심화전공실습

HW #12

Self-scoring table

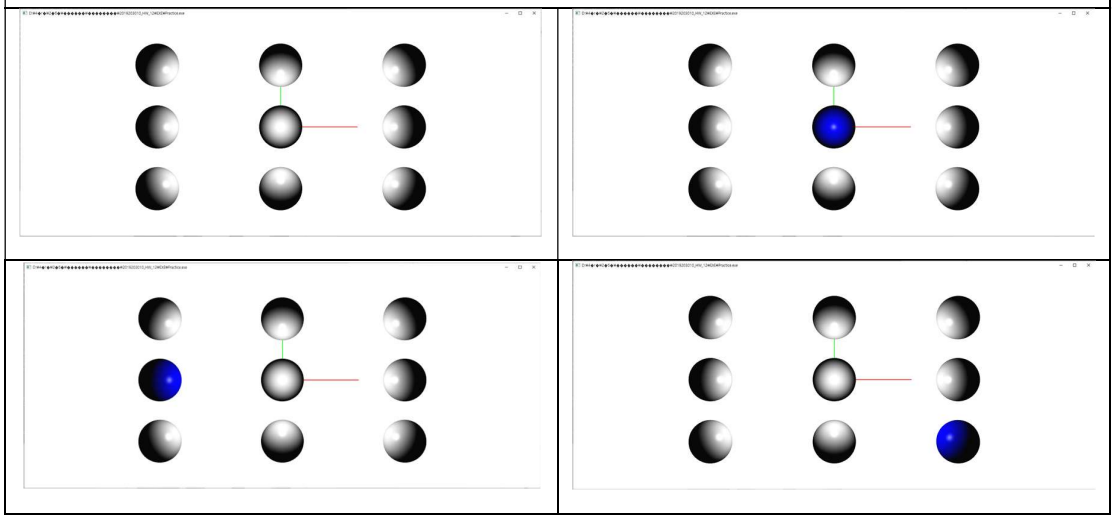
	P01	E01	E02	Total
Score	1	1	1	3

과목명	심화전공실습
학부	소프트웨어학부
학번	2019203010
이름	김민철
제출일자	2024년 11월 24일

I. Practice

Practice 01. Picking an object using selection mode

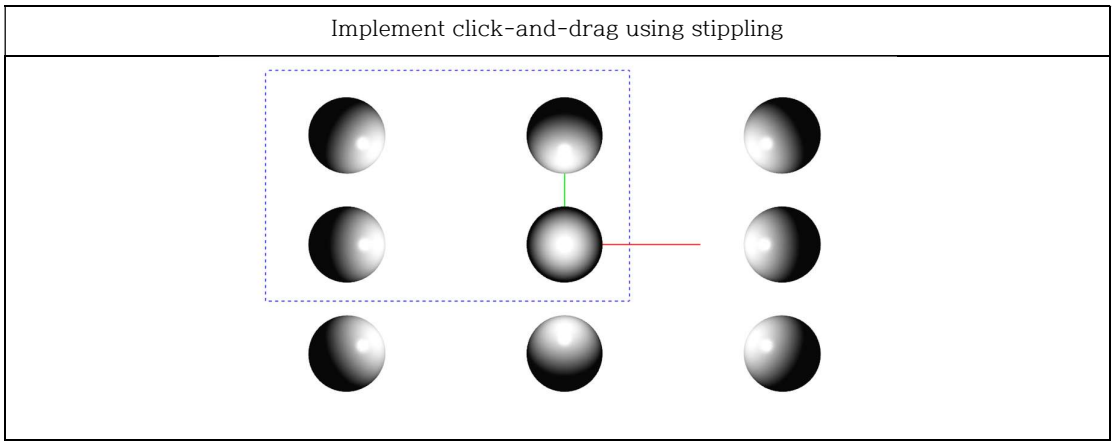
A point light rotating around 3x3 spheres

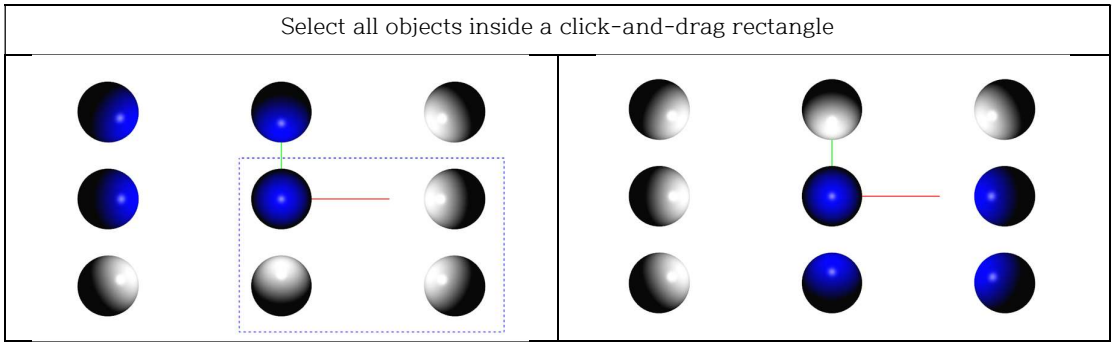


II. Exercise

Exercise 01, Exercise 02

Implement click-and-drag using stippling






```
// Picking
bool picked[9] = { 0,0,0,0,0,0,0,0,0 };

//points for the line segments
double point[2][2] = { {0, 0}, {0, 0} };
float square_p[4][2] = { {0, 0}, {0, 0}, {0,0}, {0,0} };

//Mouse input mode
enum class InputMode
{
    NONE = 0,
    DRAGGING = 1,
    COMPLETE = 2,
};

InputMode inputmode = InputMode::NONE;
```

picked[9] 배열

- 역할: 9 개의 객체가 선택(picked)되었는지를 저장.
- 값: bool 타입 (true/false).

Point[2][2] 배열

- 역할: 마우스 드래깅과 관련된 시작점 및 대각선 끝점의 좌표를 저장.
- 구성:
  - 첫 번째 행: 드래깅 시작점의 좌표.
  - 두 번째 행: 드래깅 종료점의 좌표.

square_p[4][2] 배열

- 역할: 드래깅으로 생성된 사각형의 네 꼭짓점 좌표를 저장.
- 구성: 각 행이 하나의 꼭짓점 좌표를 나타냄.

InputMode 열거형(enum class)

- 역할: 마우스 동작 상태를 나타냄.
- 값:
  - NONE: 초기 상태.
  - DRAGGING: 드래깅 중.
  - COMPLETE: 드래깅 완료.
- 초기값: NONE.

```
//mouse dragging -> stippled square
if (inputmode == InputMode::DRAGGING)
{
    glEnable(GL_LINE_STIPPLE);
    glLineStipple(int(3 * dpiScaling), 0xcccc);
}
else glDisable(GL_LINE_STIPPLE);

if (inputmode == InputMode::DRAGGING)
{

    glBegin(GL_LINE_LOOP);
    glVertex2f(square_p[0][0], square_p[0][1]);
    glVertex2f(square_p[1][0], square_p[1][1]);
    glVertex2f(square_p[2][0], square_p[2][1]);
    glVertex2f(square_p[3][0], square_p[3][1]);
    glEnd();
}
```

render 함수

- 역할: 마우스 동작을 기반으로 점선(stippled) 사각형을 그림.

조건: InputMode == DRAGGING

- 현재 상태가 드래깅 중일 때만 사각형을 그림.

점선 그리기

- GL_LINE_STIPPLE 인자와 glLineStipple 메서드 사용.
- 점선 스타일을 지정하여 선을 그림.

사각형 그리기

- GL_LINE_LOOP 사용: 사각형의 윤곽선을 연결하여 그리는 모드.
- square_p 배열: 사각형의 4 개 꼭짓점 좌표 정보를 이용해 사각형을 그림.

```
int findSelected(int select, GLuint selectBuffer[64])
{
    bool    diagnosis = true;

    for (int i = 0; i < 9; i++)
        picked[i] = 0;

    if (diagnosis)  cout << "selections = " << select << endl;

    int     index = 0;
    for (int i = 0; i < select; i++)
    {
        int     n = selectBuffer[index + 0];                        //# of names

        if (selectBuffer[index + 3] >= 0 && selectBuffer[index + 3] <= 8) // if selected
        {
            cout << "\t# of names = " << n << endl;
            cout << "\tnames: ";
            for (int j = 0; j < n; j++)
                cout << selectBuffer[index + 3 + j] << " "; //name print out
            cout << endl;
            picked[selectBuffer[index + 3]] = 1; // picked!!(=selected)
        }

        //to the next available one
        index += (3 + n);
    }
    for (int i = 0; i < 9; i++)
    {
        if (picked[i]) cout << i << " ";
    }
    cout << endl;
}
```

findselected 함수

- **역할**: selectBuffer 에 저장된 선택된 객체 정보를 기반으로 picked 배열 값을 설정.

구체적인 동작

1. **초기화**
   - 함수 시작 시, picked 배열의 모든 값을 0 으로 초기화.
2. **선택된 객체 처리**
   - selectBuffer 의 **3 번째 인덱스**부터 객체 이름이 저장되어 있음.
   - 객체의 이름을 기준으로 해당 객체의 picked 배열에 해당하는 인덱스에 접근.
3. **picked 업데이트**
   - index + 3 을 통해 selectBuffer 의 객체 정보 접근.
   - 해당 객체 이름을 picked 배열의 인덱스로 사용하여 해당 값을 1 로 설정.
   - 1 은 객체가 선택되었음을 나타냄.

```
int selectObjects(GLFWwindow* window) //CHANGING PART
{
    for (int i = 0; i < 9; i++)
        picked[i] = 0;
    //width and height of picking region in window coordinates
    double  delX = abs(point[0][0] - point[1][0]);
    double  delY = abs(point[0][1] - point[1][1]);

    double x = (point[0][0] + point[1][0]) / 2.0; //주의!
    double y = (point[0][1] + point[1][1]) / 2.0;
```

SelectObjects 함수

- **역할**: 선택된 객체의 정보를 selectBuffer 에 저장.

구체적인 동작

1. **선택 영역의 크기 지정**

   - delX, delY: 선택 영역의 넓이와 높이 값을 나타냄.
   - **특징**: 고정된 영역이 아니라 마우스로 그린 사각형의 범위에 따라 동적으로 설정.

2. **좌표 계산**

   - 사각형의 범위는 point 배열의 시작점(point[0])과 끝점(point[1]) 좌표를 활용해 지정.
   - **중심점(x, y) 계산**:
     - x = (point[0][0] + point[1][0]) / 2
     - y = (point[0][1] + point[1][1]) / 2
     - 시작점과 끝점의 중점을 계산해 선택 영역의 중심 좌표로 설정.

```
void worldCoordinate(float xs, float ys, float& xw, float& yw)
{
    // In the world space. See reshape() in glSetup.cpp
    float aspect = (float)screenW / screenH;
    xw = 3.0f * (xs / (screenW - 1) - 0.5f) * aspect;
    yw = -3.0f * (ys / (screenH - 1) - 0.5f);
}

void Setsquare_p(float p1[2], float p2[2]) //stippled square points
{
    if ((p1[0] <= p2[0] && p1[1] <= p2[1]) || (p1[0] >= p2[0] && p1[1] >= p2[1])) {
        square_p[1][0] = p1[0]; square_p[1][1] = p2[1];
        square_p[3][0] = p2[0]; square_p[3][1] = p1[1];
    }
    else if ((p1[0] <= p2[0] && p1[1] >= p2[1]) || (p1[0] >= p2[0] && p1[1] <= p2[1])) {
        square_p[1][0] = p2[0]; square_p[1][1] = p1[1];
        square_p[3][0] = p1[0]; square_p[3][1] = p2[1];
    }
}
```

마우스로 사각형을 그리기 위한 함수

1. worldCoordinate 함수

- **역할**:
  - 현재 화면 좌표계(스크린 좌표계 또는 윈도우 좌표계)를 월드 좌표계로 변환.
- **사용 이유**:
  - 마우스 입력을 월드 좌표계에서 활용하기 위함.

2. setSquare_p 함수

- **역할**:
  - 사각형의 2 개 꼭짓점 좌표를 입력받아 나머지 2 개 꼭짓점을 계산.
- **동작 방식**:
  - 입력받은 2 개 꼭짓점의 좌표를 대소 비교하여 사각형의 나머지 꼭짓점 좌표를 설정.
- **결과**:
  - square_p 배열에 사각형의 네 꼭짓점 좌표가 완성됨.

```
if (action == GLFW_PRESS && button == GLFW_MOUSE_BUTTON_LEFT)
{
    inputmode = InputMode::DRAGGING;

    //
    square_p[0][0] = xw; square_p[0][1] = yw;
    square_p[2][0] = xw; square_p[2][1] = yw;

    Setsquare_p(square_p[0], square_p[2]);
    point[0][0] = xs; point[0][1] = ys;
}

if (action == GLFW_RELEASE && button == GLFW_MOUSE_BUTTON_LEFT)
{
    inputmode = InputMode::COMPLETE;

    //end point
    square_p[2][0] = xw; square_p[2][1] = yw;

    Setsquare_p(square_p[0], square_p[2]);
    point[1][0] = xs; point[1][1] = ys;
    selectObjects(window);
}
```

mouseButton 함수

- **역할**: 마우스 동작 입력으로 발생하는 이벤트를 처리.

구체적인 동작

1. **마우스 클릭 순간**
   - **동작**: 도형의 시작점을 지정.
   - **상태**: InputMode 를 DRAGGING 으로 설정.
   - **좌표 설정**:
     - 시작점과 끝점 좌표를 동일하게 설정(아직 드래깅 중).
   - **결과**: 사각형이 그려지지 않음.
2. **마우스 버튼을 뗀 순간**
   - **동작**: 도형의 끝점을 지정, 사각형이 완성.
   - **상태**: InputMode 를 COMPLETE 로 설정.
   - **좌표 설정**: 끝점 좌표를 재설정.
   - **선택된 객체 처리**:
     - selectObjects 함수 호출.
     - 완성된 사각형 내부에 포함된 객체 정보를 selectBuffer 에 저장.

```
if (inputmode == InputMode::DRAGGING)
{
    worldCoordinate((float)x, (float)y, square_p[2][0], square_p[2][1]);

    Setsquare_p(square_p[0], square_p[2]);
}
```

mouseMove 함수

- **역할**: 마우스 움직임에 따른 이벤트 처리.

구체적인 동작

1. **드래깅 중 마우스 이동**
   - **커서 좌표 변환**:
     - 현재 커서의 좌표를 worldCoordinate 함수를 사용해 월드 좌표계로 변환.
2. **사각형 꼭짓점 설정**
   - **setSquare_p 함수 호출**:
     - 변환된 좌표를 활용해 드래깅 중 생성되는 사각형의 4 개 꼭짓점을 설정.
   - **결과**:
     - 마우스 움직임에 따라 사각형이 실시간으로 업데이트됨.

III. 느낀점

이 코드를 작성하며, 그래픽 프로그래밍에서 사용자 입력과 상호작용의 섬세함을 구현하는 과정이 얼마나 중요한지 느낄 수 있었습니다. 특히, OpenGL의 선택 모드와 뷰포트 변환을 활용해 마우스 드래그로 객체를 선택하는 기능을 구현하면서 좌표계 변환의 정확성, 입력 이벤트 처리의 타이밍, 그리고 시각적 피드백의 중요성을 깨달았습니다. 또한, 픽킹과 객체 선택을 위한 논리적 구조를 설계하면서 효율적인 데이터 관리와 조건 처리의 필요성을 경험했습니다. 이 과정은 단순히 그래픽스를 넘어서 시스템 설계의 기초 역량을 강화하는 계기가 되었고, 앞으로도 사용자 친화적인 인터페이스를 구현하는 데 큰 밑거름이 될 것이라고 생각합니다.