CG Final Project

[Minecraft 3D Maze with SOR Modeler]

- **수 강 과 목** : 컴퓨터 그래픽스
- **담 당 교 수** : 서 상 현
- **제 출 일** : 2024/12/15
- **학 부** : 예술공학부
- **학 번** : 20213222
- **성 명** : 성 현 우

목 표

1. 개요

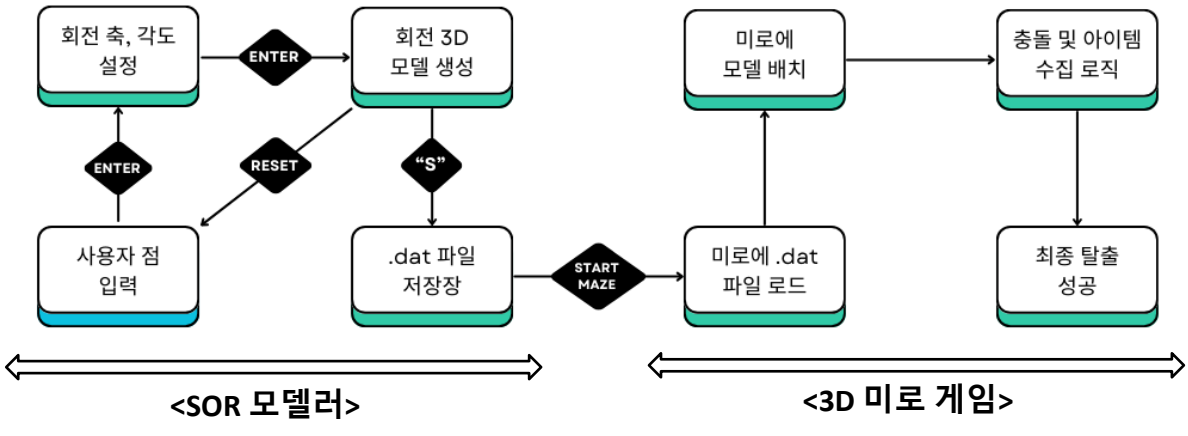
이번 프로젝트는 OpenGL 과 GLUT 를 활용하여 SOR 모델러와 3D 미로 게임을 개발하는 것을 목표로 진행되었습니다. 마인크래프트의 블록 기반 세계와 다양한 텍스처 디자인에서 영감을 받아, 미로의 벽과 바닥에 텍스처를 매핑함으로써 몰입감 있는 게임 환경을 구성하고자 하였습니다.

프로젝트는 두 가지 주요 프로그램으로 이루어졌습니다. 첫번째는 SOR 모델러로, 사용자가 2D 평면에서 점을 찍고 특정 축을 기준으로 회전시켜 3D 모델을 생성할 수 있는 도구입니다. 이 프로그램은 간단한 조작만으로도 직관적으로 3D 모델을 만들 수 있도록 설계하였습니다. 두번째는 1 인칭 시점의 3D 미로 게임으로, 플레이어는 키보드 입력을 통해 블록으로 구성된 미로를 탐험하며 아이템을 수집하고 목표 지점에 도달하면 게임이 종료되는 구조로 개발하였습니다.

개발 과정에서는 기능의 다양성보다는 사용자의 몰입감을 높이는 데 중점을 두었습니다. 이동과 회전 같은 기본 동작을 부드럽게 구현하기 위해 이동 속도와 회전 반응을 세밀하게 조정하였습니다. 벽과의 충돌 처리 시, 플레이어가 벽에 부딪히면 단순히 정지하는 대신 벽을 따라 자연스럽게 미끄러지며 이동할 수 있도록 구현하였습니다.

그 외에도 블록 기반 그래픽과 텍스처 매핑을 통해 시각적인 요소를 강조하였습니다. 벽과 바닥, 하늘 등에 다양한 텍스처를 적용하여 미로 탐험을 단순한 길 찾기에서 플레이어들에게 시각적 재미를 주려고 하였습니다. 이러한 세부적인 구현은 게임 플레이의 완성도를 높이는 데 중요한 역할을 하였습니다.

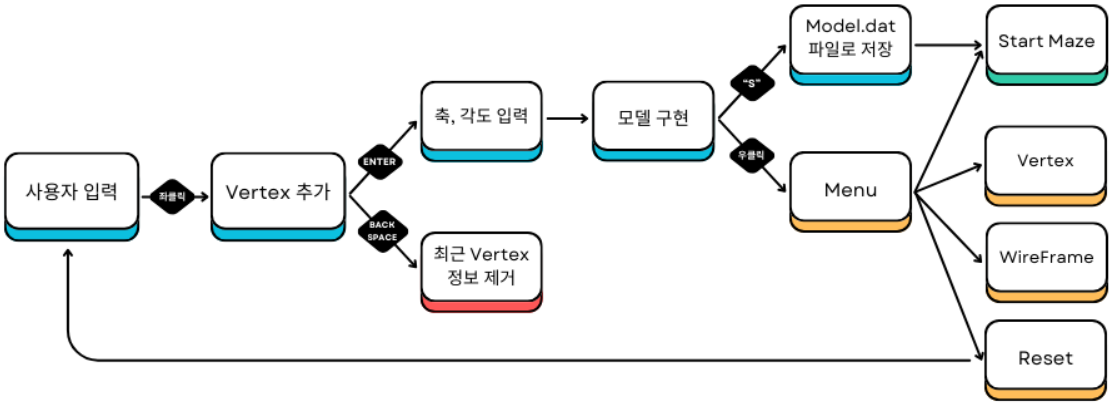
2. 프로젝트 전체 구성 및 흐름



3. SOR 모델러 코드 분석

3.1 기능 개요 및 구현 의도

SOR 모델러는 사용자가 2D 평면에서 점을 추가하고, 회전 축과 각도를 설정해 3D 모델을 생성하는 도구입니다. 생성된 모델은 정점과 폴리곤 정보로 구성된 메쉬로 표현되며, .dat 파일 형식으로 저장할 수 있습니다. 와이어프레임 및 Vertex 만 보기 등 다양한 시각화 옵션과 초기화, 점 삭제 기능을 통해 간편한 모델링 작업을 할 수 있게끔 하였습니다. 아래는 SOR 모델러의 동작 과정을 간단하게 시각적으로 정리한 플로우 차트입니다.



3.2 Vertex 추가 및 제거 기능 구현

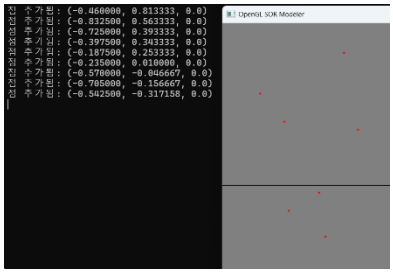
- **Vertex 추가**
  - 마우스 좌클릭으로 화면에 점(vertex) 추가
  - 클릭된 좌표는 정규화된 2D 좌표계(-1.0 ~ 1.0)로 변환되어 points 벡터에 저장

$$\text{변환 x 좌표} = \left(\frac{x}{\text{윈도우 크기}}\right) \times 2.0 - 1.0$$

$$\text{변환 y 좌표} = 1.0 - \left(\frac{y}{\text{윈도우 크기}}\right) \times 2.0$$

  - 정규화 할 때는 위 수식들을 통해 변환
  - 추가된 점들은 화면에 빨간 점으로 표시되며, 좌표값은 콘솔 창에 출력

```
// 마우스 클릭 시 점 추가
void mouse(int button, int state, int x, int y) {
    // 마우스 좌클릭 눌렸을 때 처리
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        float nx = (x / (float)glutGet(GLUT_WINDOW_WIDTH)) * 2.0f - 1.0f;
        float ny = 1.0f - (y / (float)glutGet(GLUT_WINDOW_HEIGHT)) * 2.0f;

        // 변환된 좌표를 points 벡터에 추가
        points.push_back({ nx, ny, 0.0f });

        // 점 좌표 콘솔 창에 출력
        std::cout << "점 추가됨: (" << std::fixed << std::setprecision(6) << nx <<
            ", " << ny << ", 0.0)" << std::endl;
```



- 위 코드는 좌클릭이 눌렸을 때 2D 평면에 Vertex 를 추가하는 기능을 구현한 코드입니다.

- **Vertex 제거**
  - Backspace 키를 눌러 최근에 추가된 Vertex 제거
  - 제거되면 rotatedPoints(점)와 polygonIndice(폴리곤) 정보도 자동으로 갱신

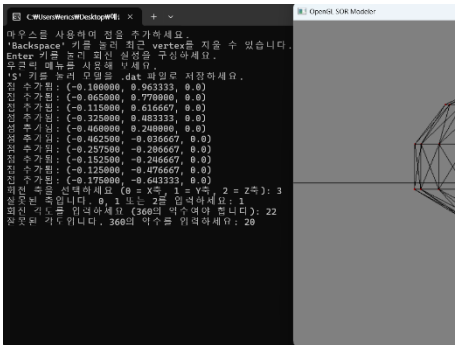
```
// 'Backspace' 키가 눌렸을 경우
else if (key == 8) {  // Backspace 키
    if (!points.empty()) {
        points.pop_back();  // 가장 최근에 추가된 점 제거
        std::cout << "가장 최근의 점이 제거되었습니다.₩n";
```

- 위 코드는 입력된 Vertex 를 'Backspace' 키를 통해 지울 수 있는 기능을 구현한 코드입니다.

3.3 모델 회전 (SOR 생성)

- **회전축 및 각도 입력**
  - Enter 키를 눌러 콘솔 창에서 회전 축(X/Y/Z)을 선택하고, 회전 각도를 입력
  - 각도가 360 의 약수여야 하며, 입력된 각도를 기반으로 SOR 회전 모델이 생성

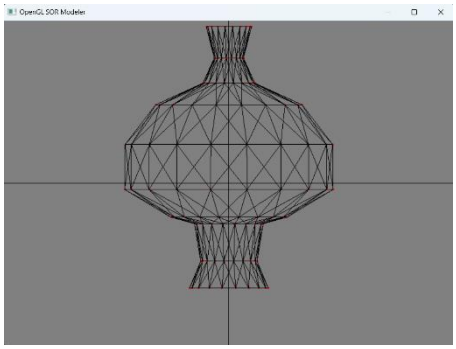
```
// 회전 입력 받는 함수
void handleRotationInput() {
    // 사용자에게 회전 축 선택 요청
    std::cout << "회전 축을 선택하세요 (0 = X축, 1 = Y축, 2 = Z축): ";
    std::cin >> rotationAxis;  // 회전 축 입력 받기

    // 회전 축이 잘못된 경우 재입력 요청
    while (rotationAxis < 0 || rotationAxis > 2) {
        std::cout << "잘못된 축입니다. 0, 1 또는 2를 입력하세요: ";
        std::cin >> rotationAxis;
    }

    // 회전 각도 입력 요청
    std::cout << "회전 각도를 입력하세요 (360의 약수여야 합니다): ";
    std::cin >> rotationAngle;  // 회전 각도 입력 받기

    // 유효하지 않은 각도가 입력되면 재입력 요청
    while ((int)rotationAngle <= 0 || 360 % (int)rotationAngle != 0) {
        std::cout << "잘못된 각도입니다. 360의 약수를 입력하세요: ";
        std::cin >> rotationAngle;
    }
```



- 위 코드는 회전 축과 각도를 입력하기 위한 코드와 해당 코드에 대한 실행 콘솔 창입니다.

- **회전 로직**
  - 회전 각도에 따라 레이어를 나누고, cos, sin 를 이용해 각 축별 회전을 수행
  - 생성된 회전 점들(rotatedPoints)을 기반으로, 삼각형 폴리곤 데이터 (polygonIndices)를 생성하여 모델의 면을 정의

```
// 각 회전 단계마다 회전 변환 수행
for (int i = 0; i <= steps; ++i) {
    float currentAngle = i * radians;  // 현재 회전 각도 계산

    // 현재 회전 단계에서의 점들을 저장할 벡터
    std::vector<Point> layer;
    layer.reserve(P);

    // 원래 점들을 순회하며 회전 변환 수행
    for (const auto& point : points) {
        Point rotated = point;  // 원래 점 복사

        // 회전 축에 따라 다른 회전 변환 수행
        if (rotationAxis == 0) {  // X축 회전
            rotated.y = point.y * cos(currentAngle) - point.z * sin(currentAngle);
            rotated.z = point.y * sin(currentAngle) + point.z * cos(currentAngle);
        }
        else if (rotationAxis == 1) {  // Y축 회전
            rotated.x = point.x * cos(currentAngle) + point.z * sin(currentAngle);
            rotated.z = -point.x * sin(currentAngle) + point.z * cos(currentAngle);
        }
        else if (rotationAxis == 2) {  // Z축 회전
            rotated.x = point.x * cos(currentAngle) - point.y * sin(currentAngle);
            rotated.y = point.x * sin(currentAngle) + point.y * cos(currentAngle);
```



- 위 코드는 축 별로 다른 회전 로직에 대한 코드이고 우측 이미지는 y 축에 대해 회전한 모델입니다.

3.4 와이어프레임, Only Vertex 및 Reset 모드

- **우클릭 메뉴로 기능 선택**
  - 우클릭 메뉴를 통해 4 가지 기능들을 선택 가능

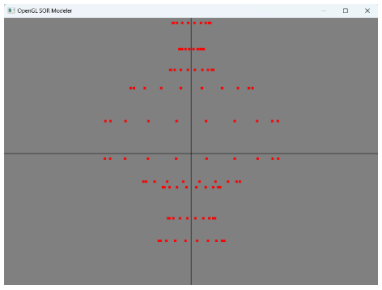
```
// 우클릭 메뉴 생성
void createMenu() {
    glutCreateMenu(handleMenu);  // 메뉴 생성
    glutAddMenuEntry("Wireframe", 1);      // 와이어프레임 표시
    glutAddMenuEntry("Only vertex", 2);    // 점만 표시
    glutAddMenuEntry("Reset", 3);          // 초기 상태로 초기화
    glutAddMenuEntry("Start Maze", 4);     // Start Maze 항목
    glutAttachMenu(GLUT_RIGHT_BUTTON);     // 우클릭 메뉴 연결
}
```

- 위 코드는 우클릭 메뉴의 개요에 대한 코드이고 우측 이미지는 우클릭 시 뜨는 메뉴 창입니다.

- **와이어프레임 모드**
  - 회전된 점들을 기반으로, 삼각형 폴리곤을 와이어프레임 형태로 렌더링
  - 특정 축의 최소/최대 점을 연결하는 선은 표시하지 않도록 처리

- **Only Vertex 모드**
  - 우클릭 메뉴에서 Only Vertex 옵션을 선택하면, 점들로만 화면에 렌더링

```
// Only Vertex 모드
    if (showPointsOnly) {
        glPointSize(5.0f);
        glColor3f(1.0f, 0.0f, 0.0f);
        glBegin(GL_POINTS);
        for (const auto& point : points) {
            glVertex3f(point.x, point.y, point.z);  // 원래 점
        }
        for (const auto& point : rotatedPoints) {
            glVertex3f(point.x, point.y, point.z);  // 회전된 점
        }
        glEnd();
    }
```



- 위 코드는 Only Vertex 모드에 대한 코드이고, 우측 이미지는 Only Vertex 모드를 선택한 모습입니다.

- **Reset 모드**
  - 우클릭 메뉴에서 Reset 옵션을 선택하면, 프로그램 상태 초기화
  - 입력된 점, 회전된 점, 폴리곤 정보, 회전 축과 각도에 대한 데이터 모두 초기화
  - 이후, glutPostRedisplay()를 호출하여 윈도우 창을 새로고침

```
case 3:  // Reset 초기화
    points.clear();
    rotatedPoints.clear();
    polygonIndices.clear();

    // 상태 및 회전 설정 초기화
    showWireframe = true;
    showPointsOnly = false;
    rotationAxis = 0;       // 회전 축 초기화
    rotationAngle = 0.0f;   // 회전 각도 초기화

    std::cout << "데이터가 초기화되었습니다. 새로 점을 찍어주세요.\n";
    glutPostRedisplay();  // 화면 새로고침
    break;
```

```
우클릭 메뉴를 사용해 보세요.
'S' 키를 눌러 모델을 .dat 파일로 저장하세요.
점 추가됨: (-0.100000, 0.963333, 0.0)
점 추가됨: (-0.065000, 0.770000, 0.0)
점 추가됨: (-0.115000, 0.616667, 0.0)
점 추가됨: (-0.325000, 0.483333, 0.0)
점 추가됨: (-0.460000, 0.240000, 0.0)
점 추가됨: (-0.462500, -0.036667, 0.0)
점 추가됨: (-0.257500, -0.206667, 0.0)
점 추가됨: (-0.152500, -0.246667, 0.0)
점 추가됨: (-0.125000, -0.476667, 0.0)
점 추가됨: (-0.175000, -0.643333, 0.0)
회전 축을 선택하세요 (0 = X축, 1 = Y축, 2 = Z축): 3
잘못된 축입니다. 0, 1 또는 2를 입력하세요: 1
회전 각도를 입력하세요 (360의 약수여야 합니다): 22
잘못된 각도입니다. 360의 약수를 입력하세요: 20
데이터가 초기화되었습니다. 새로 점을 찍어주세요
```

- 위 코드는 Reset 모드에 대한 코드이고 콘솔 창에도 데이터가 초기화 되었음을 알 수 있습니다.

3.5 모델 저장 및 외부 실행

- **모델 저장 (.dat 파일)**
  - "S" 키를 눌러 현재 모델의 데이터를 'Model.dat' 파일로 저장
  - 저장되는 데이터는 모든 Vertex 정보와 폴리곤 정보를 포함

```
// 회전된 정점 개수와 폴리곤 면 개수 계산
int vertexCount = static_cast<int>(rotatedPoints.size());    // 정점 개수
int faceCount = static_cast<int>(polygonIndices.size());     // 면 개수

// 정점 정보 기록
file << "VERTEX = " << vertexCount << "₩n";  // 정점 수 출력
for (const auto& v : rotatedPoints) {
    // 소수점 6자리 정점 좌표 기록
    file << std::fixed << std::setprecision(6)
         << v.x << "₩t" << v.y << "₩t" << v.z << "₩n";
}

// 폴리곤 면 정보 기록
file << "FACE = " << faceCount << "₩n";  // 면 개수 출력

for (const auto& tri : polygonIndices) {
    // 폴리곤 면을 구성하는 정점 인덱스를 파일에 기록
    file << tri[0] << "₩t" << tri[1] << "₩t" << tri[2] << "₩n";
}
```

```
VERTEX = 481
0.155000    0.810000    0.000000
0.152500    0.626667    0.000000
0.190000    0.450000    0.000000
0.635000    0.356667    0.000000
0.755000    0.293333    0.000000
0.600000    0.163333    0.000000
0.367500    0.100000    0.000000
0.232500    -0.186667   0.000000
0.645000    -0.396667   0.000000
```

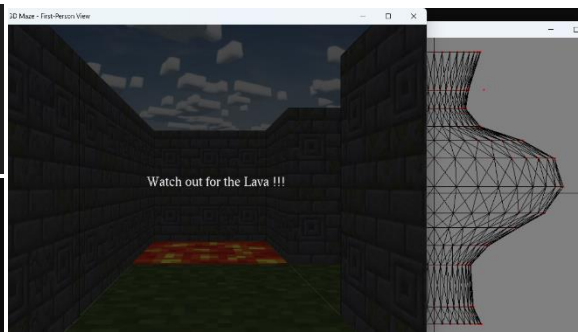
```
FACE = 936
0    13    14
0    14    1
1    14    15
1    15    2
2    15    16
2    16    3
3    16    17
3    17    4
4    17    18
```

- 위 코드는 모델 데이터를 'Model.dat'로 저장하는 코드이고 좌측처럼 Vertex 와 폴리곤 정보가 저장됩니다.

- **미로 시작**
  - 우클릭 메뉴에서 Start Maze 옵션을 선택하면 CG_Final_Maze.exe 를 실행

```
case 4:  // Start Maze 실행
    std::cout << "미로 게임 시작: CG_Final_Maze.exe 실행 중...₩n";
    system("CG_Final_Maze.exe");
    break;
}
glutPostRedisplay();  // 화면 새로고침
```

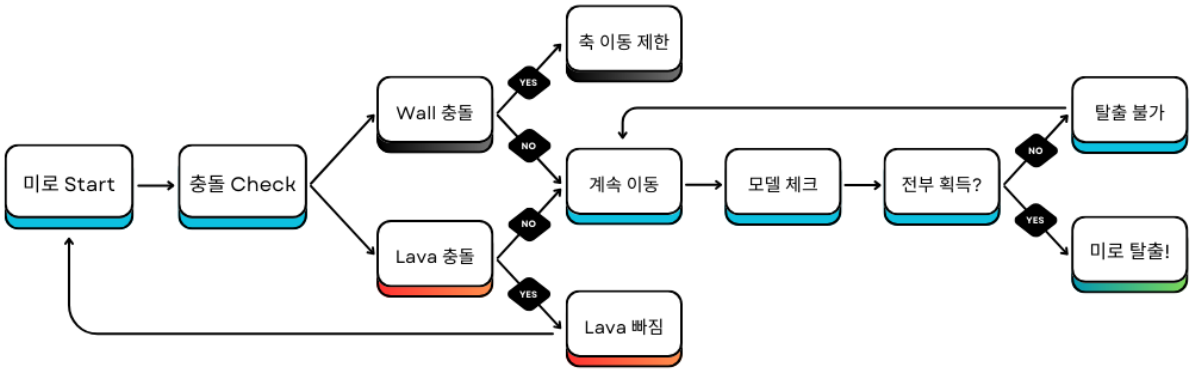
```
미로 게임 시작: CG_Final_Maze.exe 실행 중...
모델 로드 완료: 481개의 정점, 936개의 면.
Model initialized at (8.000000, 0.500000, 3.000000)
Model initialized at (11.000000, 0.500000, 9.000000)
Model initialized at (1.000000, 0.500000, 12.000000)
Total models initialized: 3
```



- 위 사진들은 'Start Maze' 옵션에 대한 코드와 미로가 실행될 시 콘솔 창과 윈도우 창의 모습입니다.

4. 3D 미로 게임 코드 분석

4.1 기능 개요 및 목적



다음은 미로 게임의 진행 과정을 시각적으로 정리한 플로우 차트입니다. OpenGL 과 GLUT 라이브러리를 활용해 1 인칭 시점의 3D 미로 게임을 구현하였으며, 사용자는 키 입력을 통해 자유롭게 이동하고 회전하며 미로를 탐험할 수 있습니다. 용암 구역, 벽 충돌, 모델 수집 등 다양한 게임적 요소와 함께 사운드 및 텍스트 메시지를 실시간으로 출력하여 몰입감을 보다 더 높였습니다. 이어서 주요 기능과 구현 세부 사항을 단계별로 설명하겠습니다.

4.2 WASD 이동 및 Q/E 회전, 충돌 처리

- **W,A,S,D 키**: 플레이어의 전후좌우 이동을 담당.
- **Q,E 키**: 플레이어의 좌우 회전을 담당. cameraYaw 를 증가/감소시키며, 시야를 회전.
- 이동할 때마다 checkCollision() 함수를 호출하여 벽에 부딪히지 않도록 검사.

  - 만약 충돌이 발생하면 축에 대한 이동을 제한하여 벽 따라 이동

```
if (keys['w']) { // 전진
    moveX += dirX * moveSpeed;
    moveZ += dirZ * moveSpeed;
}
if (keys['s']) { // 후진
    moveX -= dirX * moveSpeed;
    moveZ -= dirZ * moveSpeed;
}
if (keys['a']) { // 왼쪽 이동
    moveX += dirZ * moveSpeed;
    moveZ -= dirX * moveSpeed;
}
if (keys['d']) { // 오른쪽 이동
    moveX -= dirZ * moveSpeed;
    moveZ += dirX * moveSpeed;
}
if (keys['q']) { // 좌측 회전
    cameraYaw -= turnSpeed;
}
if (keys['e']) { // 우측 회전
    cameraYaw += turnSpeed;
}
```

```
// 이동 방향이 설정된 경우 충돌을 검사하며 이동
if (moveX != 0.0f || moveZ != 0.0f) {
    float attemptX = cameraX + moveX; // 이동 시도 x좌표
    float attemptZ = cameraZ + moveZ; // 이동 시도 z좌표

    // 충돌이 없으면 이동 적용
    if (!checkCollision(attemptX, attemptZ)) {
        newX = attemptX;
        newZ = attemptZ;
    }
    // x축 또는 z축 방향으로만 이동 가능하게 하기
    else {
        if (!checkCollision(cameraX + moveX, cameraZ)) {
            newX = cameraX + moveX;
            newZ = cameraZ;
        }
        else if (!checkCollision(cameraX, cameraZ + moveZ)) {
            newX = cameraX;
            newZ = cameraZ + moveZ;
        }
    }
```

- 위 코드는 회전, 이동과 카메라의 충돌 감지에 대한 코드입니다.

4.3 텍스처 이미지 맵핑 (벽, 땅, 하늘, 용암)

- 벽(wallTexture), 땅(groundTexture), 하늘(outerWallTexture), 용암(lavaTexture), 목표(GoalTexture) 등 다양한 BMP 텍스처를 로드하고, 사각형과 큐브를 통해 미로 지형을 렌더링.
- LoadTexture() 함수에서 BMP 파일을 읽은 뒤, OpenGL 텍스처로 업로드.
- 셀(maze[z][x])의 값에 따라 다른 텍스처를 바인딩하여 블록에서 그려주기

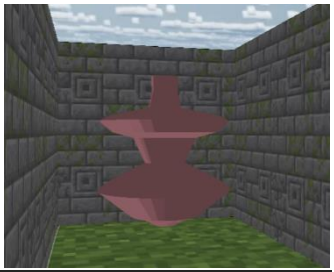
```
// 텍스처 로드 및 설정
wallTexture = LoadTexture("wall_grassbrick.bmp");  // 벽 텍스처 로드
groundTexture = LoadTexture("ground_grass.bmp");   // 땅 텍스처 로드
lavaTexture = LoadTexture("ground_lava.bmp");      // 용암 텍스처 로드
GoalTexture = LoadTexture("Goal_ORE.bmp");         // 목표 지점 텍스처 로드
outerWallTexture = LoadTexture("sky.bmp");         // 외부 벽 텍스처 로드
```


```
for (int z = 0; z < mazeHeight; ++z) { // z축 방향 순회
    for (int x = -4; x < mazeWidth; ++x) { // x축 방향 순회

        // 셀의 값에 따라 다른 텍스처를 바인딩
        if (maze[z][x] == 2) {
            glBindTexture(GL_TEXTURE_2D, lavaTexture); // 셀 값이 2면 용암 텍스처 적용
        }
        else if (maze[z][x] == 3) {
            glBindTexture(GL_TEXTURE_2D, GoalTexture); // 셀 값이 3이면 목표 텍스처 적용
        }
        else if (maze[z][x] == 4) {
            glBindTexture(GL_TEXTURE_2D, groundTexture); // 셀 값이 4이면 일반 땅 텍스처 사용
        }
        else {
            glBindTexture(GL_TEXTURE_2D, groundTexture); // 일반 땅 텍스처
        }
```

- 위 코드는 BMP 텍스처를 로드하고, 배열 값에 따라 지형에 렌더링 하는 코드입니다.

4.4 미로에 모델 .dat 배치 / 모델이 떠있는 모션

- **.dat 파일**로부터 정점/면 정보를 읽어 modelVertices 와 modelFaces 에 저장.
- initializeModels()에서 maze[ ][ ](미로 배열)을 순회하며 4 인 셀마다 모델을 생성
- 공중에 둥둥 떠 있는 모션은 drawModel() 함수에서 구현. glutGet(GLUT_ELAPSED_TIME)을 이용해 시간에 따라 sin() 함수를 적용해, y 좌표를 조금씩 변화시켜 모델이 둥둥 떠 움직이는 효과를 보여줌



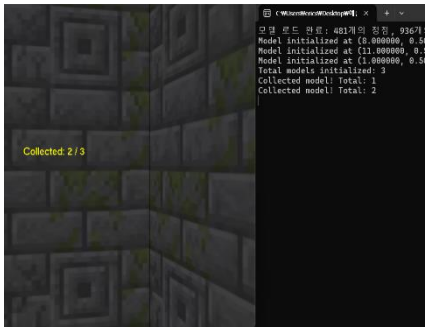
```
float elapsedTime = glutGet(GLUT_ELAPSED_TIME) / 1000.0f; // 경과 시간 (초)
float floatAmplitude = 0.07f; // 플로팅 진폭
float floatFrequency = 1.0f; // 플로팅 주파수 (Hz)
float floatingY = offsetY + floatAmplitude * sin(2 * M_PI * floatFrequency * elapsedTime + phase); // y 오프셋 계산

glTranslatef(offsetX, floatingY, offsetZ);
glScalef(0.4f, 0.4f, 0.4f);    // 모델 크기 축소 (0.4배 크기로 조정)
```

- 첨부한 사진에서는 보이지 않지만 미로 파일을 직접 실행하면 떠 있는 모델을 볼 수 있습니다.
- 위 코드는 배치한 모델이 공중에 둥둥 떠 있는 모션을 구현하기 위한 코드입니다.

4.5 용암 밟으면 초기 위치로 / 아이템 모델 수집+ 개수 카운트

- 용암 타일(미로 배열 값 '2') 위를 밟으면 playWarningSound() 재생 후 일시정지 상태로 전환하고, pauseTimer 가 끝나면 카메라를 시작 위치로 복귀
- 아이템 모델(미로 배열 값 '4')에 접근 시 checkModelCollection() 함수에서 충돌 감지 후 collectedCount 증가, 충돌한 모델은 'collected = true'로 표시해 렌더링에서 제외
- 수집 시 메시지("Collected: n/3")가 표시되고, playCollectSound() 효과음이 재생.

```
void checkModelCollection() {
    for (auto& model : models) {
        if (!model.collected && checkModelCollision(cameraX, cameraZ, model.x, model.z)) {
            model.collected = true;
            collectedCount++;

            printf("Collected model! Total: %d\n", collectedCount);

            playCollectSound();     // 모델 먹는 효과음 재생

            // 수집 메시지 표시
            showCollectedMessage = true;
            collectedMessageTimer = 120;
        }
    }
}
```



- 위 코드는 모델 수집하면 실행되는 코드이고 우측에 모델 수집 시 콘솔 창에 입력되는 것을 볼 수 있습니다.

4.6 조명 초기화, 여러 광원 통한 음영 처리

- initLighting() 함수에서 기본 광원 GL_LIGHT0 을 활성화하고, 주변광(ambient), 확산광(diffuse), 반사광(specular)을 설정하여 모델에 음영 효과를 줌
- glMaterialfv 호출로 재질 속성을 정하여 모델이 광원에 따라 빛나거나 어두워짐
- 초기 메시지 단계에서 ambientLight 를 수정하여 조명을 어둡게 만들고 메시지가 끝나면 다시 밝게 복구하는 로직을 추가

```
// 조명 초기화 함수
void initLighting() {
    glEnable(GL_LIGHTING);  // OpenGL 조명 활성화
    glEnable(GL_LIGHT0);    // 기본 광원(LIGHT0) 활성화

    // 광원 속성 설정
    GLfloat lightPos[] = { 10.0f, 10.0f, 10.0f, 1.0f };     // 광원의 위치 (x, y, z, w)
    GLfloat lightAmbient[] = { 0.2f, 0.2f, 0.2f, 1.0f };    // 주변광 (Ambient Light) 색상 - 희미한 기본 조명
    GLfloat lightDiffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };    // 확산광 (Diffuse Light) 색상 - 물체 표면 색상
    GLfloat lightSpecular[] = { 0.5f, 0.5f, 0.5f, 1.0f };   // 반사광 (Specular Light) 색상 - 빛 반사 효과
```

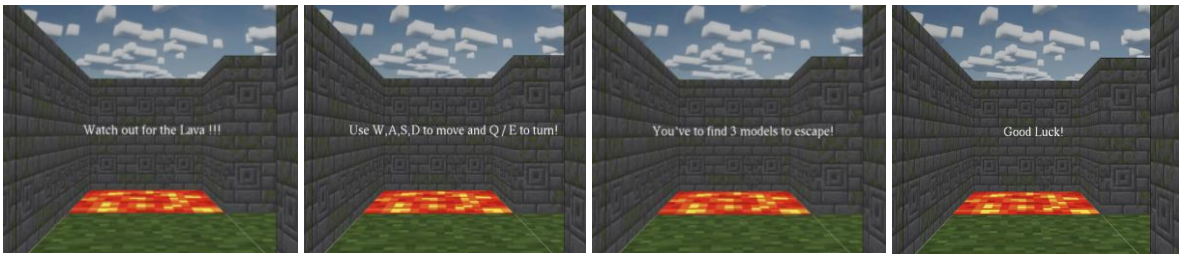
- 위 코드는 광원의 위치, 주변광, 확산광, 반사광의 값을 설정한 코드입니다.

```
// 초기 메시지가 표시 중인 경우
if (messageState < 4) {
    // 메시지가 출력될 때 어두운 효과 적용
    GLfloat ambientLight[] = { 0.1f, 0.1f, 0.1f, 1.0f }; // 어두운 조명
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
else {
    // 메시지 표시가 끝난 후 조명 밝기를 원래대로 복구
    GLfloat ambientLight[] = { 0.5f, 0.5f, 0.5f, 1.0f }; // 밝은 조명
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
}
```

- 위 코드는 메시지가 화면에 표시될 때는 어둡다가 메시지가 끝나면 밝게 하기 위한 코드 입니다.

4.7 텍스트 표시

- **초기 메시지**: 게임 시작 후 messageState 를 통해 순차적으로 "Watch out for the Lava !!!", "Use W,A,S,D to move..." 등의 안내를 표시



- **용암 빠짐**: 플레이어가 이동 중 용암 영역(미로 배열 값 '2')에 진입하면 "Oops! You fell into the lava!!" 메시지를 표시

```
// "이런!" 메시지 표시 여부 확인
if (showOopsMessage && oopsMessageTimer > 0) {
    oopsMessageTimer--;  // 메시지 타이머 감소

    // 깊이 테스트 및 조명 비활성화
    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_DEPTH_TEST);

    // 텍스트 색상 설정 (빨간색)
    glColor3f(1.0f, 0.0f, 0.0f);

    // 화면 중앙에 텍스트 "Oops!" 표시
    drawText("Oops! You fell into the lava!!", 280, 300, GLUT_BITMAP_TIMES_ROMAN_24);
```

- **아이템 획득:** "Collected: n / total" 문구를 짧게 띄운 뒤 사라지게 하기

```
// 텍스트 색상을 노란색으로 설정
glColor3f(1.0f, 1.0f, 0.0f);

// 수집된 개수 메시지 생성
std::string message = "Collected: " + std::to_string(collectedCount) +
        " / " + std::to_string(models.size());
int len = message.length();
float textWidth = 9.0f * len;
float textX = (width - textWidth) / 2.0f; // 너비 중앙에 배치
float textY = height / 2.0f + 20.0f;      // 높이 중앙에 배치

// 텍스트 그리기
drawText(message.c_str(), textX, textY, GLUT_BITMAP_HELVETICA_18);
```

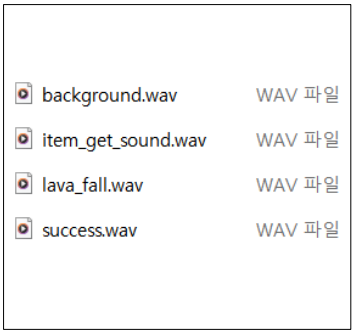
- **미로 탈출 불가**: 아이템을 다 못 모았을 때 목표 지점 도달 시 "You need to collect 'n' more model(s) before escaping!" 메시지 표시.

```
    // 텍스트 색상을 파란색으로 설정
    glColor3f(0.0f, 0.0f, 1.0f);

    // 남은 모델 개수 계산
    int remaining = models.size() - collectedCount;
    std::string message = "You need to collect " + std::to_string(remaining)
        + " more model(s) before escaping!";
    int len = message.length();
    float textWidth = 9.0f * len;
    float textX = (width - textWidth) / 2.0f; // 너비 중앙에 배치
    float textY = height / 2.0f + 20.0f;      // 높이 중앙에 배치
```

- **최종 탈출 메시지**: 아이템을 전부 수집한 뒤 목표 지점 도달 시 "You've escaped from the maze!!" 표시.

```
    // 텍스트 색상을 흰색으로 설정
    glColor3f(1.0f, 1.0f, 1.0f);

    const char* finishMessage = "You've escaped from the maze!!"; // 종료 메시지
    int len = (int)strlen(finishMessage);
    float textWidth = 9 * len;
    float textX = (width - textWidth) / 2.0f; // 텍스트를 화면 너비 중앙에 배치
    float textY = height / 2.0f;              // 텍스트를 화면 높이 중앙에 배치
```

4.8 사운드 재생 (배경, 아이템 획득, 용암, 탈출)



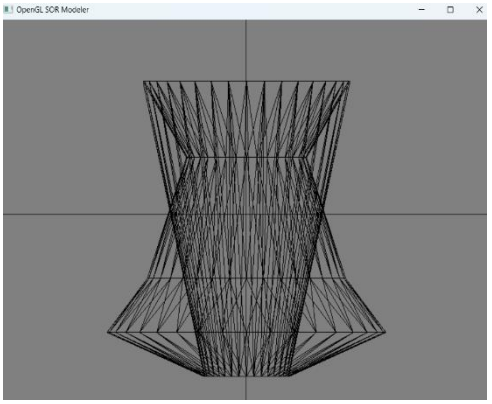
- 배경음악: playBackgroundMusic() 함수로 반복 재생

  - 배경음악은 마인크래프트 음원을 가져왔습니다.

- 아이템 획득: playCollectSound()로 효과음 재생.
- 용암 진입 시: playWarningSound() 재생.
- 탈출 성공: playSuccessSound() 사운드 재생.
- Windows API(PlaySound, mciSendString)를 사용하며, stopBackgroundMusic()으로 음원 정지 가능.

5. 개발 과정에서 어려웠던 점 및 한계점

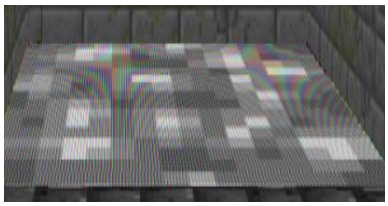
5.1 회전 시 잘못된 버텍스 연결 문제



프로젝트에서 가장 해결하는데 오랜 시간과 노력이 들었던 문제는 회전 변환 과정에서 잘못된 버텍스 간의 연결 문제였습니다. 우측 사진과 같이 버텍스들 임의로 찍은 후에 지정한 축을 기준으로 회전을 하면 회전시킨 축을 기준으로 최대값과 최소값을 가진 버텍스들이 잘못 연결되어 와이어프레임에 시각적 결함이 발생했습니다. 사진에서 볼 수 있듯이 y 축을 기준으로 회전했을 때 y 값이 가장 큰 버텍스들과 가장 작은 값을 가진 버텍스들이 연결되는 문제가 있었습니다.

이를 위해 회전된 점들의 리스트에서 각 삼각형의 변을 순회하며, 회전 축 기준으로 최소값과 최대값을 가지는 점들이 잘못 연결되지 않도록 해주었습니다. 특정 축(X, Y, Z)의 최소값과 최대값을 계산한 후, 삼각형 변의 두 점이 이 값들에 해당하는지를 검사하고, 해당 변은 그리지 않도록 건너뛰도록 코드를 작성했습니다. 이렇게 회전 시 잘못된 버텍스 연결을 방지하고 정확한 와이어프레임을 렌더링합니다.

5.2 BMP 파일 텍스처 맵핑 문제

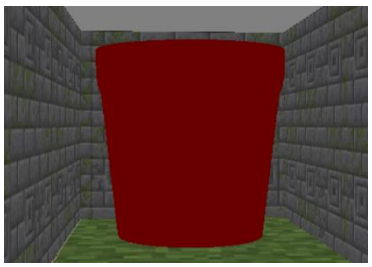


미로의 텍스처 맵핑을 위해 BMP 파일을 로드하는 과정에서 파일이 자주 깨지는 문제가 발생했습니다. 오른쪽 사진과 같이 텍스처가 제대로 적용되지 않거나, 색감 또한 비정상적으로 표시되는 현상이 나타났습니다.

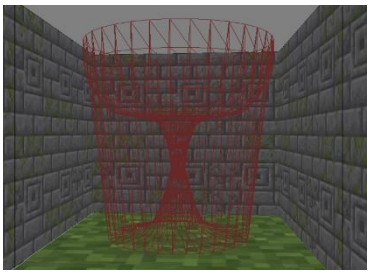
많은 시도 끝에 알아낸 이 문제에 대한 원인은 BMP 파일이 24 비트 형식이 아니거나, 텍스처 크기가 OpenGL 에서 요구하는 정확한 1:1 비율을 충족하지 않았습니다. 오류가 나는 경우를 살펴보니 BMP 파일이 32 비트거나 사진 비율이 정확한 1:1 이 아닌 경우들이었습니다.

따라서 이 맵핑 문제를 해결하기 위해 24 비트 RGB 형식을 강제 적용하도록 로딩 코드를 수정했습니다. 또한 BMP 파일로 변환하기 전에 텍스처 크기를 512x512, 1024x1024 등으로 변환했습니다.

5.3 모델 미로 배치하는 과정에서 발생한 오류



SOR 모델러에서 생성된 모델을 미로로 가져오는 과정에서 또 다시 회전축 기준 최댓값과 최솟값을 가진 버텍스들이 잘못 연결되는 문제가 발생했습니다. 이로 인해 우측 사진과 같이 모델이 왜곡되거나 일부 면이 비정상적으로 생성되는 현상이 나타났습니다.



모델러에서는 모델이 정상적으로 보였으나, 미로로 가져오는 과정에서 예상치 못한 모델이 구현되어서 처음에는 원인을 파악하기 어려웠습니다. 이후 모델을 면(Face) 처리하지 않고 와이어프레임으로만 배치해 본 결과, 우측 사진과 같이 버텍스가 이전과 같이 잘못 연결되어 있다는 것을 확인할 수 있었습니다.

이를 해결하기 위해 SOR 모델러의 방식과 유사하게 폴리곤 생성 코드를 수정하였습니다. 수정된 코드에서는 회전축 기준 극값을 가진 버텍스들을 무조건 연결하지 않도록 코드를 재작성 하여 모델이 정상적으로 생성되도록 하였습니다.

6. 결론 및 요약

본 보고서에서는 OpenGL 과 GLUT 를 기반으로 구현한 SOR 모델러와 3D 미로 게임의 기능과 구조를 중점으로 분석하였습니다. SOR 모델러는 사용자가 쉽게 3D 회전체 모델을 만들고 파일로 저장할 수 있는 기본적인 모델링 도구였습니다. 회전 변환과 폴리곤 생성하는 코드를 통해 직접 구현해서 다양한 모델을 생성할 수 있었습니다. 3D 미로 게임은 이동과 충돌 감지, 텍스처 매핑 같은 그래픽스 기술을 적용해 게임 환경을 만들었고, 카메라 제어와 다양한 메시지 시스템을 통해 게임 플레이의 재미를 더했습니다.

개발 과정에서 마인크래프트의 블록 기반 세계와 텍스처 디자인에서 영감을 받아 몰입감 있는 게임 환경을 구성하고자 노력했습니다. 특히, 미로의 벽과 바닥, 하늘 등에 텍스처를 적용하여 시각적 완성도를 높였으며, 조명과 음영 처리, 사운드 효과 추가를 통해 게임의 생동감을 강화했습니다. 이러한 세부적인 구현은 사용자의 몰입도를 높이는 중요한 요소로 작용했습니다.

프로젝트를 진행하며 다양한 어려움을 마주했습니다. SOR 모델러에서 회전 시 잘못된 버텍스 연결 문제는 수많은 시행착오 끝에 각 회전 점의 최소값과 최대값을 검사하고 올바르게 연결하는 코드를 작성함으로써 해결했습니다. 미로의 텍스처 맵핑 과정에서도 BMP 파일의 비트 수와 크기 비율 문제를 발견하고, 이를 해결하기 위해 파일 포맷과 텍스처 맵핑 코드를 수정했습니다.

3D 미로 게임 개발에서는 충돌 처리, 아이템 수집, 모델 모션 등의 기능을 구현하며 게임 개발의 다양한 측면을 경험할 수 있었습니다. 특히 모델이 공중에 떠서 움직이는 모션과 용암 지역에서의 경고 시스템은 게임의 재미 요소를 더했습니다. 각종 오류와 충돌 문제를 해결하는 과정은 복잡했지만, 코딩에 대한 기본기와 역량을 한층 성장시키는 계기가 되었습니다.

이번 프로젝트는 컴퓨터 그래픽스에 대한 프로그래밍과 소프트웨어 개발에 대한 이해를 심화시키는 값진 경험이었습니다. 프로젝트 초기의 아이디어 구상부터 코드 구현, 문제 해결, 최종 결과물 발표에 이르기까지 모든 과정에서 책임감을 가지고 최선을 다했습니다. 앞으로도 이번 프로젝트에서 배운 점을 바탕으로 더 나은 개발자가 되기 위해 끊임없이 노력하겠습니다. 감사합니다.