3d게임프로그래밍1 숙제 설명문서

2025.05.02

게임공학과

2022184015

김해님

1. 프레임워크 변경 사항

1) GameFramework

기존의 '소프트웨어 렌더러' 프로젝트에는 한가지의 게임만 구현되어 있었기 때문에 GameFramework와 하나의 Scene으로 프로그램의 동작을 관리하던 기존의 구조를 확장시켜야 할 것이라고 생각했다. 그래서 GameFramework에서 전체 4개의 Scene를 관리하도록 하고, 기존의 Scene 클래스를 다른 씬들의 부모로 이용하기 위해 내용을 수정하였다. 이 Scene 클래스를 상속받아 각 Scene들을 제작할 예정이다.

```
class CScene
{
public:
    virtual ~CScene( ) { }

    virtual void BuildObjects( ) { };
    virtual void ReleaseObjects( ) { };

    virtual void Animate(float fElapsedTime) { };
    virtual void Render(HDC hDCFrameBuffer) { };

    virtual void ResetObjects( ) { };

    virtual void OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM lParam) { };
    virtual void OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM lParam) { };
    virtual void ProcessInput(POINT oldCursorPos, HWND hWnd, float m_fElapsedTime) = 0;
};
```

GameFramework의 입력처리 등 함수에서 Scene의 같은 입력처리 함수를 호출한다.

BuildObjects 함수에서 각 씬을 생성하고 각 씬의 BuildObject 함수를 호출한다. 그 이후 Render 또는 Animate의 동작은 CurrentScene만 수행한다. 씬 변경시 ResetObjects 함수를 호출하여 초기화한다.

CGameObject에서 active를 끄면 오브젝트의 충돌과 렌더링을 끄도록 구현했다.

2) Mesh

- CTankMesh

CMesh 클래스에서 상속받은 CTankMesh 클래스를 추가하였다. 이 클래스는 head와 barrel을 나타내는 CMesh* 2개를 가지고 있고, 각 메쉬들의 정점 좌표는 하드코딩 되어있다. 이 정점들을 기반으로 바운딩 박스를 생성한다.

Render함수를 오버라이딩하여 head와 barrel 메쉬 모두 Render를 호출해주었다.

- CObjMesh

CMesh 클래스에서 상속받은 CObjMesh 클래스를 추가하였다. 이 클래스는 생성자에 std::string filepath를 넘겨받아 obj파일을 읽어온다. 다른 Mesh 클래스들과 형태를 유사하게 가져가기 위해 생성자에서 해당 obj 모델에 존재하는 폴리곤의 개수를

세는 'CountOBJToMesh' 함수를 호출한다. 여기서 얻어온 폴리곤의 개수를 이용해 CMesh() 생성자를 호출한다.

Mesh를 생성하였다면 'LoadOBJToMesh' 함수를 호출하여 파일을 읽어온다.

```
int LoadOBJToMesh(const char* filename, std::vector<XMFLOAT3>& vertices, std::vector< std::array<int, 3>>& faces)
{
    std::ifstream file(filename);
    std::string line;

    while (std::getline(file, line))
    {
        if (line.substr(0, 2) == "v ")
        {
            float x, y, z;
            sscanf_s(line.c_str(), "v %f %f %f", &x, &y, &z);
            vertices.push_back(XMFLOAT3(x, y, z));
        }
        else if (line.substr(0, 2) == "f ")
        {
            int a, b, c, t;
            sscanf_s(line.c_str(), "f %d/%d/%d %d/%d/%d %d/%d/%d", &a, &t, &t, &b, &t, &t, &c, &t, &t);
            std::array<int, 3> arr = { a, b, c };
            faces.push_back(arr);
        }
        else continue;
    }
    return (int)faces.size();
}
```

위 함수는 fstream을 이용해 파일을 열고, 한 줄씩 파일을 읽어와 obj 파일 형식에 맞게 파싱한다. input으로 std::vector 2개를 받는데, 이 vector에 정점의 정보와 폴리곤 정보를 저장한다. 위 함수를 통해 받아온 정보들을 다시 재조립하여 폴리곤을 생성하고 SetPolygon 한다.

바운딩 박스를 생성하기 위해 모든 정점의 정보가 담겨있는 벡터를 이용한다. 이 정보를 이용하기 위해 'BoundingBoxFromVert'함수를 추가하였다.

```
void BoundingBoxFromVert(BoundingOrientedBox& oobb, const std::vector<XMFLOAT3>& vertices)
{
    if (vertices.empty()) return;

    XMFLOAT3 minPos(FLT_MAX, FLT_MAX, FLT_MAX);
    XMFLOAT3 maxPos(-FLT_MAX, -FLT_MAX, -FLT_MAX);

    for (const auto& pos : vertices)
    {
        minPos.x = min(minPos.x, pos.x);
        minPos.y = min(minPos.y, pos.y);
        minPos.z = min(minPos.z, pos.z);

        maxPos.x = max(maxPos.x, pos.x);
        maxPos.y = max(maxPos.y, pos.y);
        maxPos.z = max(maxPos.z, pos.z);
    }

    XMFLOAT3 center((minPos.x + maxPos.x) * 0.5f, (minPos.y + maxPos.y) * 0.5f, (minPos.z + maxPos.z) * 0.5f);

    XMFLOAT3 extents((maxPos.x - minPos.x) * 0.5f, (maxPos.y - minPos.y) * 0.5f, (maxPos.z - minPos.z) * 0.5f);

    oobb = BoundingOrientedBox(center, extents, XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f));
}
```

위 함수는 모든 정점들에 대해 x, y, z값의 최대 최소를 구해 모든 정점을 포함하는 바운딩 박스를 생성하는 함수이다.

- CTrackMesh

위에서 설명한 ObjMesh와 똑같이 동작하지만 정점이 4개로 이루어진 폴리곤을 생성한다. 여기서 사용하는 LoadOBJToMesh 함수는 위에 설명한 함수를 오버로딩 한 것으로, 정점이 4개로 이루어진 폴리곤을 사용하는 모델의 포맷을 따르고 있다.

GetNormal 함수를 추가하였다. 이 함수는 폴리곤을 가지고 폴리곤의 법선벡터를 구하는 함수이다. 롤러코스터를 구현할 때 회전 값을 구하기 위해 추가한 함수이다. 정점을 이용해 벡터 2개를 구하고, 두 벡터를 외적하여 법선벡터를 구한다.

```
XMFLOAT3 CTrackMesh::GetNormal(int i)
{
    CPolygon* polygon = m_ppPolygons[i];
    if (!polygon) return XMFLOAT3(0.0f, 1.0f, 0.0f);

    XMFLOAT3 v0 = polygon->m_pVertices[0].m_xmf3Position;
    XMFLOAT3 v1 = polygon->m_pVertices[1].m_xmf3Position;
    XMFLOAT3 v2 = polygon->m_pVertices[2].m_xmf3Position;

    XMFLOAT3 edge1 = Vector3::Subtract(v1, v0);
    XMFLOAT3 edge2 = Vector3::Subtract(v2, v0);

    XMFLOAT3 normal = Vector3::Normalize(Vector3::CrossProduct(edge1, edge2));

    return normal;
}
```

3) 추가 클래스

- CTankPlayer 클래스

헤드와 포신, 카메라를 회전시키는 기능이 추가되어 있다. 소프트웨어 렌더러 프로젝트의 Airplane 클래스와 거의 유사하다.

- CEnemyTank 클래스

머리와 몸통을 따로 회전시키게 하기 위해서 새로 클래스를 만들었다. 적 탱크 객체는 폭발 애니메이션이 있어야 하므로 CExplosiveObject를 상속받았다. 머리와 몸통 모두 충돌 검사를 해야 하기 때문에 피킹 함수를 오버로드하여 둘 다 체크하도록 수정하였다.

2. 시작 화면

1) 구현 계획

직접 큐브를 배치하여 글씨를 쓰는 것은 어렵다고 판단하여 Obj 파일을 읽어서 렌더링하는 코드를 작성하기로 계획했다. 여기서 사용한 Obj 파일 모델은 모두

Blender에서 직접 제작한 것이다.

2) 클래스 구조 및 작동 구조

시작 화면은 'CTitleScene' 클래스로 구현하였고, 이 클래스에는 Player와 글자 메쉬를 가지고 있는 GameObject 2개, 폭발 이펙트가 끝난 후 화면을 전환하기 위한 bool 변수를 가지고 있다.

- 오브젝트 생성

BuildObjects 함수에서 오브젝트를 생성하고 있다. 여기서 이름은 폭발 애니메이션이 필요하므로 ExplosiveObject로 생성하였고, 글자 Mesh는 CObjMesh를 이용해 Obj파일을 읽어왔다. 플레이어는 카메라의 용도로 사용하려고 생성하였기 때문에 Mesh를 추가하지 않았다. 두 오브젝트는 GetRotationAxis() + SetRotationSpeed()로 자동 회전하도록 설정하였다.

- 피킹

피킹 로직은 기존의 소프트웨어 렌더러 프로젝트에서 가져왔으며, 여기서는 이름 오브젝트와의 충돌만을 확인하면 되기 때문에 충돌 여부만을 리턴하도록 반환형을 bool으로 변경하였다. 마우스를 클릭하면 피킹 함수를 호출하고, true를 반환하였다면 이름 오브젝트의 폭발 이펙트를 실행하고 Change를 true로 만든다.

- 오브젝트 업데이트

Animate 함수에서는 Scene를 변경해야 하는지 여부를 판단한다. bChange가 true이고 폭발 이펙트가 끝나면 CGameFramework의 ChangeScene를 true로 만들어서 다음 프레임에 CurrentScene를 변경하도록 지정한다.

3. 메뉴 화면

1) 구현 계획

간단한 ui 화면처럼 보이도록 구현하려고 계획했다. 아무런 움직임이 없는 씬이 심심해 보여서 간단하게 색깔 변경하는 정도만 추가했다. 카메라와 오브젝트 모두 위치가 변하지 않으므로 Animate도 생략했다.

2) 클래스 구조 및 작동 구조

- 오브젝트 생성

Tutorial, Level-1, Level-2, Start, End 글자를 Mesh로 가지는 오브젝트들을 생성했다. 여기서도 Player가 그려질 필요 없어 Mesh를 세팅하지 않았다.

- 피킹

마우스를 오브젝트 위로 올리면 글자 색이 변하는 것을 구현하고 싶어 OnProcessingMouseMessage 함수에 WM_MOUSEMOVE 메시지가 들어올 때마다 피킹 함수를 호출하도록 하였다. 이 클래스에서 피킹 함수는 모든 오브젝트 중 피킹된 오브젝트의 포인터를 반환한다. 아무것도 충돌하지 않았으면 nullptr을 반환한다. bool변수인 doChangeColor 플래그를 둬서 마우스가 빈 곳에 있다면 모든 오브젝트의 색을 검정색으로 변경하도록 구현했다.

- 오브젝트 업데이트

Animate 함수에서 피킹된 오브젝트가 있으면 해당 오브젝트의 색을 랜덤하게 변경하도록 하였다.

4. Level-1: 롤러코스터

1) 구현 계획

blender에서 롤러코스터의 경로로 이용할 Line을 만들어 obj로 불러온다. 여기서 이 경로 정점 정보를 이용하여 Mesh를 생성하려고 했다. 단순하게 x값을 -5.0f, 5.0f씩 더해서 평면을 만드는 것이다. 하지만 롤러코스터의 코스가 위의 방식대로 Mesh를 생성하기엔 너무 복잡했고, 정점 정보만을 이용하기에는 어려움이 있었다. 그래서 결국 경로 Line과 경로 Mesh 둘 다 blender에서 제작해 obj로 가져왔다.

Player Mesh는 카트의 느낌을 내기 위해 간단하게 Cube를 이용하기로 결정했고, 카메라는 1인칭 시점으로 계획했다. 카트가 롤러코스터의 경로를 따라 움직이고 경로 면의 경사에 맞게 회전하도록 구현하려고 노력하였다. 또한 경사의 기울기에 맞춰 가속도를 변경하여 속도감을 주려고 노력했다.

먼저, forward(look)벡터를 현재 지나온 포인트부터 다음 포인트까지 가는 벡터와 방향을 일치시키도록 pitch yaw roll 회전을 시키는 방법을 시도하였다. 하지만 너무 복잡한 계산이 많았고 아무리 봐도 이해가 되지 않아 포기했다. 관련하여 검색했을 때 축 간섭 문제가 생겨 제대로 회전하지 않을 수 있다는 내용도 보았다.

다음으로 생각한 방법은 forward, right, up 벡터를 새로 만들어 세팅하는 것이다. Forward는 쉽게 알 수 있고, up 벡터를 알아내면 외적으로 right 벡터도 구할 수 있을 것이다. 이 세 벡터를 이용하면 물체가 어떻게 회전해야 하는지 정확히 알 수 있을 것이라 생각하여 이 방법을 채택했다. 이때 up벡터는 트랙의 경사면과 수직이어야 하고, 트랙의 경사면을 구해 평면의 법선 벡터를 구하면 된다고 생각했다. 여기서 아까 언급하였던 GetNormal 함수를 이용하였다.

2) 클래스 구조 및 작동 구조

- 클래스 구조

여러 개의 카메라를 써서 시점을 다양하게 보도록 구현하고 싶어서 Player를 여러 개 만들어 시점을 변경 가능하게 했다. m_pCart는 경로를 따라가는 1인칭 시점, m_pDummy는 맵 전체를 볼 수 있게 멀리 배치했다. 둘 중 선택된 시점을 가리키기 위해 m_pPlayer 변수도 추가했다. 카트가 트랙 끝에 도착하고 일정 시간 이후에 씬 전환하려고 타이머 변수를 추가했다. 가속/감속은 m_fCurrentSpeed변수로 처리했다.

- 오브젝트 생성

오브젝트 생성 시에 LoadLineToMesh함수를 이용해 경로를 불러온다.

```
int CScene_1::LoadLineToMesh(const char* filename, std::vector<XMFLOAT3>& vertices, std::vector<std::array<int, 2>>& edges)
{
    std::ifstream file(filename);
    std::string line;

    while (std::getline(file, line))
    {
        if (line.substr(0, 2) == "v ")
        {
            float x, y, z;
            sscanf_s(line.c_str(), "v %f %f %f", &x, &y, &z);
            vertices.push_back(XMFLOAT3(x, y, z));
        }
        else if (line.substr(0, 2) == "l ")
        {
            int a, b;
            std::array<int, 2> arr;
            sscanf_s(line.c_str(), "l %d %d", &a, &b);
            if ( a > b)
                arr = { b - 1, a - 1 };
            else
                arr = { a - 1, b - 1 };
            edges.push_back(arr);
        }
    }
    return (int)edges.size();
}
```

위 함수를 이용해 정점의 정보들과 연결 정보들을 불러온 후, 정점 연결 정보에 맞춰서 정점의 순서를 재배치한다. 여기서 시작점과 끝점의 순서가 꼬이는 경우가 있어 함수 밖에서 추가적으로 시작과 끝을 확인하도록 한다. 경로 자체를 원점 근처에서 시작하도록 만들었기 때문에 원점과의 거리를 측정해서 시작 지점을 판단한다. 만약 순서가 거꾸로 됐다면 reverse를 호출해서 재정렬한다.

```cpp
void CScene_1::BuildOrderedLinePath(const std::vector<XMFLOAT3>& vertices, const std::vector<std::array<int, 2>>& edges, std::vector<XMFLOAT3>& outPath)
{
    std::unordered_map<int, std::vector<int>> adjacency;

    for (const auto& e : edges) {
        adjacency[e[0]].push_back(e[1]);
        adjacency[e[1]].push_back(e[0]);
    }

    int start = -1;
    for (const auto& a : adjacency) {
        if (a.second.size() == 1) {
            start = a.first;
            break;
        }
    }

    if (start == -1) return;

    std::unordered_set<int> visited;
    int current = start;
    int previous = -1;

    while (true) {
        outPath.push_back(vertices[current]);
        visited.insert(current);

        const auto& neighbors = adjacency[current];
        int next = -1;

        for (int n : neighbors) {
            if (n != previous && visited.find(n) == visited.end()) {
                next = n;
                break;
            }
        }

        if (next == -1) break;

        previous = current;
        current = next;
    }
}
```

Track 메쉬를 추가하고 카메라 시점과 위치를 변환시켜 1인칭과 유사한 시점으로 변경한다.

- 롤러코스터 진행(Animate 함수)

1) 종료 조건 확인 – 카트가 경로 끝에 도달했는지 확인하고, 3초 이후 씬 전환
2) 위치 계산

경로 점들의 배열을 이용하여 이동 방향을 결정하고, 가속도와 현재 속도를, 방향을 이용하여 다음 이동 위치를 결정한다.

노말 벡터를 구한 후 뒤집어지는 현상을 막기 위해 보정한다. 이 노말 벡터와 외적을 이용하여 look, up, right를 새로 만든다. 앞으로 튀어나가는 것을 최대한 방지하기 위해 내적을 이용해 서로 반대방향을 향하고 있으면 카트의 위치를 타겟 포인트로 변경하고 다음 경로로 이동한다.

여기서 위치와 회전값을 부드럽게 보간하기 위해서 Lerp 함수를 사용한다.

카메라를 1인칭 시점 forward에 맞게 고정시킨다.

- 추가 키

'1'을 누르면 시점 전환, 'f'를 누르면 고정 시점

5. Level-2: 탱크 게임

1) 구현 계획

- 탱크 구현: 초기 계획

처음에는 모든 파츠를 하나의 메쉬로 만든 다음 일부분만 따로따로 회전을 적용하려고 하였다. 하지만 프로그램 구조 상 그렇게 동작할 수 없다는 걸 금방 알 수 있었고 모든 파츠를 분리하여 다시 시도하였다. 하지만 여기서 포신과 헤드는 결코 각자 다른 방향을 바라볼 수 없다고 생각하였고 하나의 메쉬로 변경하여 지금의 형태가 나왔다. 같은 게임오브젝트는 하나의 회전 변환밖에 적용할 수 없어서 GameObject 객체를 하나 만들어 몸통으로 사용하였다.

- 오브젝트 구현

장애물은 기존의 소프트웨어 렌더러에 있던 움직이는 큐브를 사용하였고, 총알이 장애물에 맞으면 사라지는 것으로 구현하였다.

적 탱크는 머리와 몸통이 분리되어 일정 범위 이상 플레이어가 가까이 오면 포신이 플레이어를 바라보도록 구현할 예정이다. 탱크는 폭발해야 하기 때문에 ExplosiveObject 클래스를 상속받았다.

2) 클래스 구조 및 작동 구조

- 오브젝트 생성

기존 소프트웨어 렌더러의 맵에서 크기만 수정하여 그대로 이용하였다.

- 오브젝트 동작

적 탱크는 소프트웨어 렌더러의 큐브 오브젝트를 바탕으로 평면에서 랜덤 방향으로 움직이도록 구현하였다. 탱크가 같은 평면에서 이동하기 때문에 포신을 위아래로 회전할 필요가 없어 마우스 드래그 시에 좌우로만 회전한다. 피킹등 그 외 동작들은 기존의 소프트웨어 렌더러의 코드를 그대로 이용하였다.

w키를 누르면 모든 적들을 폭발하고 you win! 메쉬를 렌더링한다. 이 메쉬는 그려지는 동안에 일정 거리를 두고 플레이어 위치를 따라간다.