셀이 조금씩 엇나가는 정도가 다른 모습을 보면서 확실히 부동소수점 연산이 적고 구획별로 나눈 연산이 조금 더 세밀한 작업이 가능함을 눈으로 확인할 수 있었습니다. Breaenham과 Midpoint Circle의 유도 과정을 제대로는 이해하지 못한 채로 구현을 시작해서 이해하는데 어려움이 있었지만 강의 동영상을 보면서 조금씩 구현의 틀을 잡아갈 수 있었던 것 같습니다. 감사합니다.