R E P O R T

컴퓨터그래픽스및실습

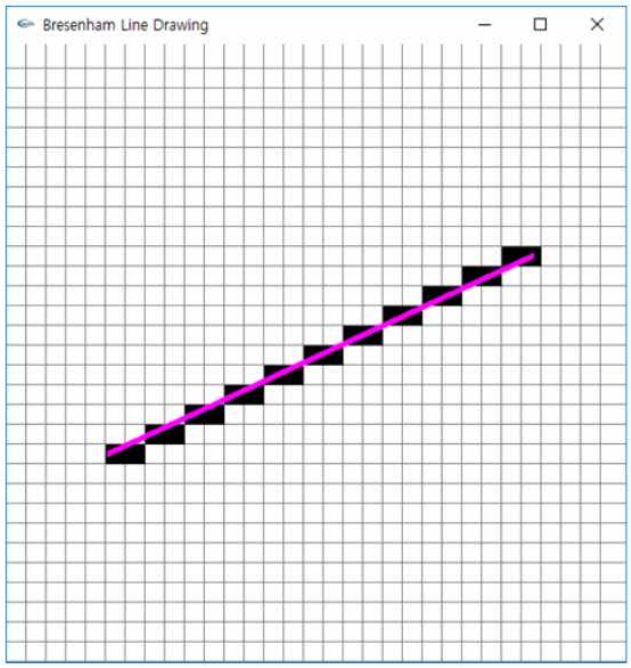
실습과제 10

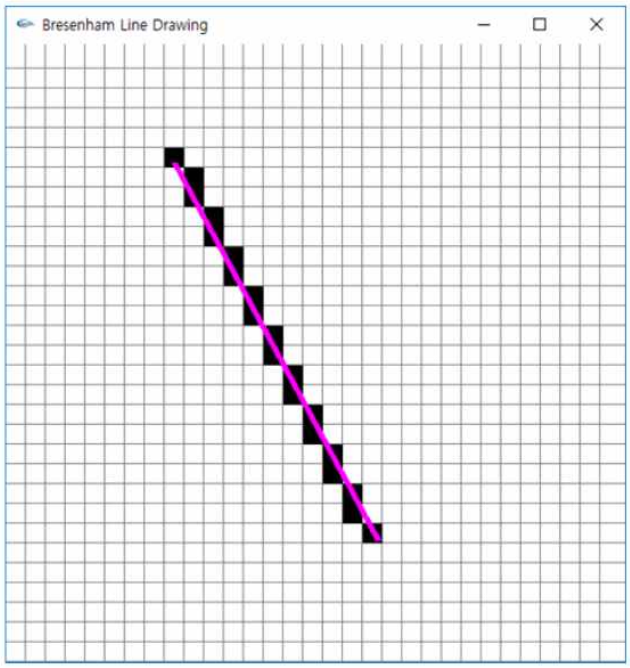
과목명	컴퓨터그래픽스및실습
분반	01
교수	최 영 규
학번	2020136129
이름	최 수 연
제출일	2022년 11월 17일 목요일

[문제 분석 및 해결 방법]

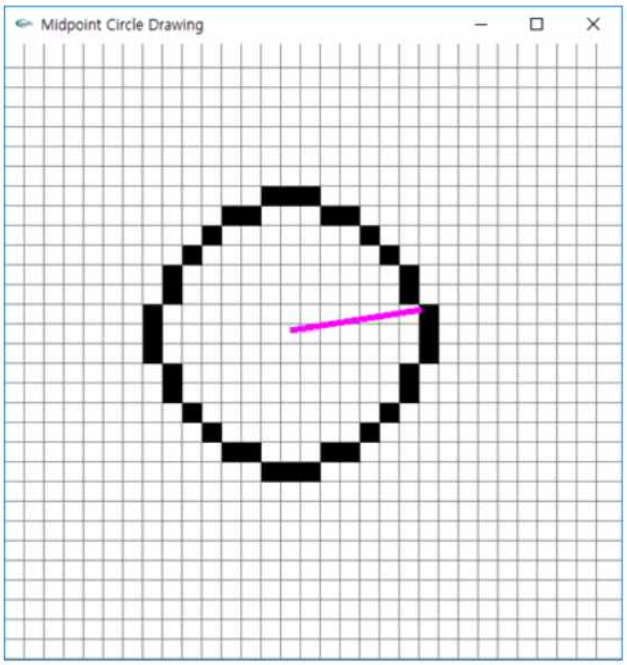
[문제 분석]

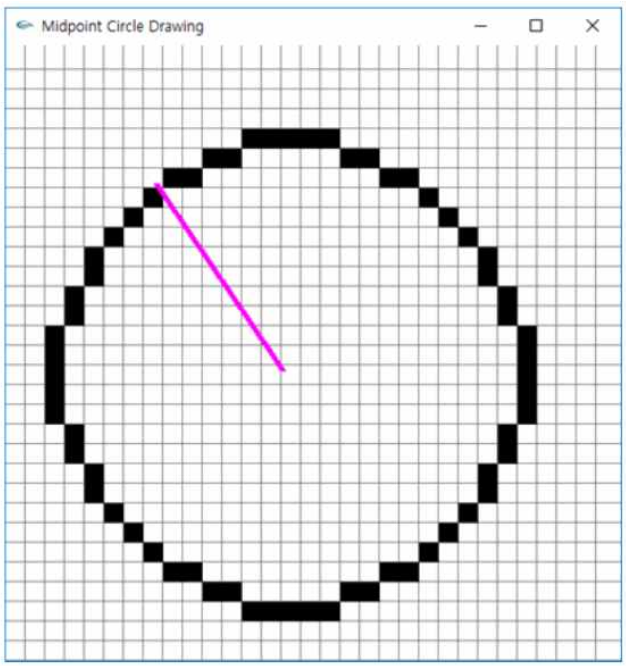
- Bresenham의 선분 그리기 알고리즘을 모든 입력 상황에 대해 처리할 수 있도록 확장하라.
  - 다음은 실행 결과의 예이다.
  - 사용자 인터페이스는 동영상 참조할 것.





- Midpoint Circle 알고리즘을 모든 입력 상황에 대해 처리할 수 있돌고 확장하라. 특히 drawCirclePoint(x,y) 함수에서는 8 화소에 대해 그려야 할 것이다.





[해결 방법]

- **먼저 Basic 알고리즘의 경우,** 직선의 방정식을 사용하여 기울기의 절댓값이 만약 1보다 크면 y 좌표를 증가시키며 x좌표를 구하고, 1보다 작으면 x좌표를 증가시키며 y좌표를 구한다.

- **DDA 알고리즘의 경우,** 증가분 m을 계산하여 반복적으로 더하는 것인데, 먼저 선분의 각 끝점에 대하여 dx, dy를 구한 뒤, 증가량이 더 적은 것에 증가분 m을 반복적으로 더하도록 하고, 상대적으로 증가량이 더 많은 것에 1을 반복적으로 더하여 이때 구한 x, y를 부동 소수점 연산하여 픽셀을 칠한다.

- **Bresenham 알고리즘의 경우,** 기울기가 0~1 사이라고 할 때, 선분 위 특정 점을 기준으로, 위아래의 두 픽셀 중 더 가까운 곳을 색칠한다. 만약 선분 위 특정 점을 기준으로 해당 점과 위아래 픽셀의 중앙에 있는 점과의 각 거리를 d1, d2라고 할 때 d1과 d2의 차이가 d라고 하면, d에 dx를 곱한 값이 p이다. 이때 p가 0보다 작으면 d1이 d2보다 작으므로 다음 화소는 d1를 계산할 때 사용한 y값을 택한다. 만약 p가 0보다 크면 d2가 d1보다 작으므로 다음 화소는 d2를 계산할 때 사용한 y값을 택한다. 이는 기울기 0~1 사이만 계산한 것이므로, 나머지 기울기의 모든 경우를 고려하여 선분이 그려지도록 구현한다.

- **Midpoint Circle 알고리즘의 경우,** 픽셀 사이의 한 점 Midpoint를 중심으로 해당 점을 원의 공식에 대입하였을 때 0보다 작을 경우, Midpoint는 원의 내부에 있는 것이므로 Midpoint 기준으로 바깥쪽에 원이 지나도록 픽셀을 칠한다. 그렇지 않고 0보다 클 경우, 원의 외부에 있는 것이므로 Midpoint 기준으로 안쪽에 원이 지나도록 픽셀을 칠한다.

[주요 설명 코드]

draw2DGrid() / drawPixel() / display()

```
void draw2DGrid() { // 전체 화면 그리드 그리기
        glLineWidth(1); // line 두께: 1
        glBegin(GL_LINES);
        for (int i = 0; i <= nGrid; i++) {
                glVertex2d(i * pixelW, 0);
                glVertex2d(i * pixelW, winH);
                glVertex2d(0, i * pixelH);
                glVertex2d(winW, i * pixelH);
        }
        glEnd();
```

} // 선의 두께를 1로 하여 GL_LINES를 통해 전체 화면에 대한 그리드를 그린다. 이때 윈도우 크기는 각각 winW=500, winH=500으로 제한하였다.

```
void drawPixel(GLint x, GLint y) { // 그리드 안에 픽셀 하나를 색칠함
        int xi = x * pixelW;
        int yi = y * pixelH;
        glRectd(xi, yi, xi + pixelW, yi + pixelH);
```

} // 각 알고리즘에 대하여 선분에 대한 픽셀을 그릴 때 불러오는 함수로, 각 선분의 위치에 대한 픽셀을 그리드에 맞게 색칠한다.

```
void display() {
        ........
        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: lineBresenham(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); // 움직이는 분홍색 선 출력
        glBegin(GL_LINES);
        glVertex2f(px, py);
        glVertex2f(qx, qy);
        glEnd();
        glFlush();
```

} // 각 알고리즘에 맞는 mode 번호를 클릭했을 때, 각 번호에 해당하는 알고리즘 함수를 호출하여 선분이 변할 때마다 계속 호출하여 함수 내부를 실행하고 이에 맞게 화면에 다시 그려지도록 한다. switch 문 아래는 마우스가 클릭되어 움직일 때마다 분홍색 선이 마우스에 따라 그려지도록 하는 코드이다.

Basic line

```
inline void swap(int* a, int* b) { int tmp = *a; *a = *b; *b = tmp; }
void lineBasic(GLint x1, GLint y1, GLint x2, GLint y2) {
        double m = (double)(y2 - y1) / (x2 - x1);
        if (-1 <= m && m <= 1) {
                if ((x2 - x1) <= 0) { swap(&x1, &x2); swap(&y1, &y2); }
                for (int x = x1; x <= x2; x++) {
                        double y = m * (x - x1) + y1;
                        drawPixel(x, ROUND(y));
                }
        }
        else {
                if ((y2 - y1) <= 0) { swap(&x1, &x2); swap(&y1, &y2); }
                for (int y = y1; y <= y2; y++) {
                        double x = (y - y1) / m + x1;
                        drawPixel(ROUND(x), y);
                }
        }
```

} // 두 점에 대한 기울기를 구하고, 기울기가 1보다 작고 -1보다 클 땐 x좌표를 증가시키며 y좌표를 구하여 픽셀을 색칠하고, 기울기가 1보다 크거나 -1보다 작을 땐 y좌표를 증가시키며 각각의 x좌표를 구하여 픽셀을 칠한다. 이때 (x2-x1)나 (y2-y1)을 했을 때 음수가 나올 경우, 두 좌표를 서로 swap 하여 계산한다.

DDA line

```
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)
                drawPixel(ROUND(x), ROUND(y));
```

} // 먼저 dx, dy를 구하여 이를 절댓값으로 바꾼 뒤, 두 값 중 더 큰 값을 steps으로 하고, 각 증가분인 incX, incY를 구하는데 각 dx, dy에서 steps을 나누어 증가분을 구한다. 그러면 steps에서 선택된 값의 증가분은 1이 되고, 다른 값은 1보다 작은 증가분 m이 된다. 이를 for 문을 통해 반복하여 선분이 지나는 모든 픽셀을 색칠한다.

Bresenham line

```
void lineBresenham(GLint x1, GLint y1, GLint x2, GLint y2) {
	int dX = abs(x2 - x1), dY = abs(y2 - y1);
	int p, const1, const2;
	int x, y;
	if (dX >= dY) {
		p = 2 * dY - dX;
		const1 = 2 * dY;
		const2 = 2 * (dY - dX);
		if (x1 < x2) { x = x1; y = y1; }
		else { x = x2; y = y2; x2 = x1; y2 = y1; }
		drawPixel(x, y);
		while (x < x2) {
			if (p < 0) p += const1;
			else {
				if (y < y2) { p += const2; y++; }
				else { p += const2; y--; }
			}
			x++;
			drawPixel(x, y);
		}
	} else {
		p = 2 * dX - dY;
		const1 = 2 * dX;
		const2 = 2 * (dX - dY);
		if (y1 < y2) { x = x1; y = y1; }
		else { x = x2; y = y2; x2 = x1; y2 = y1; }
		drawPixel(x, y);
		while (y < y2) {
			if (p < 0) p += const1;
			else {
				if (x > x2) { p += const2; x--; }
				else { p += const2; x++; }
			}
			y++;
			drawPixel(x, y);
		}
	}
} // 먼저 dx, dy의 절댓값을 모두 구하고, p 값과 p 값에 더해질 const1, const2를 정의한다. 그리고 dx의 절댓값과 dy의 절댓값을 비교하여 만약 dx가 크면 기울기가 절댓값 1보다 작으므로 첫 번째 if문을, dx가 더 작으면 기울기가 절댓값 1보다 더 크므로 두 번째 else문을 실행한다. 이때 각각 x1과 x2, y1과 y2를 비교하여 더 작은 값의 점을 각각 x, y에 넣고 큰 값을 x2, y2에 넣는다. 다음 반복문을 통해 p가 0보다 작으면 const1을 0보다 크면 const2가 더해지도록 한다. 이때도 각 x, y가 x2, y2보다 큰지 작은지에 따라 x, y를 증가할지 감소할지 결정한다.
```

Midpoint Circle

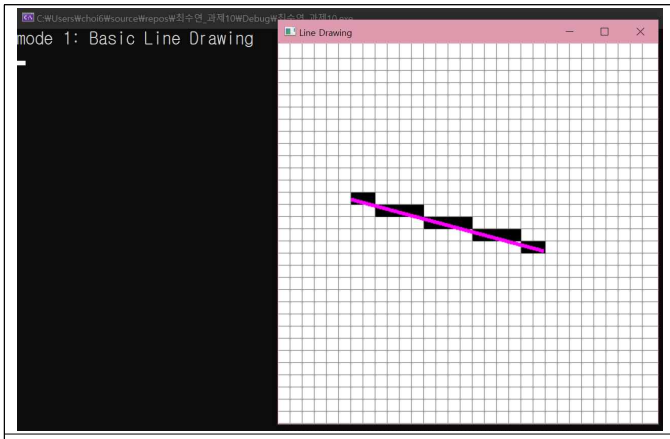
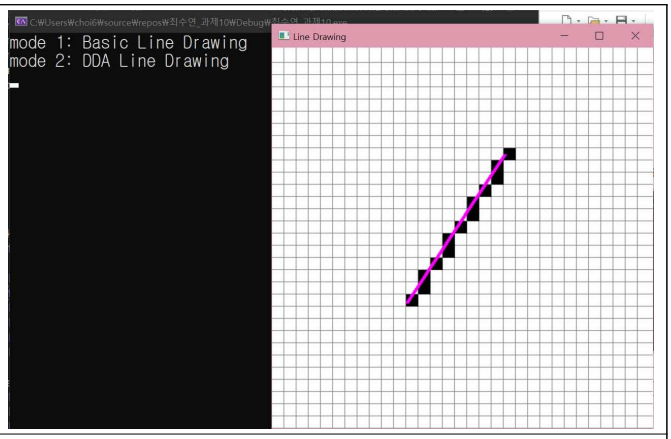
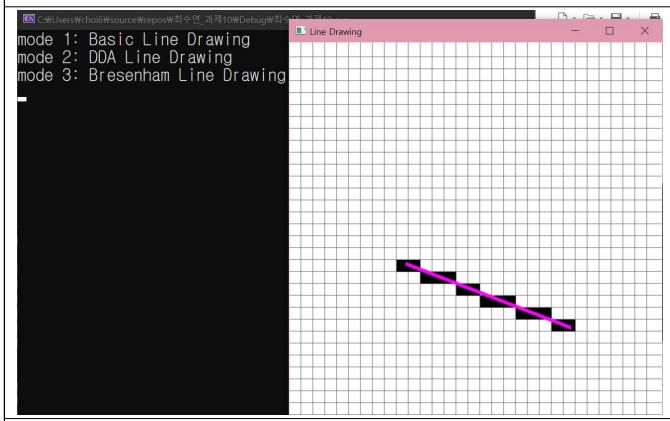
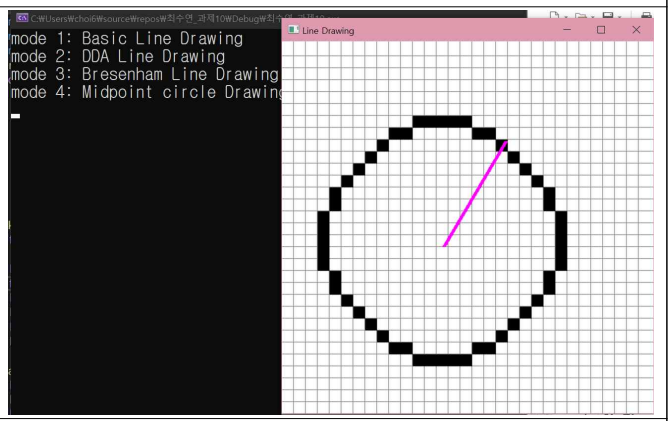
```
void circleMidPoint(GLint radius) { // 중심이 (0, 0)이고 반지름이 R인 원
        int x = 0;
        int y = radius;
        int d = 1 - radius;
        drawCirclePoint(x, y);
        for (; y > x; x++) {
                if (d < 0)
                        d = d + 2 * x + 3;
                else {
          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

} // 위 display() 함수에서 계산된 반지름을 받아 y에 넣고, x는 0, d는 1에서 반지름을 뺀 값을 저장한다. for문을 통해 x가 y보다 작을 때까지 x를 증가시키며 반복문을 수행한다. 이때 d가 0보다 작을 경우와 d보다 크거나 같을 경우를 나누어 d 값을 갱신해준다. 만약 d가 0보다 작을 경우, Midpoint가 원의 내부에 있으므로 Midpoint Circle 알고리즘 정리에 따라 d에 (2x+3)을 더한다. 그렇지 않을 경우, Midpoint가 원의 외부에 있으므로 선을 Midpoint보다 아래에 그려야 하므로 y값을 1 빼주고, d에 (2(x-y)+5)를 더해준다.

```
void drawCirclePoint(int x, int y) {
        drawPixel(x + px / pixelW, y + py / pixelH);
        drawPixel(x + px / pixelW, -y + py / pixelH);
        drawPixel(-x + px / pixelW, y + py / pixelH);
        drawPixel(-x + px / pixelW, -y + py / pixelH);
        drawPixel(y + px / pixelW, x + py / pixelH);
        drawPixel(y + px / pixelW, -x + py / pixelH);
        drawPixel(-y + px / pixelW, x + py / pixelH);
        drawPixel(-y + px / pixelW, -x + py / pixelH);
```

} // circleMidPoint() 함수는 원의 1/8만 처리한 값이므로 해당 함수에서 계산된 x, y 값으로 대칭을 사용해 원이 되도록 8방향을 모두 그리도록 한다. 이때 px, py 점을 중심으로 원이 그려져야 하고, 그리드의 픽셀에 맞게 원이 그려져야 하므로 각각에 (px / pixelW)와 (py / pixelH)를 더한다.

[실행 결과]

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Basic</td><td>DDA</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Bresenham</td><td>Midpoint Circle</td></tr>
</table>

[고찰 및 느낀점]

교수님께서 강의 영상을 올려주셔서 좀 더 수월하게 할 수 있었다. 처음부터 전부 구현하는 과제였으면 생각보다 더 어려웠을 것 같다. 근데 교수님이 알려주신 코드 중 DDA를 제외하고는 나머지 세 알고리즘은 기본 상황만 처리되어 있고 기울기에 대한 선분을 그리는 예외 처리가 되어있지 않아서, 그 부분을 해결하기 위해 강의자료 9장을 다시 복습하며 각 알고리즘에 대해 하나하나 분석해보았다. 덕분에 네 가지 알고리즘에 대해 자세히 알게 되었고 이번 기말고사를 위한 시험공부 또한 더 쉽게 할 수 있을 것 같아서 좋았다.