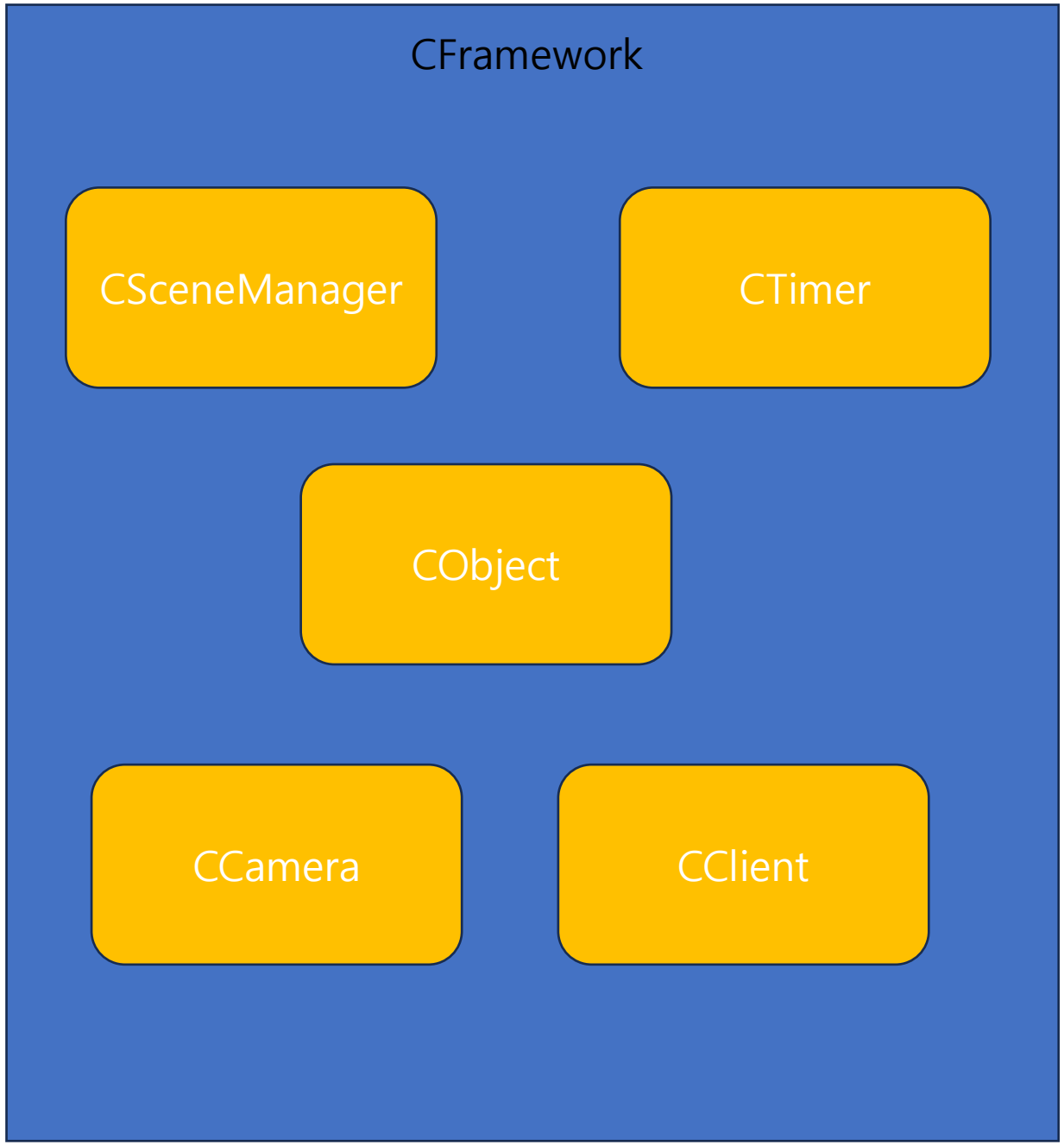
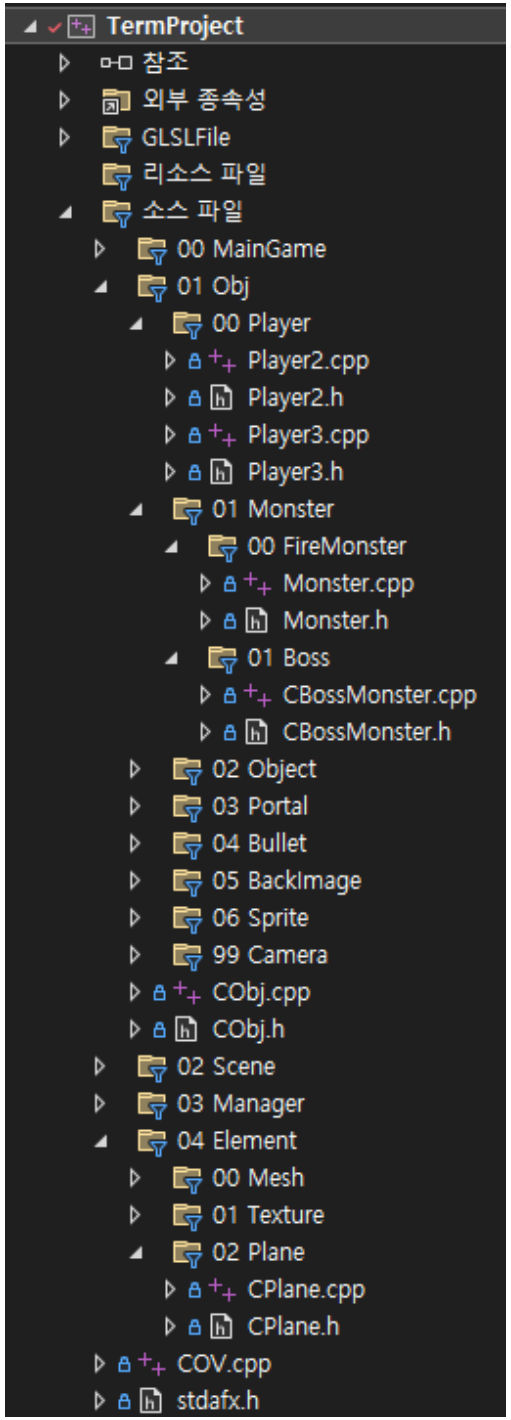
프레임워크

프레임워크란?

- 프로그램의 동작을 모듈화 한 것
- 모든 코드가 하나의 cpp에 있는 것이 아닌 클래스별 코드 구분
- 개발의 효율성 증가, 유지보수 용이


CFramework
CSceneManager
CTimer
CObject
CCamera
CClient


TermProject
참조
외부 종속성
GLSLFile
리소스 파일
소스 파일
00 MainGame
01 Obj
00 Player
Player2.cpp
Player2.h
Player3.cpp
Player3.h
01 Monster
00 FireMonster
Monster.cpp
Monster.h
01 Boss
CBossMonster.cpp
CBossMonster.h
02 Object
03 Portal
04 Bullet
05 BackImage
06 Sprite
99 Camera
CObj.cpp
CObj.h
02 Scene
03 Manager
04 Element
00 Mesh
01 Texture
02 Plane
CPlane.cpp
CPlane.h
COV.cpp
stdafx.h

프레임워크 제작

- 기초적인 베이스 제작
- 파일 분할
- 미리 컴파일 된 헤더
- 사각형 띄우기

프레임 제한

- 컴퓨터 성능에 따라 로직 수행 속도가 다르다
- 게임에서 컴퓨터 성능에 따라 다른 결과가 나오면 안된다
- 보통 게임은 60FPS

프레임 제한 방법

- Windows API이용
  - QueryPerformanceCounter(), QueryPerformanceFrequency()
- C++ Chrono라이브러리 이용
- 등등

키 동시입력

- OpenGL의 glutKeyboardFunc()을 통해 키 입력을 구현하면 동시 키 입력이 되지 않음
- 게임의 경우 동시에 여러 키에 대한 처리를 해야 한다
- 키 버퍼를 통해 매 프레임 키입력을 받고 해당 버퍼를 기준으로 처리한다
- WinAPI의 GetKeyboardState를 통해 구현

씬 구분

- 게임에는 다양한 씬이 있다

- 씬을 구분하지 않고 한번에 모든 객체를 로드하면 메모리 낭비

- 씬마다 다른 동작을 수행할 수 있기 때문에 구분 필요
➔Ex)씬1에서는 알파객체 렌더, 씬2에서는 알파객체 렌더X

- 유지보수

키 입력

- 기존의 키 입력은 Update로 전달
- Update를 가상함수로 변경 시 필요없는 곳에도 키 입력을 전달
- 키입력 처리를 수행하는 CKeyInput클래스를 싱글톤으로 변경

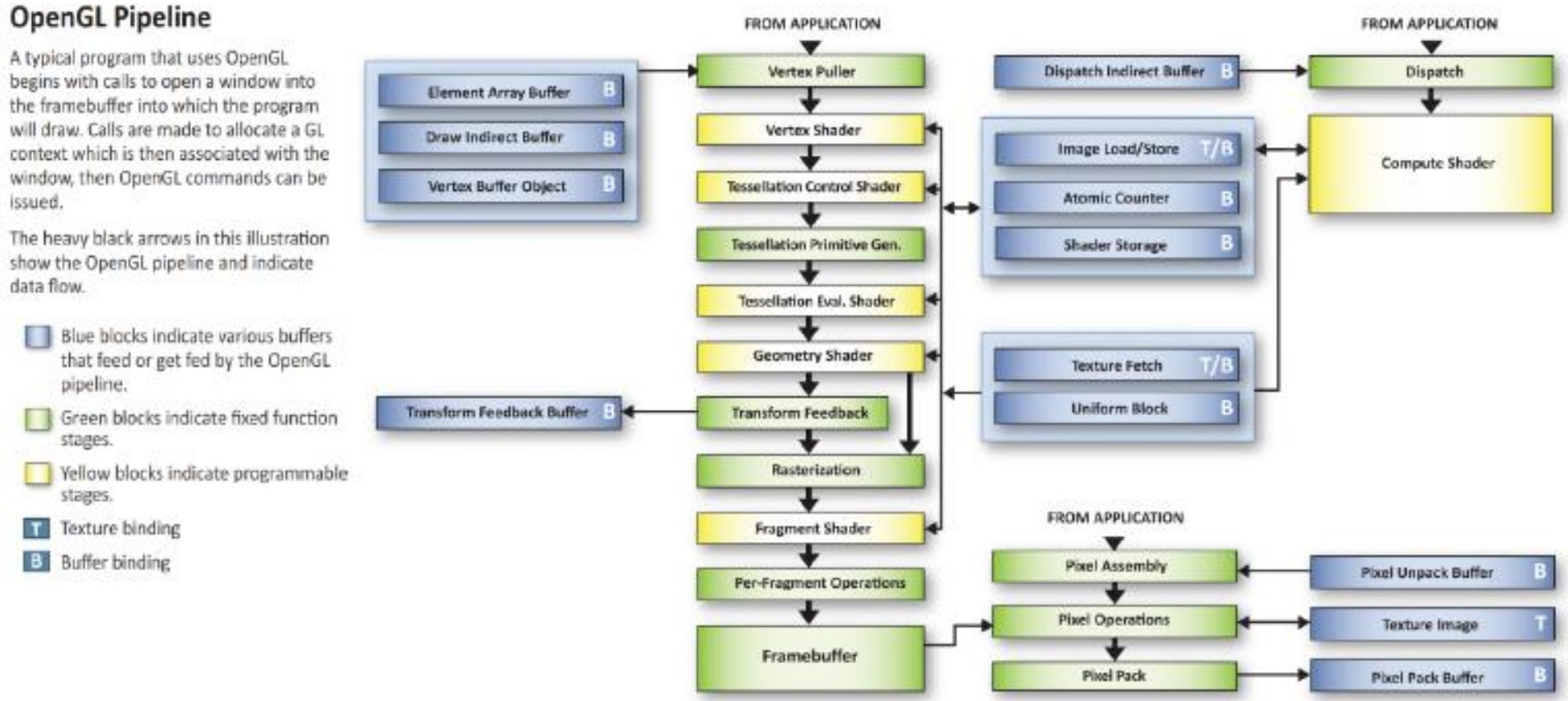
싱글톤

- 객체의 인스턴스를 하나만 생성하는 디자인 패턴
- 전역으로 접근할 수 있다는 장점
- 객체의 생성이 함부로 만들어지면 안되므로 생성자를 private으로 선언
- 싱글톤 객체의 역할이 많아지지 않도록 조심

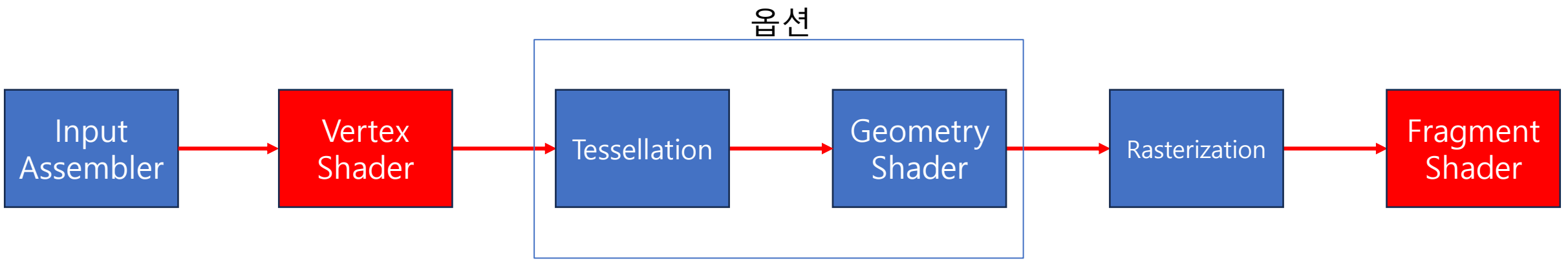
Shader

- 화면에 게임을 그리기 위해서 GPU는 여러 단계를 거친다
- 이 과정을 그래픽스 파이프라인이다
- 그래픽스 파이프라인의 단계 중 일부는 Shader를 통해 동작
- 여러 종류의 Shader를 프로그래밍 할 수 있지만 컴퓨터 그래픽스에서 다루는 Shader는 Vertex와 Fragment

OpenGL 파이프라인



그래픽스 파이프라인



정점 위치, 노말, 텍스쳐 좌표 조작

색상, 조명 등 결정

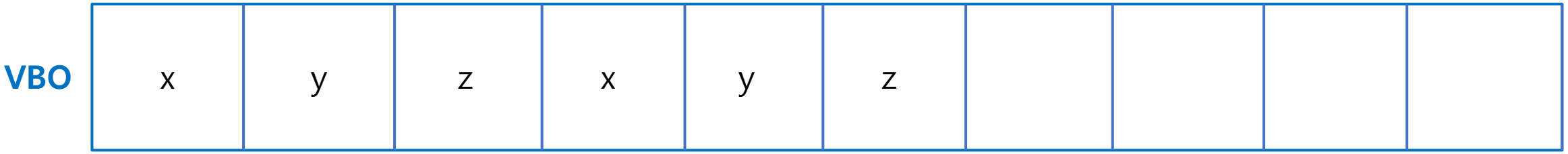
VBO, VAO

- VBO(Vertext Buffer Object)는 삼각형의 꼭짓점 정보들을 담고 있는 배열
- VAO(Vertex Array Object)는 VBO를 어떻게 해석하는지, 메타정보를 제공하는 객체
- glVertexAttribPointer()를 사용하여 VBO정보 설정

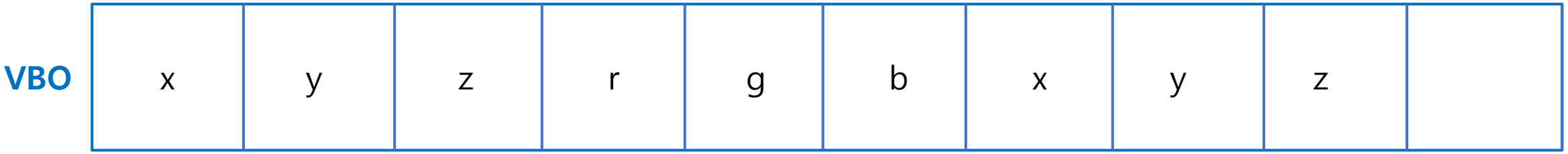
정점 속성 설정

- glVertexAttribPointer (GLuint index, GLint size, GLenum type, GLboolean normalized, GLsizei stride, const GLvoid *pointer)

- index -> 어떤 정보를 의미하는지
- size -> 몇 개의 데이터가 하나의 유의미한 정보를 나타내는지
- type -> 자료형 타입
- Normalized ->정규화 여부
- stride -> 다음 정보가 나오기까지의 간격 크기
- pointer -> 해당 데이터의 오프셋

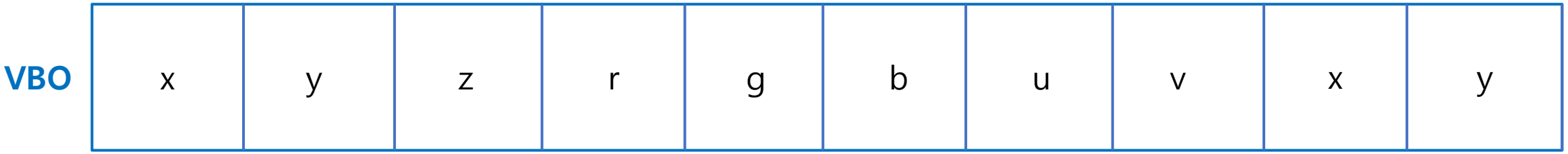


glVertexAttribPointer(?, 3, GL_FLOAT, GL_FALSE, sizeof(float) * 3, 0)



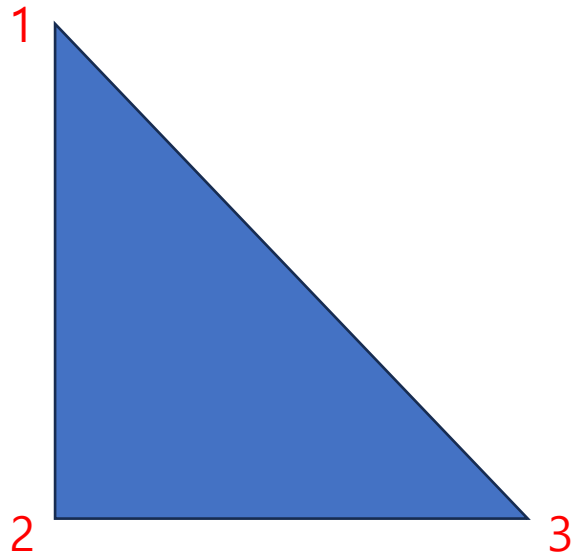
glVertexAttribPointer(?, 3, GL_FLOAT, GL_FALSE, sizeof(float) * 6, 0)

glVertexAttribPointer(?, 3, GL_FLOAT, GL_FALSE, sizeof(float) * 6, (Glvoid*)(sizeof(float) * 5))



좌표, 색상, 텍스쳐 좌표까지 총 3개의 정보가 VBO에 들어가 있다.
이 경우에는 인자가 어떻게 될까?

Vertex정보 넣기



- OpenGL에서는 정점들을 반시계 방향으로 넣어야 한다
- 어디서부터 넣는지는 중요하지 않다. 반시계만 지키자!

- 시계방향으로 넣게 된다면? -> 은면제거

GLSL

바인딩 위치

파이프라인의 이전 단계에서 넘어오는 변수

전역으로 사용 가능한 변수

파이프라인의 다음 단계로 넘기는 변수

```
#version 440
layout(location = 0) in vec3 Pos;
layout(location = 1) in vec3 Color;

uniform vec2 movePos;

out vec4 out_Color;

void main()
{
    vec3 newPos = vec3(Pos.xy + movePos, Pos.z);
    gl_Position = vec4(newPos, 1.0);
    out_Color = vec4(Color, 1.0);
}
```

GLSL(OpenGLShaderLanguage)는 문법이 조금 다르며, 여러가지 함수를 제공한다

https://registry.khronos.org/OpenGL/specs/gl/GLSLangSpec.4.50.pdf

Obj파일

- 3D 오브젝트는 여러 개의 삼각형으로 이루어져있다
- 삼각형은 3개의 점
- 연결된 삼각형들은 점을 공유한다
- 모델의 정보를 저장하고 있으며, 이때 중복된 정보를 최소화하여 가지고 있는 파일

Obj파일을 메모장으로 열어보자

```
v  0.5000 -0.5000 -0.5000
v  0.5000 -0.5000 0.5000
v  -0.5000 0.5000 0.5000
v  0.5000 0.5000 0.5000
v  0.5000 0.5000 -0.5000
v  -0.5000 0.5000 -0.5000
# 8 vertices
```

```
vn 0.0000 -1.0000 -0.0000
vn 0.0000 1.0000 -0.0000
vn 0.0000 0.0000 1.0000
vn 1.0000 0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
vn -1.0000 0.0000 -0.0000
# 6 vertex normals
```

```
vt 1.0000 0.0000 0.0000
vt 1.0000 1.0000 0.0000
vt 0.0000 1.0000 0.0000
vt 0.0000 0.0000 0.0000
# 4 texture coords
```

```
f 1/1/1 2/2/1 3/3/1
f 3/3/1 4/4/1 1/1/1
s 4
f 5/4/2 6/1/2 7/2/2
f 7/2/2 8/3/2 5/4/2
s 8
f 1/4/3 4/1/3 6/2/3
f 6/2/3 5/3/3 1/4/3
s 16
f 4/4/4 3/1/4 7/2/4
f 7/2/4 6/3/4 4/4/4
s 32
f 3/4/5 2/1/5 8/2/5
f 8/2/5 7/3/5 3/4/5
s 64
f 2/4/6 1/1/6 5/2/6
f 5/2/6 8/3/6 2/4/6
# 12 faces
```

- Obj파일의 형식은 해당 정보가 무엇인지 알려주는 알파벳
- 그 뒤에는 해당 알파벳에 대한 정보를 알려준다

#은 주석
v는 정점,
vn은 노말,
vt는 텍스쳐 좌표,
f는 면을 이루는 정점 인덱스

현재 우리에게 필요한 것

- 모델의 노말 값은 나중에 배울 라이팅 작업에 사용
- 모델의 텍스쳐 좌표 값은 텍스쳐를 적용시킬 때

- 따라서 현재 우리가 사용할 정보는 v와 f

```
v  0.5000 -0.5000 -0.5000
v  0.5000 -0.5000 0.5000
v  -0.5000 0.5000 0.5000
v  0.5000 0.5000 0.5000
v  0.5000 0.5000 -0.5000
v  -0.5000 0.5000 -0.5000
# 8 vertices
```

```
f 1/1/1 2/2/1 3/3/1
f 3/3/1 4/4/1 1/1/1
s 4
f 5/4/2 6/1/2 7/2/2
f 7/2/2 8/3/2 5/4/2
s 8
f 1/4/3 4/1/3 6/2/3
f 6/2/3 5/3/3 1/4/3
s 16
f 4/4/4 3/1/4 7/2/4
f 7/2/4 6/3/4 4/4/4
s 32
f 3/4/5 2/1/5 8/2/5
f 8/2/5 7/3/5 3/4/5
s 64
f 2/4/6 1/1/6 5/2/6
f 5/2/6 8/3/6 2/4/6
# 12 faces
```

Obj파일은 어디서?

- 모델링 작업을 할 수 있는 3D Max, Zebra, Blender 등에서 모델을 obj파일로 export할 수 있다

- 사이트에서 다운받아 올 수 있다

Ex) https://www.models-resource.com/

Obj파일 읽기

- 당연히 구현 방법에는 여러가지가 있다
- Obj파일마다 구성이 조금씩 다르기 때문에, 사용할 obj파일에 맞춰 제작하는 것이 가장 좋다

```
void LoadOBJ(const char* filename, vector<GLfloat>& out_vertex, vector<GLfloat>& out_uvs, vector<GLfloat>& out_normals) {
```

- 파일이름, 정점, 텍스쳐 좌표, 노말을 저장할 컨테이너를 인자로 받는다

- 함수에서 반환을 하는 것이 아닌 인자로 받은 vector에 넣을 것이기 때문에 &를 통해 원본을 넘김

```
vector<int> vertexindices, uvindices, normalindices;
vector<GLfloat> temp_vertex;
vector<GLfloat> temp_uvs;
vector<GLfloat> temp_normals;
```

Obj파일에서 읽은 정보를 임시로 저장할 vector
Obj파일에 각 정보가 몇 개가 있는지 모르기 때문에 크기가 가변인 vector를 사용
vertexindices, uvindices, normalindices는 인덱스 저장을 위한 임시 변수

```
ifstream in(filename, ios::in);

if (in.fail()) {
    cout << "Impossible to open file" << endl;
    return;
}
while (!in.eof()) {
    string lineHeader;
    in >> lineHeader;
```

파일을 열고 실패했을 때는 함수 종료
While문에서는 obj파일에 있는 내용을 빈칸이 있기 전까지 읽어서 로직 수행

```cpp
if (lineHeader == "v") {
    glm::vec3 vertex;
    in >> vertex.x >> vertex.y >> vertex.z;
    temp_vertex.push_back(vertex.x); temp_vertex.push_back(vertex.y); temp_vertex.push_back(vertex.z);
}
else if (lineHeader == "vt") {
    glm::vec2 uv;
    in >> uv.x >> uv.y;
    temp_uvs.push_back(uv.x); temp_uvs.push_back(uv.y);
}
else if (lineHeader == "vn") {
    glm::vec3 normal;
    in >> normal.x >> normal.y >> normal.z;
    temp_normals.push_back(normal.x); temp_normals.push_back(normal.y); temp_normals.push_back(normal.z);
}
```

v는 정점 정보 3개가 연달아 있기 때문에 3개를 읽고 선언한 임시 정점 컨테이너에 저장
Vt와 vn의 경우도 동일한 v와 동일

```
else if (lineHeader == "f") {
    string vertex1, vertex2, vertex3;
    unsigned int vertexindex[3], uvindex[3], normalindex[3];

    for (int k = 0; k < 3; ++k) {
        string temp, temp2;
        int cnt{ 0 }, cnt2{ 0 };
        in >> temp;
        while (1) {
            while ((int)temp[cnt] != 47 && cnt < temp.size()) {
                temp2 += (int)temp[cnt];
                cnt++;
            }
            if ((int)temp[cnt] == 47 && cnt2 == 0) {
                vertexindex[k] = atoi(temp2.c_str());
                vertexindices.push_back(vertexindex[k]);
                cnt++; cnt2++;
                temp2.clear();
            }
            else if ((int)temp[cnt] == 47 && cnt2 == 1) {
                uvindex[k] = atoi(temp2.c_str());
                uvindices.push_back(uvindex[k]);
                cnt++; cnt2++;
                temp2.clear();
            }
            else if (temp[cnt] = '\n' && cnt2 == 2) {
                normalindex[k] = atoi(temp2.c_str());
                normalindices.push_back(normalindex[k]);
                break;
            }
        }
    }
}
```

F는 index를 나타내기 위한 용도

While문을 통해 '/'을 제외하고 읽기(47)

F 다음에 연달아 나오는 값의 인덱스는 정점, 텍스쳐, 노말 순

따라서 각 정보를 읽으면서 위에서 선언한 indices벡터에 넣어주기

조금 불필요한 작업이 들어가 있는데, 각자 공부해서 알아서 수정하기 -> 수정하지 않아도 정상적으로 동작

```
for (int i = 0; i < vertexindices.size(); ++i) {
    unsigned int vertexIndex = vertexindices[i];
    vertexIndex = (vertexIndex - 1) * 3;
    glm::vec3 vertex = { temp_vertex[vertexIndex], temp_vertex[vertexIndex + 1], temp_vertex[vertexIndex + 2] };
    out_vertex.push_back(vertex.x); out_vertex.push_back(vertex.y); out_vertex.push_back(vertex.z);
}
for (unsigned int i = 0; i < uvindices.size(); ++i) {
    unsigned int uvIndex = uvindices[i];
    uvIndex = (uvIndex - 1) * 2;
    glm::vec2 uv = { temp_uvs[uvIndex], temp_uvs[uvIndex + 1] };
    out_uvs.push_back(uv.x); out_uvs.push_back(uv.y);
}

for (unsigned int i = 0; i < normalindices.size(); ++i) {
    unsigned int normalIndex = normalindices[i];
    normalIndex = (normalIndex - 1) * 3;
    glm::vec3 normal = { temp_normals[normalIndex], temp_normals[normalIndex + 1], temp_normals[normalIndex + 2] };
    out_normals.push_back(normal.x); out_normals.push_back(normal.y); out_normals.push_back(normal.z);
}
```

- F를 통해서 읽은 인덱스를 기준으로 인자로 받은 벡터에 값을 넣어줌(반환할 값)

- vertexIndices에 저장된 인덱스 값, 즉 obj에서 읽어온 값은 우리가 바로 사용하기 적합하지 않다

- vertexIndex = (vertexIndex - 1) * 3; 을 통해 사용할 수 있는 값으로 가공
  → (vertexIndex - 1)은 one-base의 인덱스를 zero-base로 변경
  → *3은 인덱스들이 벡터를 기준으로 저장되는데 우리는 float으로 1개의 값을 3개로 저장했기 때문에 이를 맞춰주기 위함

- 가공된 값은 반환할 컨테이너에 넣어줌

실제 사용하기

```
void main(int argc, char** argv)
{
    glutInit(&argc, argv); // glut 초기화
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowPosition(0, 0); // 윈도우의 위치 지정
    glutInitWindowSize(WinWidth, WinHeight); // 윈도우의 크기 지
    glutCreateWindow("LoadObj"); // 윈도우 생성(윈도우 이름)
    //--- GLEW 초기화하기
    glewExperimental = GL_TRUE;
    glewInit();
    //--- 세이더 읽어와서 세이더 프로그램 만들기
    make_shaderProgram();

    LoadOBJ("cube.obj", vertex, uv, normal);
    for (int i = 0; i < vertex.size() / 3; ++i) {
        color.push_back(1.0f);
        color.push_back(0.0f);
        color.push_back(0.0f);
    }

    InitBuffer();

    glutDisplayFunc(drawScene); // 출력 함수의 지정
    glutMainLoop(); // 이벤트 처리 시작
}
```

- InitBuffer에서 버퍼를 생성하고 초기화하기 때문에 그보다 먼저 obj로부터 정보를 읽어야함

- Obj를 통해 읽은 정보에는 색상 정보가 없다
- 따라서 색은 직접 넣어주자

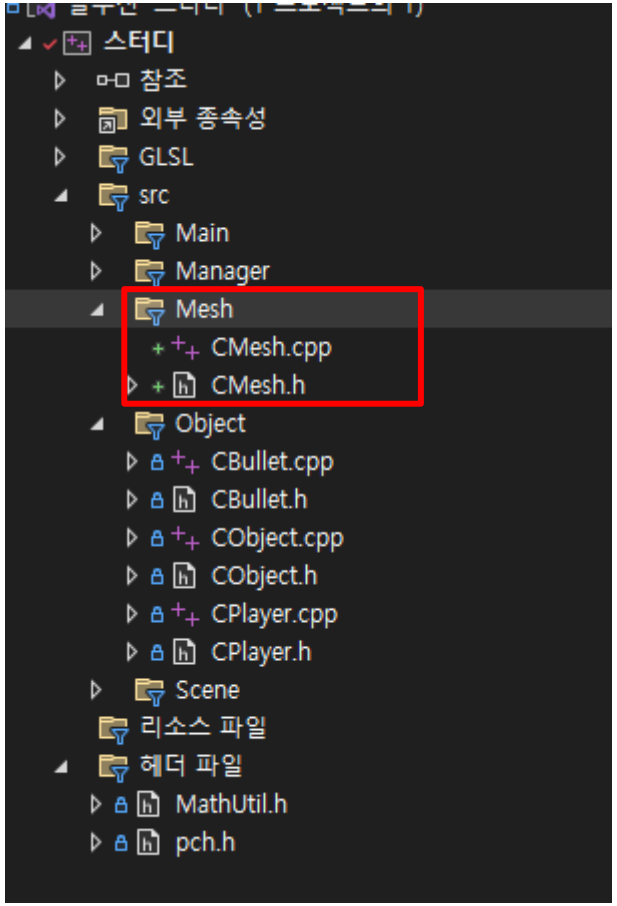
이후 그리기 코드 등은 소스파일을 통해 볼 것
Obj파일 읽는 거 분석해보기
모르면 질문하기

Vao, vbo 만들고 사용하는 것이
이해가 안된다면 공부가 부족한 것이니 공부하세요~

CMesh클래스

- 모델의 정점, UV, 색상, 노말값들을 저장하기 위한 클래스
- 이전 시간에 했던 OBJ로더를 프레임워크에 적용
- Mesh구조체를 Cmesh 클래스로 변경하여 기능 추가
- 일단은 여러 개의 객체가 동일한 Mesh를 공유할 때 하나의 Mesh만을 로드하는 기능 수행

클래스 추가 및 구조체 삭제



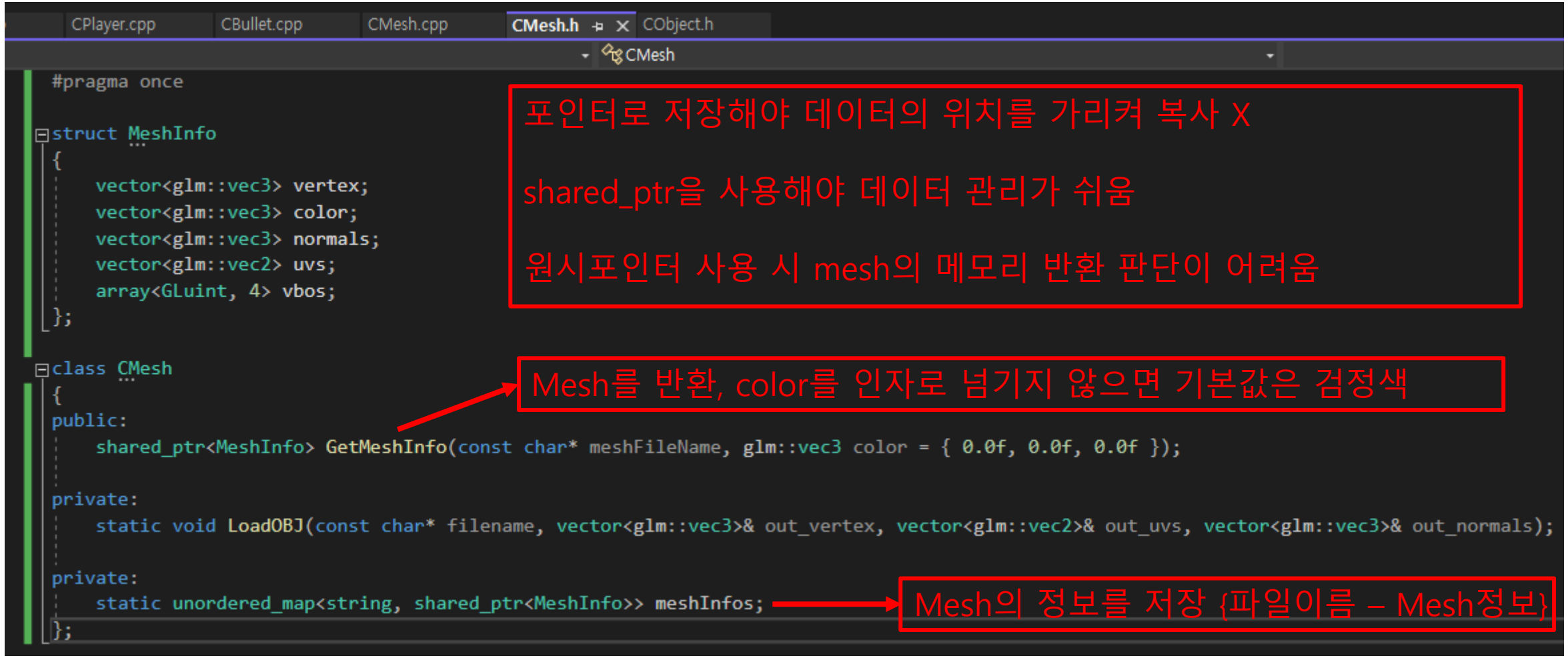
CMesh.h* CObject.h

스터디 CObject

삭제

```h
#pragma once

struct Mesh
{
    vector<glm::vec3> vertex;
    vector<glm::vec3> color;
    array<GLuint, 2> vbos;
};

class CObject
{
public:
    CObject();
    virtual ~CObject();

    virtual bool Update(float elapsedTime);
    virtual void Render();

    void SetDir(const Vector2& dir);

protected:
    Vector2 m_pos;  //사각형의 정가운데
    Vector2 m_dir;  //방향
    Vector2 m_size;
    float m_speed;

    Mesh* m_mesh;
};
```

CMesh.h


CPlayer.cpp
CBullet.cpp
CMesh.cpp
CMesh.h
CObject.h
CMesh
#pragma once
struct MeshInfo
{
vector<glm::vec3> vertex;
vector<glm::vec3> color;
vector<glm::vec3> normals;
vector<glm::vec2> uvs;
array<GLuint, 4> vbos;
};
class CMesh
{
public:
shared_ptr<MeshInfo> GetMeshInfo(const char* meshFileName, glm::vec3 color = { 0.0f, 0.0f, 0.0f });
private:
static void LoadOBJ(const char* filename, vector<glm::vec3>& out_vertex, vector<glm::vec2>& out_uvs, vector<glm::vec3>& out_normals);
private:
static unordered_map<string, shared_ptr<MeshInfo>> meshInfos;
};
포인터로 저장해야 데이터의 위치를 가리켜 복사 X
shared_ptr을 사용해야 데이터 관리가 쉬움
원시포인터 사용 시 mesh의 메모리 반환 판단이 어려움
Mesh를 반환, color를 인자로 넘기지 않으면 기본값은 검정색
Mesh의 정보를 저장 {파일이름 – Mesh정보}

CMesh.h

```
#pragma once

struct MeshInfo
{
vector<glm::vec3> vertex;
vector<glm::vec3> color;
vector<glm::vec3> normals;
vector<glm::vec2> uvs;
array<GLuint, 4> vbos;
};

class CMesh
{
public:
shared_ptr<MeshInfo> GetMeshInfo(const char* meshFileName, glm::vec3 color = { 0.0f, 0.0f, 0.0f });

private:
static void LoadOBJ(const char* filename, vector<glm::vec3>& out_vertex, vector<glm::vec2>& out_uvs, vector<glm::vec3>& out_normals);

private:
static unordered_map<string, shared_ptr<MeshInfo>> meshInfos;
};
```

CMesh.cpp

```cpp
#include "pch.h"
#include "CMesh.h"
#include <fstream>

unordered_map<string, shared_ptr<MeshInfo>> CMesh::meshInfos;

shared_ptr<MeshInfo> CMesh::GetMeshInfo(const char* meshFileName, glm::vec3 color)
{
    if (!meshInfos.contains(meshFileName)) {
        shared_ptr<MeshInfo> meshInfo = make_shared<MeshInfo>();
        LoadOBJ(meshFileName, meshInfo->vertex, meshInfo->uvs, meshInfo->normals);
        for (int i = 0; i < meshInfo->vertex.size(); ++i)
            meshInfo->color.push_back(color);
        meshInfos.insert({ meshFileName, meshInfo });
    }

    return meshInfos[meshFileName];
}

void CMesh::LoadOBJ(const char* filename, vector<glm::vec3>& out_vertex, vector<glm::vec2>& out_uvs, vector<glm::vec3>& out_normals) { ... }
```

Static 변수 초기화

해당 파일이름의 MeshInfo가 없다면
파일을 로드 후 반환
있다면 바로 반환

CMesh.cpp 코드

```cpp
#include "pch.h"
#include "CMesh.h"
#include <fstream>

unordered_map<string, shared_ptr<MeshInfo>> CMesh::meshInfos;

shared_ptr<MeshInfo> CMesh::GetMeshInfo(const char* meshFileName, glm::vec3 color)
{
    if (!meshInfos.contains(meshFileName)) {
        shared_ptr<MeshInfo> meshInfo = make_shared<MeshInfo>();
        LoadOBJ(meshFileName, meshInfo->vertex, meshInfo->uvs, meshInfo->normals);
        for (int i = 0; i < meshInfo->vertex.size(); ++i)
            meshInfo->color.push_back(color);
        meshInfos.insert({ meshFileName, meshInfo });
    }

    return meshInfos[meshFileName];
}

void CMesh::LoadOBJ(const char* filename, vector<glm::vec3>& out_vertex, vector<glm::vec2>& out_uvs, vector<glm::vec3>& out_normals)
{
    vector<int> vertexindices, uvindices, normalindices;
    vector<GLfloat> temp_vertex;
    vector<GLfloat> temp_uvs;
    vector<GLfloat> temp_normals;

    ifstream in(filename, ios::in);

    if (in.fail()) {
        cout << "Impossible to open file" << endl;
        return;
    }
    while (!in.eof()) {
        string lineHeader;
        in >> lineHeader;

        if (lineHeader == "v") {
            glm::vec3 vertex;
            in >> vertex.x >> vertex.y >> vertex.z;
            temp_vertex.push_back(vertex.x); temp_vertex.push_back(vertex.y); temp_vertex.push_back(vertex.z);
        }
        else if (lineHeader == "vt") {
            glm::vec2 uv;
            in >> uv.x >> uv.y;
            temp_uvs.push_back(uv.x); temp_uvs.push_back(uv.y);
        }
        else if (lineHeader == "vn") {
            glm::vec3 normal;
            in >> normal.x >> normal.y >> normal.z;
            temp_normals.push_back(normal.x); temp_normals.push_back(normal.y); temp_normals.push_back(normal.z);
        }
        else if (lineHeader == "f") {
            string vertex1, vertex2, vertex3;
            unsigned int vertexindex[3], uvindex[3], normalindex[3];

            for (int k = 0; k < 3; ++k) {
                string temp, temp2;
                int cnt{ 0 }, cnt2{ 0 };
                in >> temp;
                while (1) {
                    while ((int)temp[cnt] != 47 && cnt < temp.size()) {
                        temp2 += (int)temp[cnt];
                        cnt++;
                    }
                    if ((int)temp[cnt] == 47 && cnt2 == 0) {
                        vertexindex[k] = atoi(temp2.c_str());
                        vertexindices.push_back(vertexindex[k]);
                        cnt++; cnt2++;
                        temp2.clear();
                    }
                    else if ((int)temp[cnt] == 47 && cnt2 == 1) {
                        uvindex[k] = atoi(temp2.c_str());
                        uvindices.push_back(uvindex[k]);
                        cnt++; cnt2++;
                        temp2.clear();
                    }
                    else if (temp[cnt] = 'Wn' && cnt2 == 2) {
                        normalindex[k] = atoi(temp2.c_str());
                        normalindices.push_back(normalindex[k]);
                        break;
                    }
                }
            }
        }
        else {
            continue;
        }
    }
    for (int i = 0; i < vertexindices.size(); ++i) {
        unsigned int vertexIndex = vertexindices[i];
        vertexIndex = (vertexIndex - 1) * 3;
        glm::vec3 vertex = { temp_vertex[vertexIndex], temp_vertex[vertexIndex + 1], temp_vertex[vertexIndex + 2] };
        out_vertex.push_back( vertex );
    }
    for (unsigned int i = 0; i < uvindices.size(); ++i) {
        unsigned int uvIndex = uvindices[i];
        uvIndex = (uvIndex - 1) * 2;
        glm::vec2 uv = { temp_uvs[uvIndex], temp_uvs[uvIndex + 1] };
        out_uvs.push_back(uv);
    }

    for (unsigned int i = 0; i < normalindices.size(); ++i) {
        unsigned int normalIndex = normalindices[i];
        normalIndex = (normalIndex - 1) * 3;
        glm::vec3 normal = { temp_normals[normalIndex], temp_normals[normalIndex + 1], temp_normals[normalIndex + 2] };
        out_normals.push_back(normal);
    }
}
```

빌드를 해보자

빌드가 성공해야 한다

3D출력을 위한 작업

```
#version 440
layout(location = 0) in vec3 vPos;
layout(location = 1) in vec3 in_Color;

uniform mat4 modelTransform;
out vec4 out_Color;

void main()
{
    gl_Position = modelTransform * vec4(vPos, 1.0);

    out_Color = vec4(in_Color, 1.0);
}
```

- 버텍스 쉐이더 수정

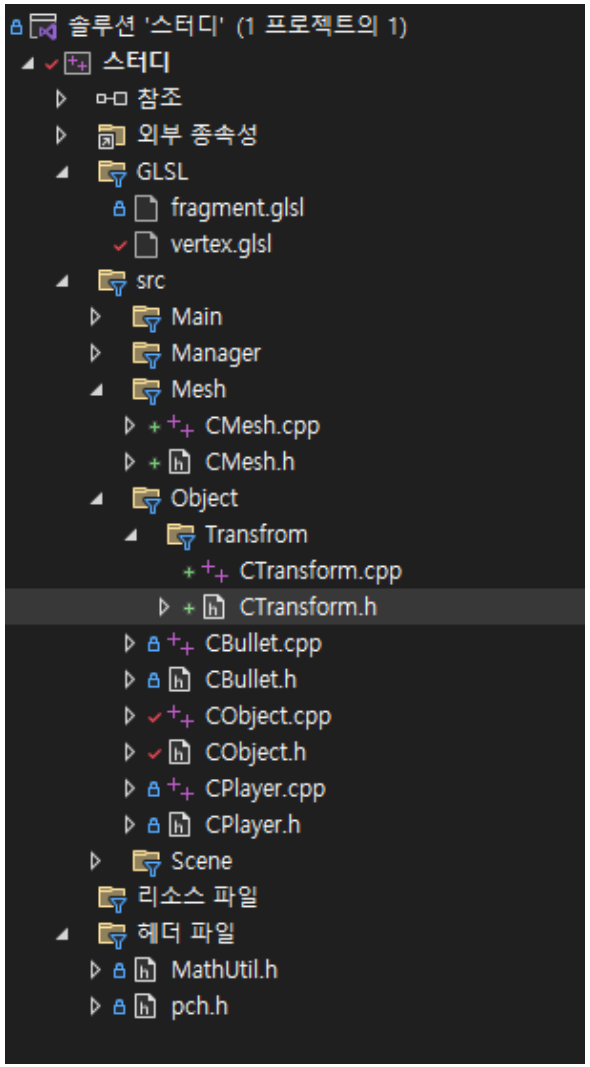
```
#version 440
layout(location = 0) in vec3 vPos;
layout(location = 1) in vec3 in_Color;

uniform mat4 modelTransform;
out vec4 out_Color;

void main()
{
gl_Position = modelTransform * vec4(vPos, 1.0);

out_Color = vec4(in_Color, 1.0);
}
```

CTransform클래스

- 객체의 이동, 회전, 스케일 등을 Object클래스에서 수행하는 것이 아니라 Ctransform클래스의 역할로 이전

- 따라서 행렬, 좌표 등의 값은 Ctransform클래스가 가지고 Cobject는 CTransform클래스의 포인터를 가지도록 수정

클래스 추가

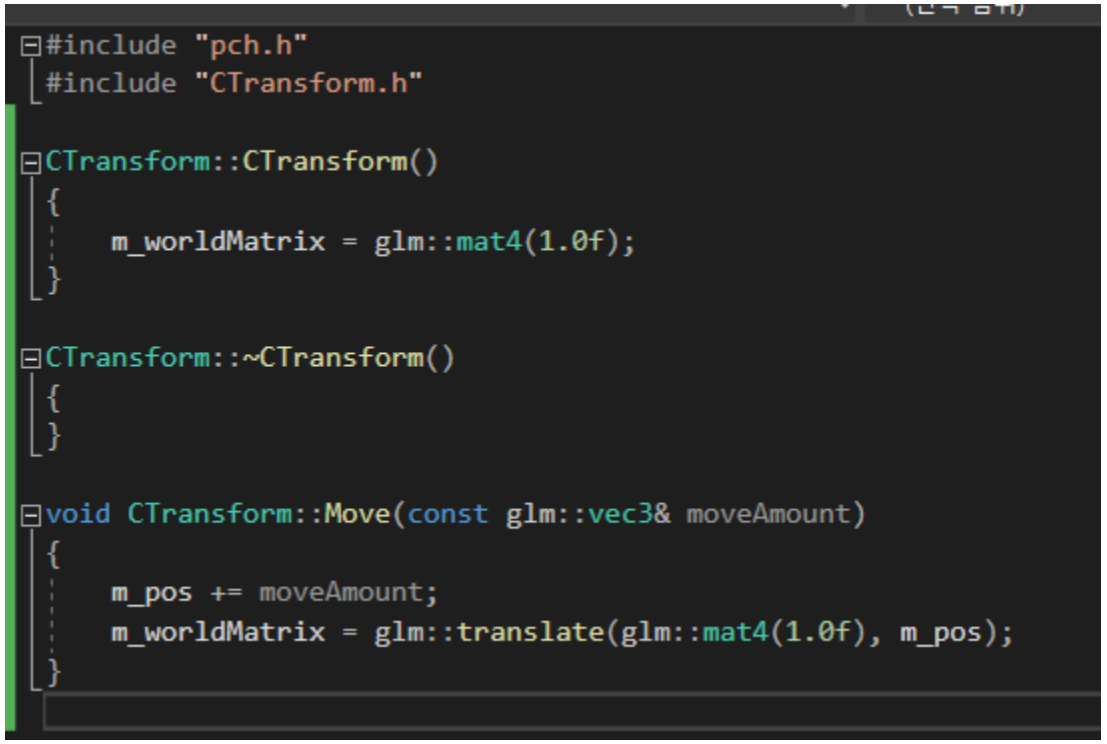

솔루션 '스터디' (1 프로젝트의 1)
스터디
참조
외부 종속성
GLSL
fragment.glsl
vertex.glsl
src
Main
Manager
Mesh
CMesh.cpp
CMesh.h
Object
Transfrom
CTransform.cpp
CTransform.h
CBullet.cpp
CBullet.h
CObject.cpp
CObject.h
CPlayer.cpp
CPlayer.h
Scene
리소스 파일
헤더 파일
MathUtil.h
pch.h

CTransform.h

```cpp
#pragma once
class CTransform
{
public:
    CTransform();
    ~CTransform();

    const glm::vec3& GetPos() { return m_pos; }
    const glm::vec3& GetScale() { return m_scale; }
    const glm::vec3& GetRadian() { return m_rotateRadian; }
    const glm::mat4& GetWorldMatrix() { return m_worldMatrix; }

    void SetPos(const glm::vec3 pos) { m_pos = pos; }
    void SetScale(const glm::vec3 scale) { m_scale = scale; }
    void SetRadian(const glm::vec3 radian) { m_rotateRadian = radian; }

    void Move(const glm::vec3& moveAmount);

private:
    glm::vec3 m_pos;
    glm::vec3 m_scale;
    glm::vec3 m_rotateRadian;
    glm::mat4 m_worldMatrix;
};
```


```cpp
#pragma once
class CTransform
{
public:
CTransform();
~CTransform();

const glm::vec3& GetPos() { return m_pos; }
const glm::vec3& GetScale() { return m_scale; }
const glm::vec3& GetRadian() { return m_rotateRadian; }
const glm::mat4& GetWorldMatrix() { return m_worldMatrix; }

void SetPos(const glm::vec3 pos) { m_pos = pos; }
void SetScale(const glm::vec3 scale) { m_scale = scale; }
void SetRadian(const glm::vec3 radian) { m_rotateRadian = radian; }

void Move(const glm::vec3& moveAmount);

private:
glm::vec3 m_pos;
glm::vec3 m_scale;
glm::vec3 m_rotateRadian;
glm::mat4 m_worldMatrix;
};
```

CTransform.cpp



```cpp
#include "pch.h"
#include "CTransform.h"

CTransform::CTransform()
{
    m_worldMatrix = glm::mat4(1.0f);
}

CTransform::~CTransform()
{
}

void CTransform::Move(const glm::vec3& moveAmount)
{
    m_pos += moveAmount;
    m_worldMatrix = glm::translate(glm::mat4(1.0f), m_pos);
}
```


```cpp
#include "pch.h"
#include "CTransform.h"

CTransform::CTransform()
{
m_worldMatrix = glm::mat4(1.0f);
}

CTransform::~CTransform()
{
}

void CTransform::Move(const glm::vec3& moveAmount)
{
m_pos += moveAmount;
m_worldMatrix = glm::translate(glm::mat4(1.0f), m_pos);
}
```

Cobject.h 수정

```
#pragma once
#include "CMesh.h"
#include "CTransform.h"

class CObject
{
public:
    CObject();
    virtual ~CObject();

    virtual bool Update(float elapsedTime);
    virtual void Render();

    void SetDir(const Vector2& dir);

protected:
    CTransform* m_transform;
    Vector2 m_dir;  //방향
    float m_speed;

    shared_ptr<MeshInfo> m_mesh;
};
```

```
#pragma once
#include "CMesh.h"
#include "CTransform.h"

class CObject
{
public:
CObject();
virtual ~CObject();

virtual bool Update(float elapsedTime);
virtual void Render();

void SetDir(const Vector2& dir);

protected:
CTransform* m_transform;
Vector2 m_dir;//방향
float m_speed;

shared_ptr<MeshInfo> m_mesh;
};
```

CObject.cpp

```cpp
#include "pch.h"
#include "CObject.h"
#include "CShaderManager.h"

constexpr float PLAYER_SPEED = 1.f;

CObject::CObject()
{
    m_transform = new CTransform();
}

CObject::~CObject()
{
    for (auto& vbo : m_mesh->vbos) {
        glDeleteBuffers(1, &vbo);
    }

    delete m_transform;
}

bool CObject::Update(float elapsedTime)
{
    Vector2 moveAmount = m_dir * elapsedTime * m_speed;
    m_transform->Move(glm::vec3(moveAmount.x, moveAmount.y, 0.0f));
    return true;
}

void CObject::Render()
{
    GLuint program = CShaderManager::GetInstance()->UseShader(SHADER_TYPE::DEFAULT);

    glUniformMatrix4fv(glGetUniformLocation(program, "modelTransform"), 1, GL_FALSE, glm::value_ptr(m_transform->GetWorldMatrix()));

    for (int i = 0; i < m_mesh->vbos.size(); ++i) {
        glEnableVertexAttribArray(i);
        glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[i]);
        glVertexAttribPointer(i, 3, GL_FLOAT, GL_FALSE, 0, 0);
    }

    glDrawArrays(GL_TRIANGLES, 0, m_mesh->vertex.size());

    for (int i = 0; i < m_mesh->vbos.size(); ++i) {
        glDisableVertexAttribArray(i);
    }
}
```

SetDir은 수정사항이 없어서 그대로 유지

```cpp
#include "pch.h"
#include "CObject.h"
#include "CShaderManager.h"

constexpr float PLAYER_SPEED = 1.f;

CObject::CObject()
{
m_transform = new CTransform();
}

CObject::~CObject()
{
for (auto& vbo : m_mesh->vbos) {
glDeleteBuffers(1, &vbo);
}

delete m_transform;
}

bool CObject::Update(float elapsedTime)
{
Vector2 moveAmount = m_dir * elapsedTime * m_speed;
m_transform->Move(glm::vec3(moveAmount.x, moveAmount.y, 0.0f));
return true;
}

void CObject::Render()
{
GLuint program = CShaderManager::GetInstance()->UseShader(SHADER_TYPE::DEFAULT);

glUniformMatrix4fv(glGetUniformLocation(program, "modelTransform"), 1, GL_FALSE, glm::value_ptr(m_transform->GetWorldMatrix()));

for (int i = 0; i < m_mesh->vbos.size(); ++i) {
glEnableVertexAttribArray(i);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[i]);
glVertexAttribPointer(i, 3, GL_FLOAT, GL_FALSE, 0, 0);
}

glDrawArrays(GL_TRIANGLES, 0, m_mesh->vertex.size());

for (int i = 0; i < m_mesh->vbos.size(); ++i) {
glDisableVertexAttribArray(i);
}
}
```

CPlayer.cpp

```cpp
CPlayer::CPlayer()
{
    m_transform->SetPos(glm::vec3(0.5f, 0.5f, 0.0f));
    m_speed = PLAYER_SPEED;

    m_mesh = CMesh::GetMeshInfo("cube.obj", glm::vec3(1.0f, 0.0f, 0.0f));

    glGenBuffers(1, &m_mesh->vbos[0]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->vertex.size(), m_mesh->vertex.data(), GL_STATIC_DRAW);

    glGenBuffers(1, &m_mesh->vbos[1]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->color.size(), m_mesh->color.data(), GL_STATIC_DRAW);

    glGenBuffers(1, &m_mesh->vbos[2]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[2]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->normals.size(), m_mesh->normals.data(), GL_STATIC_DRAW);

    glGenBuffers(1, &m_mesh->vbos[3]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[3]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec2) * m_mesh->uvs.size(), m_mesh->uvs.data(), GL_STATIC_DRAW);
}
```

```cpp
bool CPlayer::Update(float elapsedTime)
{
    DWORD keyInput = CKeyInput::GetInstance()->GetKeyInput();

    if (keyInput) {
        if (keyInput & DIR_FORWARD)
            SetDir({ 0,1 });
        if (keyInput & DIR_BACKWARD)
            SetDir({ 0,-1 });
        if (keyInput & DIR_LEFT)
            SetDir({ -1,0 });
        if (keyInput & DIR_RIGHT)
            SetDir({ 1,0 });
        if (keyInput & FIRE_BULLET) {
            if (high_resolution_clock::now() - m_lastFireTime >= milliseconds(BULLET_COOLTIME)) {
                CSceneManager::GetInstance()->AddUpdateObj(OBJ_TYPE::BULLET, new CBullet(m_transform->GetPos(), m_dir));
                m_lastFireTime = high_resolution_clock::now();
            }
        }

    }
    //m_prevPos = m_pos;

    CObject::Update(elapsedTime);

    //if (!CCollisionManager::GetInstace()->CheckValidPos(m_pos, m_size)) {
    //  m_pos = m_prevPos;
    //}

    CSceneManager::GetInstance()->AddRenderObj(OBJ_TYPE::PLAYER, this);
    m_dir.x = m_dir.y = 0.f;

    return true;
}
```

```
CPlayer::CPlayer()
{
m_transform->SetPos(glm::vec3(0.5f, 0.5f, 0.0f));
m_speed = PLAYER_SPEED;

m_mesh = CMesh::GetMeshInfo("cube.obj", glm::vec3(1.0f, 0.0f, 0.0f));

glGenBuffers(1, &m_mesh->vbos[0]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->vertex.size(), m_mesh->vertex.data(), GL_STATIC_DRAW);

glGenBuffers(1, &m_mesh->vbos[1]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[1]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->color.size(), m_mesh->color.data(), GL_STATIC_DRAW);

glGenBuffers(1, &m_mesh->vbos[2]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[2]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->normals.size(), m_mesh->normals.data(), GL_STATIC_DRAW);

glGenBuffers(1, &m_mesh->vbos[3]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[3]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec2) * m_mesh->uvs.size(), m_mesh->uvs.data(), GL_STATIC_DRAW);
}
```

```
bool CPlayer::Update(float elapsedTime)
{
DWORD keyInput = CKeyInput::GetInstance()->GetKeyInput();

if (keyInput) {
if (keyInput & DIR_FORWARD)
SetDir({ 0,1 });
if (keyInput & DIR_BACKWARD)
SetDir({ 0,-1 });
if (keyInput & DIR_LEFT)
SetDir({ -1,0 });
if (keyInput & DIR_RIGHT)
SetDir({ 1,0 });
if (keyInput & FIRE_BULLET) {
if (high_resolution_clock::now() - m_lastFireTime >= milliseconds(BULLET_COOLTIME)) {
CSceneManager::GetInstance()->AddUpdateObj(OBJ_TYPE::BULLET, new CBullet(m_transform->GetPos(), m_dir));
m_lastFireTime = high_resolution_clock::now();
}
}

}
//m_prevPos = m_pos;

CObject::Update(elapsedTime);

//if (!CCollisionManager::GetInstace()->CheckValidPos(m_pos, m_size)) {
//m_pos = m_prevPos;
//}

CSceneManager::GetInstance()->AddRenderObj(OBJ_TYPE::PLAYER, this);
m_dir.x = m_dir.y = 0.f;

return true;
}
```

Cbullet.cpp

```
CBullet::CBullet(const glm::vec3& pos, const Vector2& dir)
{
    m_transform->SetPos(pos);
    m_dir = dir;
    m_speed = BULLET_SPEED;

    m_mesh = CMesh::GetMeshInfo("cube.obj", glm::vec3(1.0f, 0.0f, 0.0f));

    glGenBuffers(1, &m_mesh->vbos[0]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->vertex.size(), m_mesh->vertex.data(), GL_STATIC_DRAW);

    glGenBuffers(1, &m_mesh->vbos[1]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->color.size(), m_mesh->color.data(), GL_STATIC_DRAW);

    glGenBuffers(1, &m_mesh->vbos[2]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[2]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->normals.size(), m_mesh->normals.data(), GL_STATIC_DRAW);

    glGenBuffers(1, &m_mesh->vbos[3]);
    glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[3]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec2) * m_mesh->uvs.size(), m_mesh->uvs.data(), GL_STATIC_DRAW);
}
```

```cpp
bool CBullet::Update(float elapsedTime)
{
    CObject::Update(elapsedTime);
    //if (!CCollisionManager::GetInstace()->CheckValidPos(m_pos, m_size)) {
    //  delete this;
    //  return false;
    //}
    //else {
        CSceneManager::GetInstance()->AddRenderObj(OBJ_TYPE::BULLET, this);
        return true;
    //}
}
```

```
CBullet::CBullet(const glm::vec3& pos, const Vector2& dir)
{
m_transform->SetPos(pos);
m_dir = dir;
m_speed = BULLET_SPEED;

m_mesh = CMesh::GetMeshInfo("cube.obj", glm::vec3(1.0f, 0.0f, 0.0f));

glGenBuffers(1, &m_mesh->vbos[0]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->vertex.size(), m_mesh->vertex.data(),
GL_STATIC_DRAW);

glGenBuffers(1, &m_mesh->vbos[1]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[1]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->color.size(), m_mesh->color.data(), GL_STATIC_DRAW);

glGenBuffers(1, &m_mesh->vbos[2]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[2]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * m_mesh->normals.size(), m_mesh->normals.data(),
GL_STATIC_DRAW);

glGenBuffers(1, &m_mesh->vbos[3]);
glBindBuffer(GL_ARRAY_BUFFER, m_mesh->vbos[3]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec2) * m_mesh->uvs.size(), m_mesh->uvs.data(), GL_STATIC_DRAW);
}
```

```
bool CBullet::Update(float elapsedTime)
{
CObject::Update(elapsedTime);
//if (!CCollisionManager::GetInstace()->CheckValidPos(m_pos, m_size)) {
//delete this;
//return false;
//}
//else {
CSceneManager::GetInstance()->AddRenderObj(OBJ_TYPE::BULLET, this);
return true;
//}
}
```

- **CPlayer와 Cbullet은생성자와 Update함수 외에 수정사항 없음**

클래스를 분할하는 이유

- Ctransform을 만들지 않고 해당 정보를 Cobject클래스에 넣어도 되는데 왜 나눌까
- 이전에도 말했듯이 유지보수를 위해
- 객체지향프로그래밍 설계 원칙 SOLID에 대해서 한번 알아보자
- SOLID 설계원칙 → 프레임워크를 만드는 이유