클라이언트 포트폴리오

**< 졸업작품 - REVENGER>**





**게임 소개**

* 게임 이름 : REVENGER
* 게임 장르 : 3D FPS 멀티 협동 대전 게임
* 영상 링크 : <https://youtu.be/31BfTQHflNc>
* 프로젝트 소스 : <https://github.com/kiption/XD.git>
* 사용언어 : C++11
* 개발환경 : Visual Studio 2022, DirectX12, FMOD
* 제작 인원 : 3명 (서버 2명 , 클라이언트 1명)
* 역할 : 메인 클라이언트(맵로드, 애니메이션, 그림자, 파티클 등 DirectX12를 이용한
* 게임에 필요한 전반적인 그래픽적인 부분을 구현)
* 제작기간 : 2022/08 ~ 2023/07 (약 1년)
* 게임목표 : 로비에서 군인2명과 헬리콥터1명으로 구성된, 총 3명의 플레이어가 모여 다같이 방에 입장하고, 거리를 활보하는 적들을 정해진 시간내에 소탕하고 전장을 점령하게되면, 승리하는 게임이다.

**세부내용**

1. **전체적인 시스템 구조**

* DirectX12 디바이스를 생성하고, 메모리 할당과 초기화 과정을 거쳐 해당 Scene에 Build될 객체들을 로드한다. Scene은 오프닝 씬과, 스테이지 2가지로 분류하고, 로비 창은 UI로 처리하였음.
* Scene이 생성되고, 해당 Scene에서 사용하는 리소스들을 생성하고 게임시간에 따른 충돌, 상태변화, 애니메이션 등을 Animate 함수로 묶어서 동작이 수행함으로써, 변화되는 부분들을 매 프레임마다 화면에 렌더링되게 한다.

1. **프레임워크 세부 설명**

**1. GameFrameWork**

* Directx12 디바이스, 명령 큐 , 명령 리스트, 스왑체인등 기본자원들을 생성하고, 해당 Scene의 객체들을 빌드하고, Animate한다.

키보드, 마우스 입력처리를 담당하고, 게임타임에 대한 프레임을 관리한다.

**2. Scene**

* 루트 시그니처, 객체의 정보, 쉐이더, 조명 등 해당 장면에 렌더링 되는 것들을 관리한다.

Scene의 구성은 오프닝 장면과 인게임 장면 2가지로 구성된다.

각 Scene 마다 해당되는 객체들의 상호작용과, Animate를 수행하게 된다.

OPENING\_SCENE

INGAME\_SCENE

Scene

**3. Mesh**

* 객체들을 이루는 메쉬정보를 다루고, 노말,색상,텍스쳐 등의 정보 포함여부를 고려하여 작성된다.

Bin 파일로 로드되는 객체의 메쉬정보를 읽고 쓰는 과정을 거친다.

SkyBoxMesh

SkinMesh

Mesh

StandardMesh

Vertex

**4. Object**

* 메쉬의 정보를 바탕으로 렌더링 할 객체의 상태, 속성 등을 다룬다.

Animation을 사용하는 객체일 때는 로딩된 애니메이션 컨트롤러, 애니메이션 트랙에 맞춰 렌더링한다.

Texture를 사용할 경우, dds 파일을 읽어 텍스쳐 리소스를 생성하여 루트 시그니처에 바인딩하고 , 재질정보를 상수버퍼에 저장한다.

Object

Animation

Texture

Mesh

**5. Camera**

* 게임내에서 카메라 역할을 하기위해 필요한 역할을 한다.

FPS 장르이기에, 군인은 1인칭, 헬기는 헬기 내부시점으로 카메라 위치를 조절하여주었다.

**6. Shader**

Shadow

Shader

Skybox

Standard

Billboard

BillboardSprite

**본인이 작성한 각각의 shader 소스코드와 설명 추가**

1. **개발하면서 느낀점**

1. **새로 얻어간 부분들**

* 그림자 매핑을 구현하는 것에서 특히나 많은 어려움을 겪었습니다. 구현이 마무리된 상황에서 느낀점은, 이론적인 지식이 부족했기도 했었고, 구조를 만들고 코드를 작성해야하는 점들을 많이 느끼게 되었습니다.
* 구현이 안되고 있던 가장 큰 문제점 중 하나는 텍스쳐 매핑과 그림자 매핑을 하나의 파이프라인상태에서 사용하고있던 것이었습니다.

해당 객체가 그림자가 적용되는지에 대한 유무를 판단하여 파이프라인 set을 해주었습니다.

**(소스코드 탭간격이 너무 크다. 반으로 줄이자, {}도 가능하면 없애자.)**

ShadowRender

**void GameObjectMgr::ShadowRender(ID3D12GraphicsCommandList\* pd3dCommandList, CCamera\* pCamera, bool bPrerender, ShaderMgr\* pShaderComponent)**

**{**

**if (m\_pSkinnedAnimationController) m\_pSkinnedAnimationController->UpdateShaderVariables(pd3dCommandList);**

**OnPrepareRender();**

**UpdateShaderVariable(pd3dCommandList, &m\_xmf4x4World);**

**if (m\_pMesh)** **{**

**if (m\_nMaterials > 0)**

**{**

**for (int i = 0; i < m\_nMaterials; i++)**

**{**

**if (m\_ppMaterials[i])**

**{**

**if (m\_ppMaterials[i]->m\_pShader)**

**{**

****

**m\_ppMaterials[i]->m\_pShader->Render(pd3dCommandList, pCamera, 0, false);**

**pShaderComponent->SetPipelineState(pd3dCommandList, 0);**

**UpdateShaderVariables(pd3dCommandList);**

**}**

**m\_ppMaterials[i]->UpdateShaderVariable(pd3dCommandList);**

**for (int k = 0; k < m\_ppMaterials[i]->m\_nTextures; k++)**

**{**

**if(m\_ppMaterials[i]->m\_ppTextures[k])m\_ppMaterials[i]->m\_ppTextures[k]->UpdateShaderVariables(pd3dCommandList);**

**}**

**}**

**m\_pMesh->Render(pd3dCommandList, i);**

**}**

**}**

**}**

**if (m\_pSibling) m\_pSibling->ShadowRender(pd3dCommandList, pCamera, bPrerender, pShaderComponent);**

**if (m\_pChild) m\_pChild->ShadowRender(pd3dCommandList, pCamera, bPrerender, pShaderComponent);**

**}**

Effect Render

**void GameObjectMgr::Render(ID3D12GraphicsCommandList\* pd3dCommandList, CCamera\* pCamera, bool bPrerender)**

**{**

**if (m\_pSkinnedAnimationController) m\_pSkinnedAnimationController->UpdateShaderVariables(pd3dCommandList);**

**OnPrepareRender();**

**UpdateShaderVariable(pd3dCommandList, &m\_xmf4x4World);**

**if (m\_pMesh)**

**{**

**if (m\_nMaterials > 0)**

**{**

**for (int i = 0; i < m\_nMaterials; i++)**

**{**

**if (m\_ppMaterials[i])**

**{**

**if (m\_ppMaterials[i]->m\_pShader)**

**{**

**m\_ppMaterials[i]->m\_pShader->Render(pd3dCommandList, pCamera, 0, bPrerender);**

**UpdateShaderVariables(pd3dCommandList);**

**}**

**m\_ppMaterials[i]->UpdateShaderVariable(pd3dCommandList);**

**for (int k = 0; k < m\_ppMaterials[i]->m\_nTextures; k++)**

**{**

**if (m\_ppMaterials[i]->m\_ppTextures[k])m\_ppMaterials[i]->m\_ppTextures[k] ->UpdateShaderVariables(pd3dCommandList);**

**}**

**}**

**m\_pMesh->Render(pd3dCommandList, i);**

**}**

**}**

**}**

**if (m\_pSibling) m\_pSibling->Render(pd3dCommandList, pCamera, bPrerender);**

**if (m\_pChild) m\_pChild->Render(pd3dCommandList, pCamera, bPrerender);**

**}**

VSShadowMap

**VS\_SHADOW\_MAP\_OUTPUT VSShadowMapVS\_LIGHTING\_INPUT input)**

**{**

**VS\_SHADOW\_MAP\_OUTPUT output = (VS\_SHADOW\_MAP\_OUTPUT)0;**

**float4 positionW = (float4) 0.0f;**

**if (!bAnimationShader)**

**{**

**positionW = mul(float4(input.position, 1.0f), gmtxGameObject);**

**output.positionW = positionW.xyz;**

**output.position = mul(mul(positionW, gmtxView), gmtxProjection);**

** output.normalW = mul(float4(input.normal, 0.0f), gmtxGameObject).xyz;**

**output.uv = input.uv;**

**output.tangentW = mul(input.tangent, (float3x3) gmtxGameObject);**

**output.bitangentW = mul(input.bitangent, (float3x3) gmtxGameObject);**

**}**

**if (bAnimationShader)**

**{**

**float4x4 mtxVertexToBoneWorld = (float4x4) 0.0f;**

**for (int i = 0; i < MAX\_VERTEX\_INFLUENCES; i++)**

**{**

**mtxVertexToBoneWorld += input.weights[i] \* mul(gpmtxBoneOffsets[input.indices[i]], gpmtxBoneTransforms[input.indices[i]]);**

**}**

**positionW = mul(float4(input.position, 1.0f), mtxVertexToBoneWorld);**

**output.positionW = positionW.xyz;**

**output.normalW = mul(input.normal, (float3x3) mtxVertexToBoneWorld).xyz;**

**output.tangentW = mul(input.tangent, (float3x3) mtxVertexToBoneWorld).xyz;**

**output.bitangentW = mul(input.bitangent, (float3x3) mtxVertexToBoneWorld).xyz;**

**output.position = mul(mul(float4(output.positionW, 1.0f), gmtxView), gmtxProjection);**

**output.uv = input.uv;**

**}**

**for (int i = 0; i < MAX\_LIGHTS; i++)**

**{**

**if (gcbToLightSpaces[i].f4Position.w != 0.0f)**

**output.uvs[i] = mul(positionW, gcbToLightSpaces[i].mtxToTexture);**

**}**

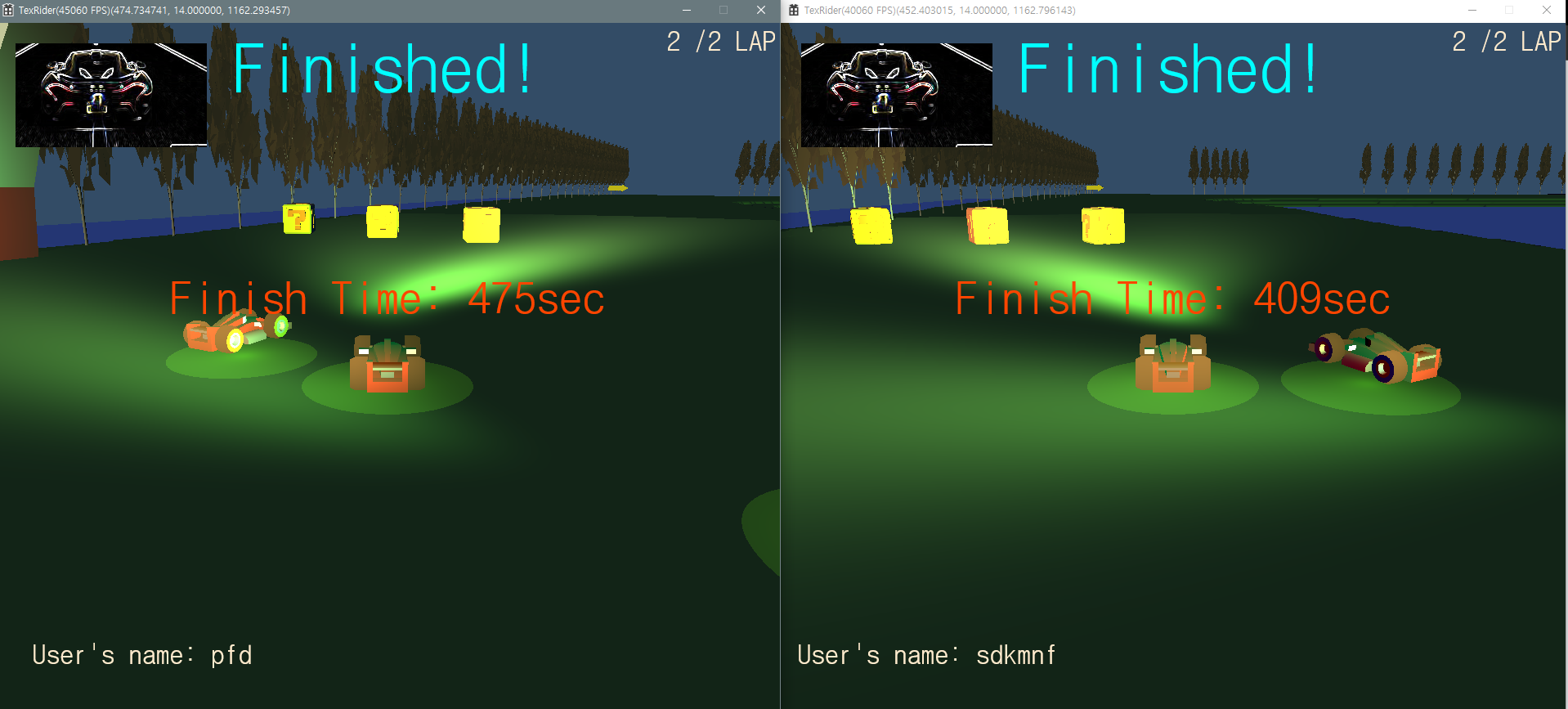
**return(output);**

**}**

* 서버와 클라이언트간의 소통도 처음에는 쉽지않았지만, 작은 단계부터 주고 받다 보니 복잡한 정보도 주고 받을때 어려움이 점차 없어졌고, 서버 개발자에게 “이렇게 전달해주어야겠다.” 라는 생각이 생겨서 소통이 원활하게 될 수 있었던 것 같습니다. **[좀더 문장을 다듬어 보자]**

**< 네트워크 게임프로그래밍 텀프로젝트 >**

****

****

**게임 소개**

* 게임 장르 : 3D 레이싱 멀티게임
* 영상 링크 : <https://www.youtube.com/watch?v=GooATFiHUyA>
* 프로젝트 소스 : <https://github.com/kiption/Network_Game_Programming.git>
* 사용언어 : C++11
* 개발환경 : Visual Studio 2022, DirectX12, DirecX11
* 제작 인원 : 3명
* 역할 : 전반적인 클라이언트 제작,로그인 패킷 송수신,충돌판단 여부와 충돌시 연출 동기화를 담당.
* 제작기간 : 2022/10 ~ 2022/12
* 게임줄거리 : 3명이 출발선에서 출발하여, 아이템박스를 통해 아이템을 획득한다.

해당아이템을 사용하여 상대방의 완주를 방해하면서 먼저 3바퀴를 완주하면 승리하는 게임이다.

**제작과정**

1. **전체 클라이언트 제작**
2. **로그인 패킷 송수신**
3. 클라이언트 실행
4. 서버 IP 입력
5. 클라이언트의 Connect - 서버의 Accept
6. 등록된 계정 여부에 따른 응답처리
7. 계정이 존재 할 경우 계정 이름을 입력 받아 로그인 패킷 전송
8. 계정이 존재 하지 않을 경우 등록할 계정을 묻기

새로 만든 계정이 등록이되면 유저 List에 등록후 클라이언트에게 등록성공 패킷을 전송

1. **충돌판단 여부 판단**

* 서버에서 매프레임마다 연산되는 플레이어와 다른 플레이어의 Vector 와 Position 정보를 사용해 충돌 처리를 하고 그 판단여부를 클라이언트에 broadcasting 한다.

**< Humornoid\_War >**

**게임 소개**

* 게임 장르 : 3D TPS 싱글 슈팅 게임
* 영상 링크 : <https://youtu.be/d32hA8sPGP0>
* 프로젝트 소스 : <https://github.com/pj002321/Humanoid_war.git>
* 사용언어 : C#
* 개발환경 : Visual Studio 2022, Unity engine
* 제작 인원 : 1명
* 제작 내용 : 한학기 동안 배운 Unity 과제 제작물 (애니메이션 블렌딩, 맵 제작, 물리엔진 적용)
* 제작기간 : 2022/11 ~ 2022/12

**<스샷추가>**

**< Paragon 모작 – 싱글플레이 >**

****

**게임 소개**

* 게임 장르 : 3D TPS 액션 대전 게임
* 영상 링크 : LINK 첨부
* 사용언어 : 블루프린트
* 개발환경 : Unreal engine 5
* 개발인원 : 3명
* 역할 : 애니메이션, AI 배치 및 모션, 공격 및 피격/사망 처리, 이펙트
* 제작기간 : 2023/06 ~ 2023/09
* 게임줄거리 : 사원속에서 배치되어있는 적과 싸우고, 적 뿐만 아니라 다른 플레이어들과 데스매치를 하여서 혼자만이 살아남게 되면 이기게된다.

캐릭터는 Paragon에서 제공하는 Kwang, Muriel, Lt. Belica 캐릭을 사용하게 된다.

**< 그 외 >**

* **다렉 과제나 쉐이더 프로그래밍은 노션으로 뺸다.**