목차

1. 문제에 대한 분석 및 해결 방법

실습 과제 10: Bresenham과 Midpoint Circle 알고리즘 확장하여 구현하기

이번 과제의 핵심은 일부 기울기 구간에서 작동하는 Bresenham 알고리즘과 원의 일부만을 그리는 Midpoint Circle 알고리즘을 알맞은 조건과 케이스 처리를 통해 전체 입력 상황에 대해서 작동하도록 처리하는 것이다. 따라서 Bresenham 알고리즘의 경우 dY, dX의 절댓값을 이용하여 두개의 분기 함수로 서로 다른 기울기 구간에 대해서 처리하는 부분을 구현하였고, Midpoint Circle 알고리즘은 기존 알고리즘 구현에 8방향 반사를 구현하여 온전한 원이 그려질 수 있도록 구현하였다.

2. 자신이 구현한 주요 코드

실습 과제 10: Bresenham과 Midpoint Circle 알고리즘 확장하여 구현하기

```
/**
 * y를 기준으로 x를 계산하는 Bresenham 구현체
 */
void lineBresenhamHigh(GLint x1, GLint y1, GLint x2, GLint y2) {
    int dX = x2 - x1;
    int dY = y2 - y1;
    int xi = 1;
    if (dX < 0) {
        xi = -1;
        dX = -dX;
    }
    int D = (2 * dX) - dY;
    int x = x1;

    for (int y = y1; y <= y2; y++) {
        drawPixel(x, y);
        if (D > 0) {
            x = x + xi;
            D = D + (2 * (dX - dY));
        } else {
            D = D + 2 * dX;
        }
    }
}

/**
 * x를 기준으로 y를 계산하는 Bresenham 구현체
 */
void lineBresenhamLow(GLint x1, GLint y1, GLint x2, GLint y2) {
    int dX = x2 - x1;
    int dY = y2 - y1;
    int yi = 1;
    if (dY < 0) {
        yi = -1;
        dY = -dY;
    }
    int D = (2 * dY) - dX;
    int y = y1;

    for (int x = x1; x <= x2; x++) {
```

```
        drawPixel(x, y);
        if (D > 0) {
            y = y + yi;
            D = D + (2 * (dY - dX));
        } else {
            D = D + 2 * dY;
        }
    }
}

/**
 * Bresenham 알고리즘 구현체
 */
void lineBresenham(GLint x1, GLint y1, GLint x2, GLint y2) {
    int yDelta = abs(y2 - y1);
    int xDelta = abs(x2 - x1);
    // xDelta, yDelta 의 대소를 비교하여 계산의 기준이 되는 변수를 달리 설정
    if (yDelta < xDelta) {
        if (x1 > x2) {
            lineBresenhamLow(x2, y2, x1, y1);
        } else {
            lineBresenhamLow(x1, y1, x2, y2);
        }
        // 항상 작은 점이 함수의 첫번째 인자로 처리될 수 있게 하는 조건 분기
    } else {
        if (y1 > y2) {
            lineBresenhamHigh(x2, y2, x1, y1);
        } else {
            lineBresenhamHigh(x1, y1, x2, y2);
        }
        // 항상 작은 점이 함수의 첫번째 인자로 처리될 수 있게 하는 조건 분기
    }
}
```

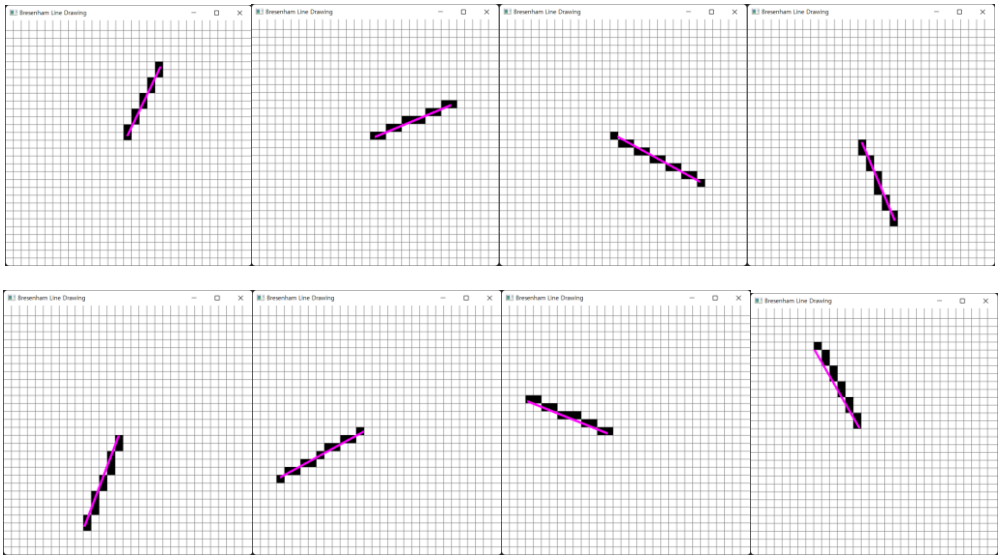
```
/**
 * 점을 8 방향 반사하여 그리는 함수
 */
void drawCirclePoint(GLint x, GLint y) {
    displayPixel(x, y);
    displayPixel(-x, y);
    displayPixel(x, -y);
    displayPixel(-x, -y);

    displayPixel(y, x);
    displayPixel(-y, x);
    displayPixel(y, -x);
    displayPixel(-y, -x);
}

/**
 * Midpoint Circle 알고리즘 구현체
 */
void circleMidPoint(GLint radius) {
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    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);
    }
}
```

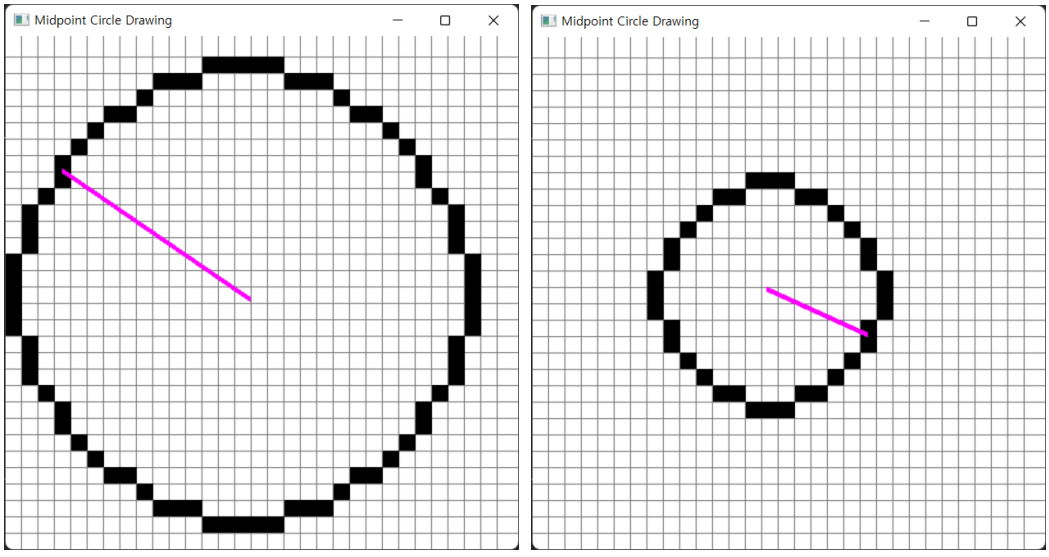
3. 테스트 결과

실습 과제 10: Bresenham과 Midpoint Circle 알고리즘 확장하여 구현하기

```
Basic Line Drawing (b)
DDA Line Drawing (d)
Bresenham Line Drawing (r)
Midpoint Circle Drawing (c)
Exit (q)
```



< Bresenham Line 알고리즘 >



<Midpoint Circle 알고리즘>

4. 느낀 점

수업 시간에 배운 래스터화 알고리즘을 직접 구현해보니 각 알고리즘의 원리가 어떻게 되는지 완전히 이해할 수 있어서 좋았다. 부동소수점 연산을 줄이기 위해서 이런 알고리즘까지 고안해 내는 등, 컴퓨터 그래픽스에서 조금이라도 더 빠른 방법을 위해 고생하신 분들이 새삼 대단하게 느껴졌다. 이번 과제로 익힌 알고리즘은 향후에 래스터화가 필요한 다양한 분야에서 사용될 수 있을 것 같아서 도움이 많이 될 것 같다.