포트폴리오

원휘재

Mail : id8440540@naver.com
Phone : 010 6431 4396

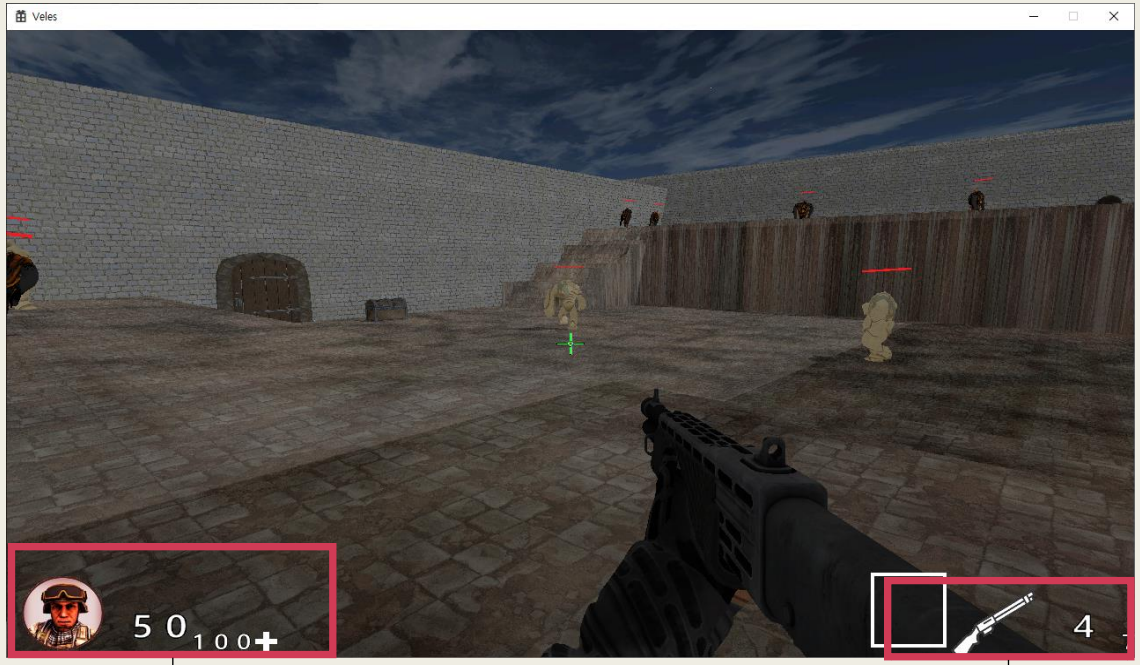
목차

1. Veles DirectX12 졸업작품
2. Cannon Defence OpenGL 컴퓨터그래픽스 텀 프로젝트
3. 3D 슈팅 게임 DirectX12 3D게임프로그래밍 텀 프로젝트

1.Veles

- 게임소개 : 어드벤쳐 던전 온라인 FPS게임
- 플레이인원 : 4인
- 개발 환경 : DirectX 12, IOCP
- 개발 기간 : 2020.12.26 ~ 2021.09.03
- 개발 인원 : 3인(2 client, 1 server)
- 역할 : 메인 클라이언트
- 영상 : https://youtu.be/qBWcUVGxcPI
- Github : https://github.com/dnjsgnlwo/BOBY

Veles 소개



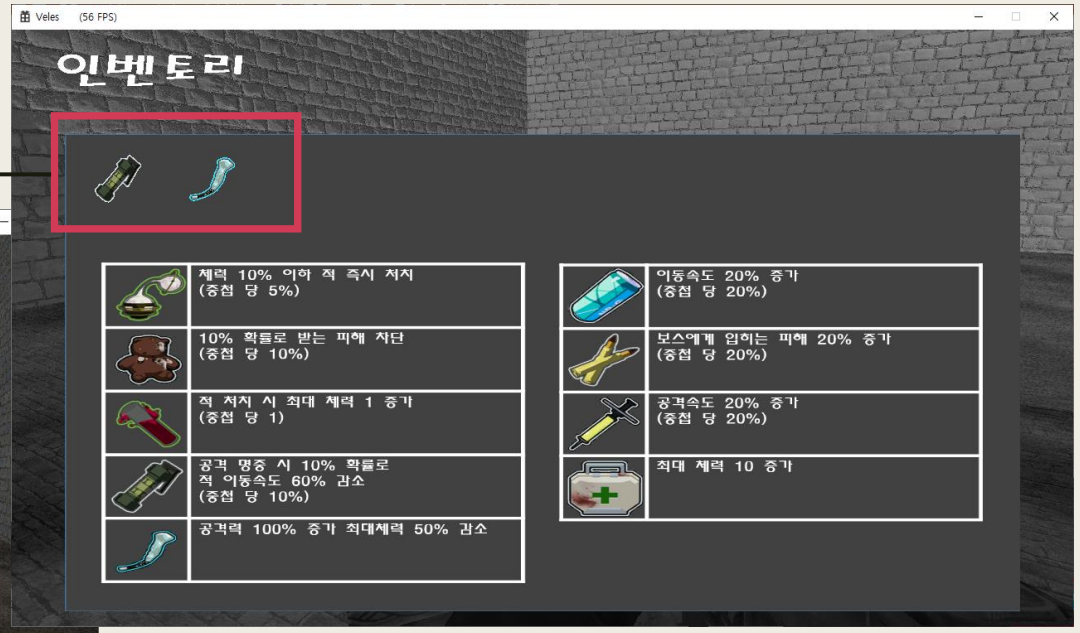
플레이어 STATUS

스테이지 보스와 HP바

플레이어 목표: 던전 각각의 방에 위치한 몬스터나 퍼즐을 해결하고 아이템을 획득해 각 스테이지 최종 방의 보스를 처치하면 된다.

Veles 소개

아이템 오브젝트



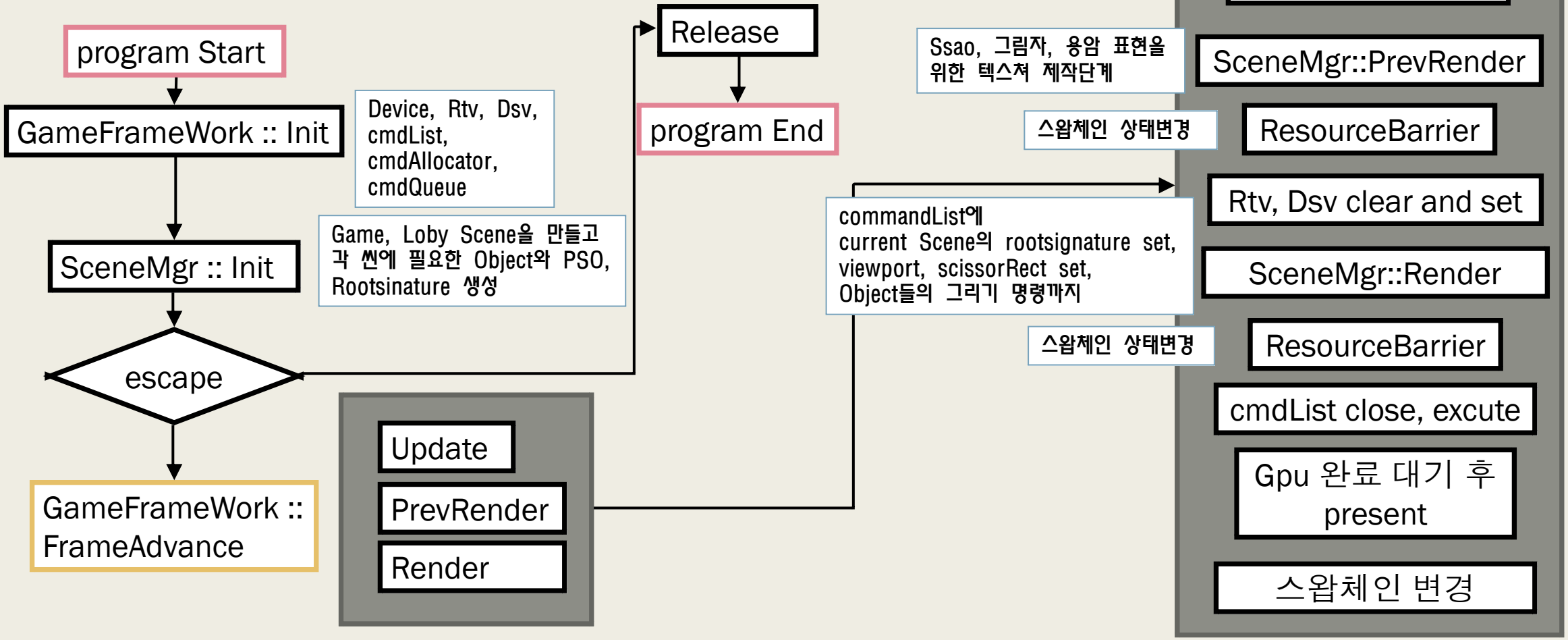
방 클리어시 열 수 있는 상자 오브젝트를 통해 아이템을 획득,
아이템 획득과 누적을 통한 캐릭터의 스펙업이 가능하다.

Veles 구현기술

- DirectX12 기반 렌더러 개발
- 그래픽 개선 (SSAO, 그림자)
- Fbx 모델 로드 with Assimp
- 셰이더(이펙트, 용암, 안개, 빌보드, UI)
- 서버에서의 충돌처리

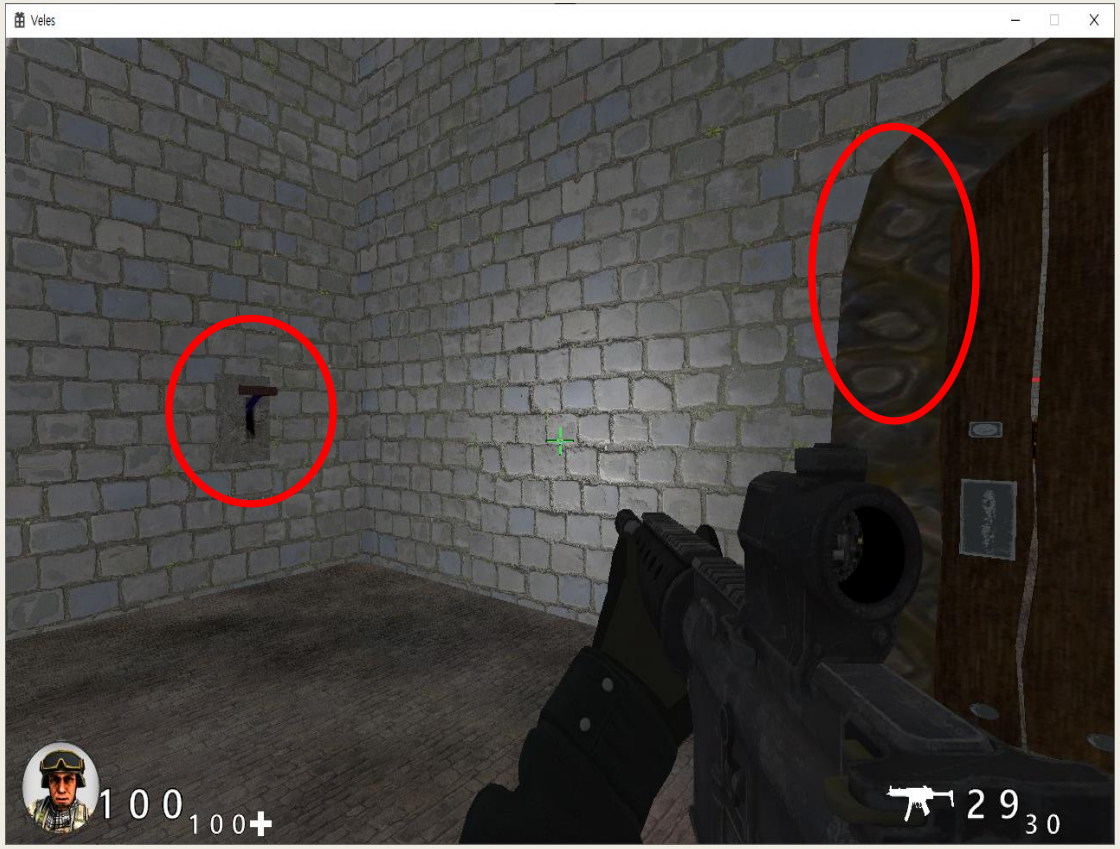
Veles 구현기술

- DirectX12 기반 렌더러 개발



Veles 구현기술

- Ssao

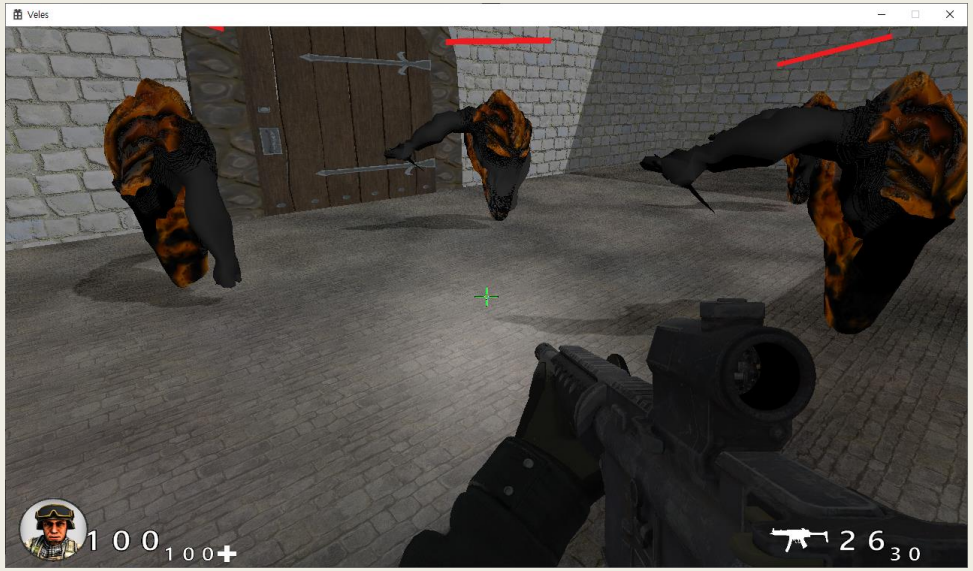


처리 전

처리 후

Veles 구현기술

- Shadow



DirectionalLight와 플레이어의 SpotLight에 대한 그림자 구현

엘리어싱 문제 해결을 위해 PCF를 이용해 그림자 생성

조명의 위치에서 생성한 Depth 텍스쳐

```
float CalculateShadowFactor(float3 pos, int index)
{
    float4 shadowPosition;
    float depth;
    uint width, height, numMips;
    float percentLit = 0.0f, fBias = 0.005f;

    percentLit = 0.0f;
    shadowPosition = mul(mul(mul(float4(pos, 1.0f), shadowViews[index]), shadowProjs[index]), gmtxProjectToTexture);
    shadowPosition.xyz /= shadowPosition.w;
    depth = shadowPosition.z;
    gtxtShadowMap[index].GetDimensions(0, width, height, numMips);

    float dx = 1.0f / (float) width;
    const float2 offsets[9] =
    {
        float2(-dx, -dx), float2(0.0f, -dx), float2(dx, -dx),
        float2(-dx, 0.0f), float2(0.0f, 0.0f), float2(dx, 0.0f),
        float2(-dx, +dx), float2(0.0f, +dx), float2(dx, +dx)
    };
    for (int i = 0; i < 9; i++)
    {
        float fsDepth = gtxtShadowMap[index].SampleCmpLevelZero(gShadowSamplerState, shadowPosition.xy + offsets[i], depth).r;
        if (shadowPosition.z - fBias < fsDepth)
            percentLit += 1.0f;
    }
    percentLit /= 9;

    return percentLit;
}
```

Veles 구현기술

- Fbx모델 로드 (Assimp)

```
string materialName;
if (node->mNumMeshes != 0)
{
    pTexture = new CTexture(node->mNumMeshes * 2, RESOURCE_TEXTURE2D_ARRAY, 0, 1);
}
for (unsigned int i = 0; i < node->mNumMeshes; i++)
{
    aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];

    if (mesh->mMaterialIndex >= 0)
    {
        materialName = scene->mMaterials[mesh->mMaterialIndex]->GetName().C_Str();
        std::string s = materialName + "Wn";
        OutputDebugStringA(s.c_str());
        TEXTURE_INFO texInfo = textures.find(materialName)->second;
        pTexture->LoadTextureFromFile(pd3dDevice, pd3dCommandList, texInfo.diffuse, RESOURCE_TEXTURE2D, i * 2);
        pTexture->LoadTextureFromFile(pd3dDevice, pd3dCommandList, texInfo.normal, RESOURCE_TEXTURE2D, i * 2 + 1);
    }


    if (mesh->HasBones())
    {
        for (UINT j = 0; j < mesh->mNumBones; j++) { ... }
        if (isSkin)
        {
            aiMatrix4x4 aitrs = node->mTransformation;
            pFbxModelObject->m_Transform = Matrix4x4::identity;
        }
    }
    CFbxHierarchyMesh* fbxmesh = new CFbxHierarchyMesh(pd3dDevice, pd3dCommandList, mesh, scene, i, boneDataMap);
    pFbxModelObject->SetMesh(i, fbxmesh);
}
```

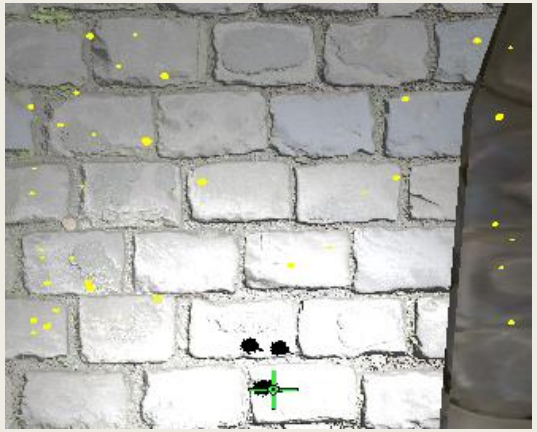
Assimp를 사용한 fbx모델 로드 계층구조로 fbx에서 필요한 정보만 저장

계층에 메시가 있을경우 fbx로부터 meshdata를 읽어옴

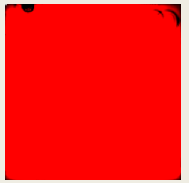
```
MeshData CFbxHierarchyMesh::processMesh(aiMesh* mesh, const aiScene* scene, UINT materialnum, std::map<s
{
    MeshData meshdata;
    std::string s = mesh->mName.C_Str();
    s += " ";
    //fbx에서 인덱스정보 저장
    for (UINT i = 0; i < mesh->mNumFaces; ++i)
    {
        for (UINT j = 0; j < mesh->mFaces[i].mNumIndices; ++j)
        {
            meshdata.indices.emplace_back(mesh->mFaces[i].mIndices[j]);
        }
    }
    meshdata.vertices.reserve(mesh->mNumVertices);
    for (UINT i = 0; i < mesh->mNumVertices; ++i)
    {
        VertexInfo vi;
        vi.position = { mesh->mVertices[i].x,mesh->mVertices[i].y, mesh->mVertices[i].z };
        vi.normal = { mesh->mNormals[i].x, mesh->mNormals[i].y, mesh->mNormals[i].z };
        vi.texcoord0 = { mesh->mTextureCoords[0][i].x, mesh->mTextureCoords[0][i].y };
        vi.tangent = { mesh->mTangents[i].x,mesh->mTangents[i].y,mesh->mTangents[i].z };
        vi.bitangent = { mesh->mBitangents[i].x,mesh->mBitangents[i].y,mesh->mBitangents[i].z };
        vi.MaterialNum = materialnum;
        meshdata.vertices.push_back(vi);
    }
    if (mesh->HasBones()) { ... }

    return meshdata;
}
```

Veles 구현기술

- 셰이더



총알 충돌 시
파티클과 데칼



용암 표현을 위해
Displacement map
생성 및 적용한 모습

기하셰이더를 사용해
만든 빌보드 HP바

플레이어 주변에 안개
효과 구현

2.Cannon Defence

- 게임소개 : 포탄 디펜스 게임
- 플레이인원 : 1인
- 개발 환경 : OpenGL
- 개발 기간 : 2019.11.25 ~ 2019.12.16
- 개발 인원 : 2인(2 client)
- 역할 : 게임로직, 객체관리

Cannon Defence 소개

플레이어 목표: 돌진하는 애니그마를 처치하여 최대한 오랜 시간 동안 바리게이트와 성벽을 지켜야한다.

남은 포탄과 체력

플레이 점수

Cannon Defence 구현기술

- 게임오브젝트를 관리하는 ObjectManager

```
class CObjectManager {
private:
    std::vector<CObject*> vector_Objects;
    CCamera& camera;
public:
    CObjectManager(CCamera& cam);

    void Update(glm::vec3 lightPos = glm::vec3{ 0,0,0 }, glm::vec3 lightColor = glm::vec3{ 1,1,1 }, float lightPower = 1000.f);

    void Update(std::vector<glm::vec3> lightPos, std::vector<glm::vec3> lightColor, std::vector<float> lightPower);

    void Draw();

    int GetState();

    void GetKeyboard(unsigned int key);

    void AddObject(CObject* object);

    void DeleteObject(CObject* object);

    void GetMouseMotion(int x, int y);

    void GetMouse(int button, int state, int x, int y);

    void DeleteAll();

    std::vector<CObject*> GetObjects();

    bool IsCollide(std::vector<float>& object, std::vector<float>& other);

    void ChangeFov(glm::mat4 proj);

    std::vector<glm::vec3> CheckCollision(std::vector<CObject*>& objects, std::vector<CObject*>& others);
};
```

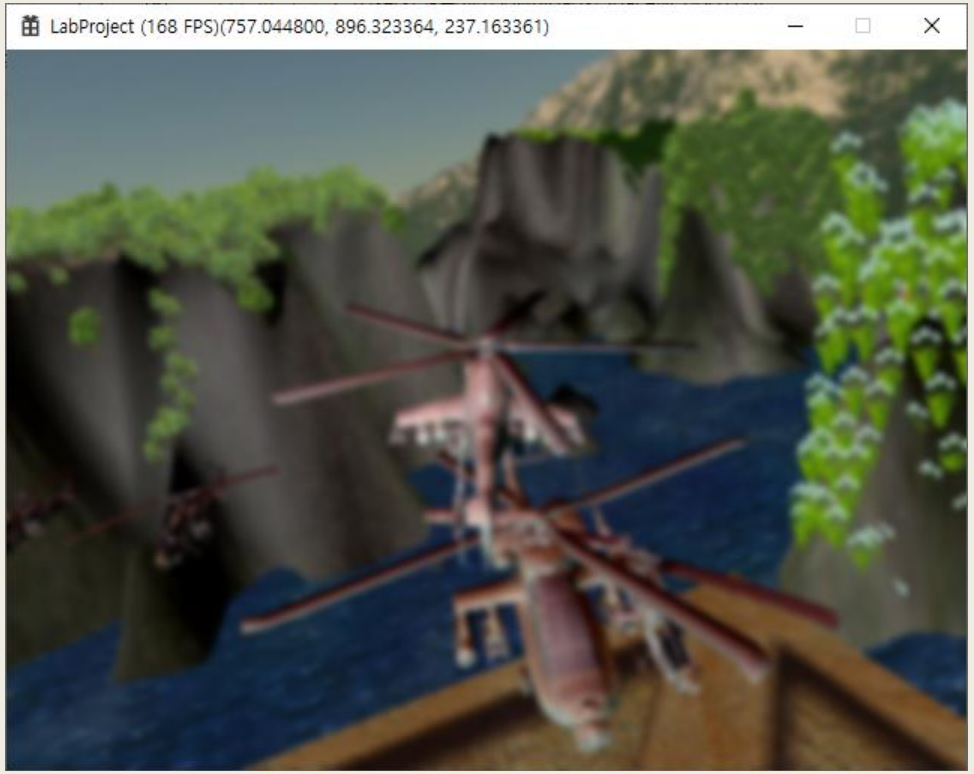
많은 오브젝트를 필요로 하였고
오브젝트들을 한번에 관리하기 위한
매니저를 만듬
같이 작업 하던 2명이
서로 다른 오브젝트들을 만들었고
매니저의 사용은 협업에 도움

3.3D 슈팅 게임

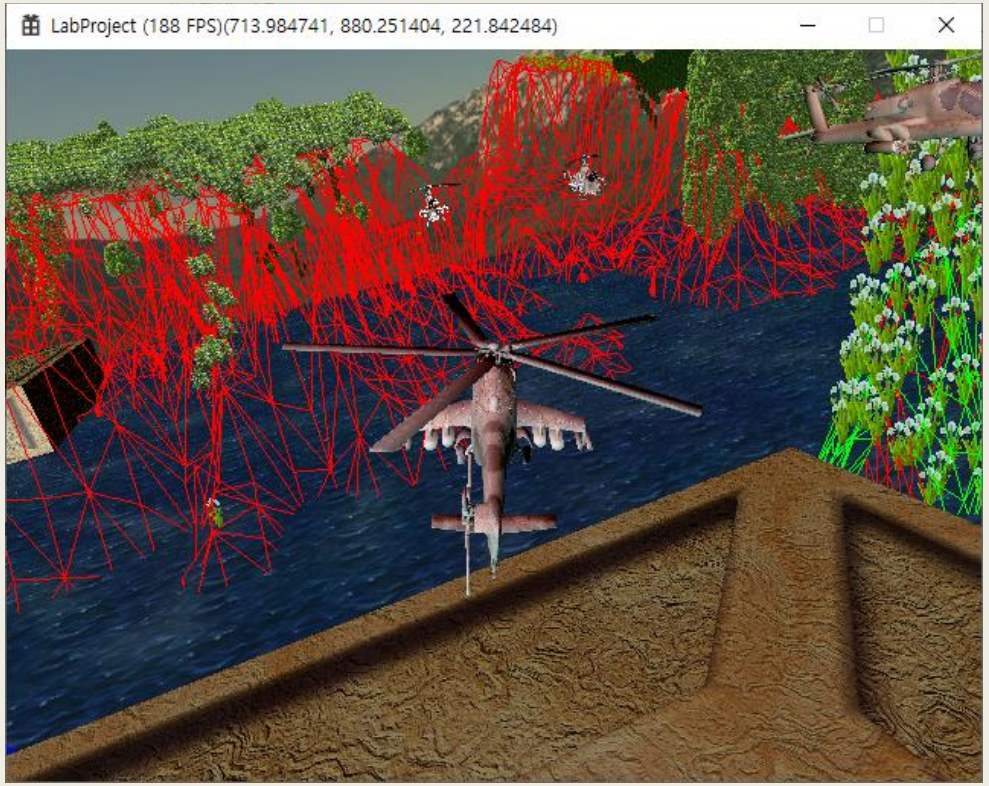
- 게임소개 : 3D 3인칭 슈팅게임
- 플레이인원 : 1인
- 개발 환경 : DirectX 12
- 개발 기간 : 2020.11.01 ~ 2020.12.23
- 개발 인원 : 1인
- 역할 : 클라이언트

3D 슈팅 게임 구현기술

- Blur, Tessellation



충돌시 Blur 효과



지형 Tessellation