목차

1. 문제에 대한 분석 및 해결 방법

실습 과제 09: 선분 절단 알고리즘 구현하기

이번 과제의 핵심은 평면 분할을 이용한 Liang-Barsky 알고리즘과 매개변수 방정식을 이용한 Cohen-Sutherland 알고리즘을 정확하게 구현하는 것이다. 따라서 수업 시간에 배운 두 알고리즘의 절차와 의사 코드를 참조하여, Region, Point2D, PointPair등의 평면, 점, 직선과 경계 박스를 표현하는 자료형을 정의하고 실제로 작동하는 알고리즘을 구현하고, OpenGL 그래픽 프로그램에 적용하여 작동을 테스트하였다.

2. 자신이 구현한 주요 코드

실습 과제 09: 선분 절단 알고리즘 구현하기

```
/**
 * 경계 박스와 점을 매개변수로 받아 점의 위치를 Region 열거형으로 반환한다
 * @param boundary 경계 박스
 * @param point 위치를 판별할 점
 * @return 점의 Region
 */
Region toRegion(PointPair boundary, Point2D point) {
    int region = INSIDE;
    if (point.x < boundary.start.x) {
        region |= LEFT;
    } else if (boundary.end.x < point.x) {
        region |= RIGHT;
    }
    if (point.y < boundary.start.y) {
        region |= BOTTOM;
    } else if (boundary.end.y < point.y) {
        region |= TOP;
    }
    return (Region) region;
}
```

```
/**
 * Cohen-Sutherland 알고리즘 구현체
 * @param boundary 경계 박스
 * @param line 자를 직선
 * @param acceptCallback aceept 시 호출할 콜백 함수
 * @return aceept 여부 (bool)
 */
bool clipCohenSutherland(PointPair boundary, PointPair line, void
(*acceptCallback)(PointPair line)) {
    while (true) {
        Region startRegion = toRegion(boundary, line.start);
        Region endRegion = toRegion(boundary, line.end);

        if (startRegion == INSIDE && endRegion == INSIDE) {
            acceptCallback(line);
```

```
            return true;
        } else if (startRegion & endRegion) {
            return false;
        } else {
            Region outerRegion = startRegion;
            if (outerRegion == INSIDE) {
                outerRegion = endRegion;
                // 항상 start, end 점 중 바깥에 있는 점의 Region 을 사용하도록 처리
                SWAP(line);
                // 항상 x1, y1 을 외부의 있는 점으로 설정함
            }
            double m = GRADIENT(line);
            if (outerRegion & LEFT) {
                y1 += (xMin - x1) * m;
                x1 = xMin;
            } else if (outerRegion & RIGHT) {
                y1 += (xMax - x1) * m;
                x1 = xMax;
            } else if (outerRegion & BOTTOM) {
                x1 += (yMin - y1) / m;
                y1 = yMin;
            } else if (outerRegion & TOP) {
                x1 += (yMax - y1) / m;
                y1 = yMax;
            }
        }
    }
}
```

```
/**
 * Liang-Barsky 알고리즘에서 사용하는 직선 테스트 함수
 * @param p Liang-Barsky 부등식에서의 p_k
 * @param q Liang-Barsky 부등식에서의 q_k
 * @param u1 P_outer 의 매개변수 (외부->내부 직선의 좌측점)
 * @param u2 P_inner 의 매개변수 (내부->외부 직선의 우측점)
 * @return 클립 테스트 결과 (참인 경우 이어서 다음 클립 테스트 진행)
 */
bool clipTest(double p, double q, double &u1, double &u2) {
    bool result = true;
    double r = q / p;
    if (p < 0) {
        // 외부 -> 내부
        if (r > u2) {
            result = false;
        } else if (r > u1) {
            u1 = r;
        }
    } else if (p > 0) {
        // 내부 -> 외부
        if (r < u1) {
            result = false;
        } else if (r < u2) {
            u2 = r;
        }
    } else {
        // 수직 또는 수평선
        if (q < 0) {
            result = false;
        }
```

```
    }

    return result;
}
```

```
/**
 * Liang-Barsky 알고리즘 구현체
 * @param boundary 경계 박스
 * @param line 자를 직선
 * @param acceptCallback aceept 시 호출할 콜백 함수
 * @return
 */
bool clipLiangBarsky(PointPair boundary, PointPair line, void
(*acceptCallback)(PointPair line)) {
    double u1 = 0;
    double u2 = 1;
    double dx = x2 - x1;
    double dy = y2 - y1;
    PointPair originalLine = line;
    if (clipTest(-dx, x1 - xMin, u1, u2)) {
        if (clipTest(dx, xMax - x1, u1, u2)) {
            if (clipTest(-dy, y1 - yMin, u1, u2)) {
                if (clipTest(dy, yMax - y1, u1, u2)) {
                    if (u2 < 1) {
                        x2 = x1 + u2 * dx;
                        y2 = y1 + u2 * dy;
                    }
                    if (u1 > 0) {
                        x1 = x1 + u1 * dx;
                        y1 = y1 + u1 * dy;
                    }
                }
            }
        }
    }

    if (!LINE_EQUALS(originalLine, line)) {
        acceptCallback(line);
        return true;
    } else {
        return false;
    }
}
```

```
/**
 * 디스플레이 콜백 함수
 */
void display() {
    glClearColor(0.85, 1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    drawBoundaryLine();
    for (PointPair &line: lineList) {
        glColor3f(1, 0, 0);
        glLineWidth(1);
        DRAW_LINE(line);
        glColor3f(1, 1, 0);
        glLineWidth(2);
        clipAlgorithm(boundary, line, accept);
    }
    // 이전에 그린 직선 그리기
```

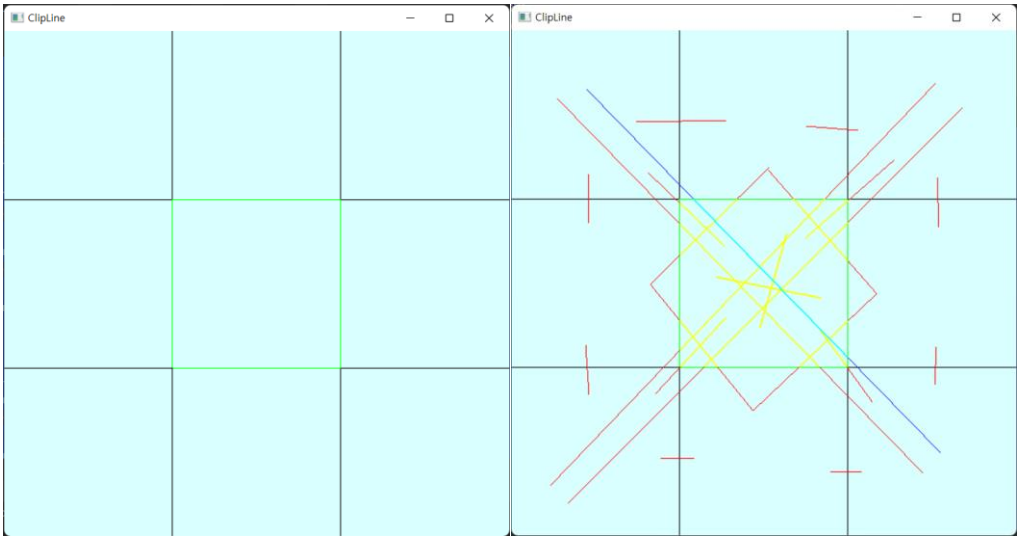
```
    if (isDragging) {
        PointPair line = {lineStart, lineEnd};
        glColor3f(0, 0, 1);
        glLineWidth(1);
        DRAW_LINE(line);
        glColor3f(0, 1, 1);
        glLineWidth(2);
        clipAlgorithm(boundary, line, accept);
    }
    // 드래그 중인 직선 그리기

    glutSwapBuffers();
}
```

3. 테스트 결과

실습 과제 09: 선분 절단 알고리즘 구현하기

```
Use Cohen-Sutherland Algorithm (c)
Use Liang-Barsky Algorithm (l)
Clear line (r)
Exit (q)
```




```
Algorithm: Cohen-Sutherland
Line: (0.073333, 0.580000) to (-0.606667, -0.050000) ---> Accept: (-0.333333, 0.203235) to (-0.192910, 0.333333)
Algorithm: Cohen-Sutherland
Line: (0.170000, 0.523333) to (0.386667, -0.186667) ---> Accept: (0.333333, -0.011897) to (0.227981, 0.333333)
Algorithm: Cohen-Sutherland
Line: (-0.556667, -0.606667) to (-0.203333, -0.606667) ---> Reject
Algorithm: Liang-Barsky
Line: (0.240000, -0.703333) to (0.520000, -0.446667) ---> Reject
Algorithm: Liang-Barsky
Line: (0.716667, -0.250000) to (-0.510000, -0.453333) ---> Accept: (0.333333, -0.313542) to (0.213934, -0.333333)
```

1. 마우스를 드래그해서 화면에 직선을 그릴 수 있게 구현
2. 그린 직선은 초록색 박스를 경계로 클립핑 된 부분이 다른 색으로 표시
3. 이미 그린 직선은 빨간색과 노란색으로, 지금 드래그 중인 직선은 파란색과 하늘색으로 표시

4. 직선을 그릴 때마다 현재 사용 중인 알고리즘, Accept 여부, 원래 직선 정보, 잘린 직선 정보 출력

4. 느낀 점

수업 시간에 배운 두 직선 클리핑 알고리즘을 직접 구현하면서 수업 시간에 잘 이해하지 못했던 부분까지 완전히 이해되어 좋았고 OpenGL으로 클리핑된 직선을 가시화하는 프로그램을 만들어서 직접 직선이 잘라져서 화면에 나타나는 모습을 보니 뿌듯했다. 특히 Cohen-Sutherland 방법은 평면을 나누고, 점의 구획을 구분하는 부분이 직선 클리핑이 아니더라도 다른 알고리즘을 만들 때도 많이 사용될 것 같아 앞으로도 많은 도움이 될 것 같다.