3D게임프로그래밍2(01)

과제1 설명

제출일: 2023.11.05

전공: 게임공학과

학번: 2019184020

성명: 윤은지

목차

1. 조작법(키)에 대한 설명
   1) 플레이어
   2) 적 헬리콥터

2. 모델+터레인 프로젝트에 빌보드 프로젝트 통합

3. 모델+터레인+빌보드 프로젝트에 물 프로젝트 통합

4. 충돌처리

5. 적 헬리콥터의 이동

6. 플레이어 가속

7. 폭발 애니메이션과 점수판

8. 실행 화면

1. 조작법(키)에 대한 설명

1)플레이어

```
462  void CGameFramework::ProcessInput()
463  {
464      static UCHAR pKeysBuffer[256];
465      bool bProcessedByScene = false;
466      if (GetKeyboardState(pKeysBuffer) && m_pScene) bProcessedByScene = m_pScene->ProcessInput(pKeysBuffer);
467      if (!bProcessedByScene)
468      {
469          DWORD dwDirection{};
470          if (pKeysBuffer[0x57] & 0xF0) dwDirection |= DIR_FORWARD;//W
471          if (pKeysBuffer[0x53] & 0xF0) dwDirection |= DIR_BACKWARD;//S
472          if (pKeysBuffer[0x41] & 0xF0) dwDirection |= DIR_LEFT;//A
473          if (pKeysBuffer[0x44] & 0xF0) dwDirection |= DIR_RIGHT;//D
474          if (pKeysBuffer[0x58] & 0xF0) dwDirection |= DIR_UP;//X
475          if (pKeysBuffer[0x43] & 0xF0) dwDirection |= DIR_DOWN;//C
476          if (pKeysBuffer[0x10] & 0xF0 && dwDirection) dwDirection |= DIR_RUN;// Shift run
```

w : 앞으로 이동

s : 뒤로 이동

a : 왼쪽으로 이동

d : 오른쪽으로 이동

x : 위로 이동

c : 아래로 이동

SHIFT : 가속

```
287  void CGameFramework::OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM lParam)
288  {
289      random_device rd;
290      mt19937 gen(rd());
291      uniform_int_distribution<> dist(3, 5);
292
293      if (m_pScene) m_pScene->OnProcessingMouseMessage(hWnd, nMessageID, wParam, lParam);
294      switch (nMessageID)
295      {
296      case WM_LBUTTONDOWN:
297          ::SetCapture(hWnd);
298          ::GetCursorPos(&m_ptOldCursorPos);
299          break;
300      case WM_RBUTTONDOWN:
301
302          m_pScene->pMultiSpriteObjectShader->m_ppObjects[0]->m_ppMaterials[0]->m_pTexture->m_bActive = true;
303          m_pPlayer->attack = true;
304
305
306          m_pScene->pMultiSpriteObjectShader->score = dist(gen);
307          cout << m_pScene->pMultiSpriteObjectShader->score << endl;
308
309          break;
```

마우스 우클릭 : 공격

2)적 헬리콥터

```
392  bool CScene::OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM lParam)
393  {
394      switch (nMessageID)
395      {
396      case WM_KEYDOWN:
397          switch (wParam)
398          {
399          case VK_UP:
400
401              for (int i{}; i < pObjectsShader->m_nObjects; ++i)
402                  pObjectsShader->m_ppObjects[i]->MoveForward(+1.0f);
403              break;
404          case VK_DOWN:
405
406              for (int i{}; i < pObjectsShader->m_nObjects; ++i)
407                  pObjectsShader->m_ppObjects[i]->MoveForward(-1.0f);
408              break;
409          case VK_LEFT:
410
411              for (int i{}; i < pObjectsShader->m_nObjects; ++i)
412                  pObjectsShader->m_ppObjects[i]->MoveStrafe(-1.0f);
413              break;
414          case VK_RIGHT:
415
416              for (int i{}; i < pObjectsShader->m_nObjects; ++i)
417                  pObjectsShader->m_ppObjects[i]->MoveStrafe(+1.0f);
418              break;
419          case VK_RETURN:
420
421              for (int i{}; i < pObjectsShader->m_nObjects; ++i)
422                  pObjectsShader->m_ppObjects[i]->MoveUp(+1.0f);
423              break;
424          case 0x10://SHIFT
425
426              for (int i{}; i < pObjectsShader->m_nObjects; ++i)
427                  pObjectsShader->m_ppObjects[i]->MoveUp(-1.0f);
428              break;
```

UP : 앞으로 이동

DOWN : 뒤로 이동

LEFT : 왼쪽으로 이동

RIGHT : 오른쪽으로 이동

ENTER : 위로 이동

SHIFT : 아래로 이동

```
firstAssignment                                                    CGameFramework
327  void CGameFramework::OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM lParam)
328  {
329      if (m_pScene) m_pScene->OnProcessingKeyboardMessage(hWnd, nMessageID, wParam, lParam);
330      switch (nMessageID)
331      {
332      case WM_KEYUP:
333          switch (wParam)
334          {
335          case VK_ESCAPE:
336              if (onFullScreen)
337                  ChangeSwapChainState();
338
339              ::PostQuitMessage(0);
340              break;
341          case VK_RETURN:
342              break;
343          case VK_F1:
344          case VK_F2:
345          case VK_F3:
346              m_pCamera = m_pPlayer->ChangeCamera((DWORD)(wParam - VK_F1 + 1), m_GameTimer.GetTimeElapsed());
347              break;
348          case VK_CONTROL:
349              if (onFullScreen)
350                  onFullScreen = false;
351              else
352                  onFullScreen = true;
353              ChangeSwapChainState();
```

ESC : 종료

CTRL : 전체 화면으로 전환

또한 Release/x64 모드를 사용하여 프로젝트를 빌드하였습니다.

2. 모델+터레인 프로젝트에 빌보드 프로젝트 통합

Shader.h에 다음 클래스들을 추가하고, Shader.cpp에 다음 클래스들의 구현부를 추가하였습니다.

```
class CBillboardObjectsShader : public CObjectsShader
```

```
class CMultiSpriteObjectsShader : public CObjectsShader
```

```
class CTexturedShader : public CShader
```

Scene.cpp의 다음 함수에서

```
void CScene::BuildObjects(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList)
```

```
pMultiSpriteObjectShader = new CMultiSpriteObjectsShader();
pMultiSpriteObjectShader->CreateShader(pd3dDevice, pd3dCommandList, m_pd3dGraphicsRootSignature);
pMultiSpriteObjectShader->BuildObjects(pd3dDevice, pd3dCommandList, m_pTerrain);
pMultiSpriteObjectShader->SetActive(false);
m_ppShaders[1] = pMultiSpriteObjectShader;

CBillboardObjectsShader* pBillboardObjectShader = new CBillboardObjectsShader();
pBillboardObjectShader->CreateShader(pd3dDevice, pd3dCommandList, m_pd3dGraphicsRootSignature);
pBillboardObjectShader->BuildObjects(pd3dDevice, pd3dCommandList, m_pTerrain);
m_ppShaders[2] = pBillboardObjectShader;
```

폭발 애니메이션을 재생하기 위한 객체와 풀, 나무, 꽃을 출력하기 위한 객체를 빌드했습니다.

다음 함수에

```
ID3D12RootSignature* CScene::CreateGraphicsRootSignature(ID3D12Device* pd3dDevice)
```

이 부분을 추가하였습니다.

```
pd3dDescriptorRanges[9].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[9].NumDescriptors = 1;
pd3dDescriptorRanges[9].BaseShaderRegister = 0; //t0: gtxtTexture
pd3dDescriptorRanges[9].RegisterSpace = 0;
pd3dDescriptorRanges[9].OffsetInDescriptorsFromTableStart = D3D12_DESCRIPTOR_RANGE_OFFSET_APPEND;

pd3dRootParameters[12].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[12].DescriptorTable.NumDescriptorRanges = 1;
pd3dRootParameters[12].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[9];
pd3dRootParameters[12].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
```

Shader.cpp의 다음 함수에서

```
void CBillboardObjectsShader::BuildObjects(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, void* pContext)
```

이 부분을 수정하였습니다.

```
CreateShaderResourceViews(pd3dDevice, ppGrassTextures[0], 0, 12);
CreateShaderResourceViews(pd3dDevice, ppGrassTextures[1], 0, 12);
CreateShaderResourceViews(pd3dDevice, ppFlowerTextures[0], 0, 12);
CreateShaderResourceViews(pd3dDevice, ppFlowerTextures[1], 0, 12);
CreateShaderResourceViews(pd3dDevice, ppTreeTextures[0], 0, 12);
CreateShaderResourceViews(pd3dDevice, ppTreeTextures[1], 0, 12);
CreateShaderResourceViews(pd3dDevice, ppTreeTextures[2], 0, 12);
```

그리고 다음 함수들을 추가하였습니다.

`D3D12_SHADER_BYTECODE CBillboardObjectsShader::CreateVertexShader()`

`D3D12_SHADER_BYTECODE CBillboardObjectsShader::CreatePixelShader()`

`D3D12_INPUT_LAYOUT_DESC CBillboardObjectsShader::CreateInputLayout()`

Object.h에 다음 클래스들을 추가하고, Object.cpp에 다음 클래스들의 구현부를 추가하였습니다.

`class CGrassObject : public CGameObject`

`class CMultiSpriteObject : public CGameObject`

Shaders.hlsl에 다음을 추가하였습니다.

```
179  struct VS_TEXTURED_INPUT
185  struct VS_TEXTURED_OUTPUT
191  VS_TEXTURED_OUTPUT VSTextured(VS_TEXTURED_INPUT input)
201  VS_TEXTURED_OUTPUT VSSpriteAnimation(VS_TEXTURED_INPUT input)
223  float4 PSTextured(VS_TEXTURED_OUTPUT input) : SV_TARGET
```

Mesh.h에 다음 클래스를 추가하고, Mesh.cpp에 다음 클래스의 구현부를 추가하였습니다.

`class CVertex`

Mesh.cpp의 다음 함수에서

```
70  CTexturedRectMesh::CTexturedRectMesh(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, float fWidth, float fHeight, float fDepth,
```

이 부분을 수정하였습니다.

```
191     m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pVertices, sizeof(CTexturedVertex) * m_nVertices,
192         D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, &m_pd3dVertexUploadBuffer);
193
194     m_d3dVertexBufferView.BufferLocation = m_pd3dVertexBuffer->GetGPUVirtualAddress();
195     m_d3dVertexBufferView.StrideInBytes = sizeof(CTexturedVertex);
196     m_d3dVertexBufferView.SizeInBytes = sizeof(CTexturedVertex) * m_nVertices;
197 }
```

3. 모델+터레인+빌보드 프로젝트에 물 프로젝트 통합

Mesh.h에 다음 클래스들을 추가하고, Mesh.cpp에 다음 클래스들의 구현부를 추가하였습니다.

```
class CGridMesh : public CMesh
```

```
class CDiffusedVertex : public CVertex
```

```
class CDiffusedTexturedVertex : public CDiffusedVertex
```

Object.h에 다음 클래스를 추가하고, Object.cpp에 다음 클래스의 구현부를 추가하였습니다.

```
class CRippleWater : public CGameObject
```

Scene.cpp의 다음 함수에서

```
void CScene::BuildObjects(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList)
```

```
m_pTerrainWater = new CRippleWater(pd3dDevice, pd3dCommandList, m_pd3dGraphicsRootSignature, 257, 257, 17, 17, xmf3Scale, xmf4Color);
m_pTerrainWater->SetPosition(+(257 * 0.5f), 610 /*667*/ /*m_pTerrain->GetHeight(257 * 0.5f, 257 * 0.5f)*/, +(257 * 0.5f));
//m_pTerrainWater->SetPosition(+(257 * 0.5f), 155.0f, +(257 * 0.5f));
```

물 객체를 빌드하고, 위치를 설정했습니다.

Shader.h에 다음 클래스를 추가하고, Shader.cpp에 다음 클래스의 구현부를 추가하였습니다.

```
class CRippleWaterShader : public CTexturedShader
```

Shaders.hlsl에 다음을 추가하였습니다.

```
struct VS_RIPPLE_WATER_INPUT

struct VS_RIPPLE_WATER_OUTPUT

VS_RIPPLE_WATER_OUTPUT VSRippleWater(VS_RIPPLE_WATER_INPUT input)

float4 PSRippleWater(VS_RIPPLE_WATER_OUTPUT input) : SV_TARGET
```

다음 구조체에서

```
struct MATERIAL
```

```
float4 m_cSpecular; //a = power
```

위 라인을 주석 처리하고

```
13
14  {
15      float       gfCurrentTime;
16      float       gfElapsedTime;
17      float2      gf2CursorPos;
};
```

다음 변수들을 추가하였습니다.

```
input.position.y += sin(gMaterial.gfCurrentTime * 1.0f + (((input.position.x * input.position.x) + (input.position.z * input.position.z))) * 0.0001f) * 10.0f;
//input.position.y += sin(gMaterial.gfCurrentTime * 0.35f + input.position.x * 0.35f) * 2.95f + cos(gMaterial.gfCurrentTime * 0.30f + input.position.z * 0.35f) * 2.05f;
```

```
#ifdef _WITH_STATIC_MATRIX
    sf3x3TextureAnimation._m21 = gMaterial.gfCurrentTime * 0.00125f;

#else
    uv.y += gMaterial.gfCurrentTime * 0.0825f /*0.00125f*/;
    uv.x += gMaterial.gfCurrentTime * 0.00125f;
```

위 코드에서 gfCurrentTime을 gMaterial.gfCurrentTime로 바꿔주었습니다.

그리고 물의 위치를 변경하기 위해

```
if (610.0f < output.position.y) output.position.y = 610.0f;
```

다음과 같이 155.0f를 610.0f로 수정했습니다.

물을 출렁이게 하기 위해

```
input.position.y += sin(gMaterial.gfCurrentTime * 1.0f + (((input.position.x * input.position.x) + (input.position.z * input.position.z))) * 0.0001f) * 10.0f;
```

위 라인을 주석 해제하고

```
//input.position.y += sin(gMaterial.gfCurrentTime * 0.35f + input.position.x * 0.35f) * 2.95f + cos(gMaterial.gfCurrentTime * 0.30f + input.position.z * 0.35f) * 2.05f;
```

위 라인을 주석 처리 했습니다.

물을 더 빠르게 흐르게 하기 위해

```
#else
    uv.y += gMaterial.gfCurrentTime * 0.0825f /*0.00125f*/;
```

위 라인에서 0.00125f를 0.0825f로 수정하였습니다.

Shader.cpp의 다음 함수에

```
void CRippleWaterShader::CreateShader(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, ID3D12RootSignature* pd3dGraphicsRootSignature)
```

다음 코드를 추가하였습니다.

```
1433        m_nPipelineStates = 1;
1434        m_ppd3dPipelineStates = new ID3D12PipelineState * [m_nPipelineStates];
```

Scene.cpp의 다음 함수에

```
ID3D12RootSignature* CScene::CreateGraphicsRootSignature(ID3D12Device* pd3dDevice)
```

다음 코드를 추가하였습니다.

```
pd3dDescriptorRanges[10].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[10].NumDescriptors = 3;
pd3dDescriptorRanges[10].BaseShaderRegister = 3; //t4: gtxtWaterBaseTexture, t5: gtxtWaterDetailTe
pd3dDescriptorRanges[10].RegisterSpace = 0;
pd3dDescriptorRanges[10].OffsetInDescriptorsFromTableStart = D3D12_DESCRIPTOR_RANGE_OFFSET_APPEND;
```

```
pd3dRootParameters[13].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[13].DescriptorTable.NumDescriptorRanges = 1;
pd3dRootParameters[13].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[10];
pd3dRootParameters[13].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;
```

```
void CScene::Render(ID3D12GraphicsCommandList* pd3dCommandList, CCamera* pCamera)
```

다음 함수에

```
pd3dCommandList->SetGraphicsRoot32BitConstants(1, 1, cuT, 28);
pd3dCommandList->SetGraphicsRoot32BitConstants(1, 1, elT, 29);
pd3dCommandList->SetGraphicsRoot32BitConstants(1, 1, x, 30);
pd3dCommandList->SetGraphicsRoot32BitConstants(1, 1, y, 31);
```

다음 코드를 추가하였습니다.

위의 변수들은 GameFramework.cpp의 다음 함수에서

```
620  void CGameFramework::UpdateShaderVariables()
621  {
622      float fCurrentTime = m_GameTimer.GetTotalTime();
623      float fElapsedTime = m_GameTimer.GetTimeElapsed();
624
625
626      m_pScene->cuT = &fCurrentTime;
627      m_pScene->elT = &fElapsedTime;
628
629      POINT ptCursorPos;
630      ::GetCursorPos(&ptCursorPos);
631      ::ScreenToClient(m_hWnd, &ptCursorPos);
632      float fxCursorPos = (ptCursorPos.x < 0) ? 0.0f : float(ptCursorPos.x);
633      float fyCursorPos = (ptCursorPos.y < 0) ? 0.0f : float(ptCursorPos.y);
634
635      m_pScene->x = &fxCursorPos;
636      m_pScene->y = &fyCursorPos;
637  }
```

이렇게 값을 할당했습니다.

Object.cpp의 다음 함수를

```
void CMaterial::UpdateShaderVariables(ID3D12GraphicsCommandList* pd3dCommandList)
```

다음과 같이 수정하였습니다.

```
void CMaterial::UpdateShaderVariables(ID3D12GraphicsCommandList* pd3dCommandList)
{
        pd3dCommandList->SetGraphicsRoot32BitConstants(1, 4, &m_xmf4AmbientColor, 16);
        pd3dCommandList->SetGraphicsRoot32BitConstants(1, 4, &m_xmf4AlbedoColor, 20);
-       pd3dCommandList->SetGraphicsRoot32BitConstants(1, 4, &m_xmf4SpecularColor, 24);
```

```
306     -       pd3dCommandList->SetGraphicsRoot32BitConstants(1, 1, &m_nType, 31);
    316 +       pd3dCommandList->SetGraphicsRoot32BitConstants(1, 1, &m_nType, 27);
```

다음 함수는

```
void CGameObject::Render(ID3D12GraphicsCommandList* pd3dCommandList, CCamera* pCamera)
```

이 부분을 수정하였습니다.

```
494     -       if (!(m_ppMaterials[0]->m_pShader) && m_ppMaterials[0]->m_pTexture)//
    509 +       if (!(m_ppMaterials[0]->m_pShader) && m_ppMaterials[0]->m_pTexture
    510 +           ||3== m_ppMaterials[0]->m_pTexture->m_nTextures)//
495 511         {
496     -           pd3dCommandList->SetGraphicsRoot32BitConstants(1, 3, &m_ppMaterials[0]->m_pTexture->texMat, 28);
    512 +           pd3dCommandList->SetGraphicsRoot32BitConstants(1, 3, &m_ppMaterials[0]->m_pTexture->texMat, 24);
```

다음 함수는

```
CRippleWater::CRippleWater(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, ID3D12RootSignature* pd3dGraphicsRootSignature, int nWidth, int
```

```
nLength, int nBlockWidth, int nBlockLength, XMFLOAT3 xmf3Scale, XMFLOAT4 xmf4Color) //: CGameObject(0)
```

이 부분을 수정하였습니다.

```
1334        -       CRippleWaterShader* pRippleWaterShader = new CRippleWaterShader();
1335        -       pRippleWaterShader->CreateShader(pd3dDevice, pd3dCommandList, pd3dGraphicsRootSignature);
1336        -       pRippleWaterShader->CreateCbvSrvDescriptorHeaps(pd3dDevice, 0, 3);
1337        -       pRippleWaterShader->CreateShaderVariables(pd3dDevice, pd3dCommandList);
            1350 +
            1351 +
1338        1352
1339        1353        CTexture* pWaterTexture = new CTexture(3, RESOURCE_TEXTURE2D, 0, 1);
1340        1354        pWaterTexture->LoadTextureFromDDSFile(pd3dDevice, pd3dCommandList, L"Image/Water_Base_Texture_0.dds", RESOURCE_TEXTURE2D, 0);
1341        1355        pWaterTexture->LoadTextureFromDDSFile(pd3dDevice, pd3dCommandList, L"Image/Water_Detail_Texture_0.dds", RESOURCE_TEXTURE2D, 1);
1342        1356        pWaterTexture->LoadTextureFromDDSFile(pd3dDevice, pd3dCommandList, L"Image/Lava(Diffuse).dds", RESOURCE_TEXTURE2D, 2);
1343        -       pRippleWaterShader->CreateShaderResourceViews(pd3dDevice, pWaterTexture, 0, 5);
            1357 +      CRippleWaterShader* pRippleWaterShader = new CRippleWaterShader();
            1358 +      pRippleWaterShader->CreateCbvSrvDescriptorHeaps(pd3dDevice, 0, 3);
            1359 +      //SetSbvCPUDescriptorHandle(pRippleWaterShader->GetGPUCbvDescriptorStartHandle());
            1360 +      pRippleWaterShader->CreateShaderResourceViews(pd3dDevice, pWaterTexture, 0, 13);
1344        1361
1345        1362        UINT ncbElementBytes = ((sizeof(CB_GAMEOBJECT_INFO) + 255) & ~255); //256◆◆ ◆◆◆
1346        1363
            1364 +
            1365 +      pRippleWaterShader->CreateShader(pd3dDevice, pd3dCommandList, pd3dGraphicsRootSignature);
            1366 +
            1367 +      pRippleWaterShader->CreateShaderVariables(pd3dDevice, pd3dCommandList);
```

Mesh.cpp의 다음 함수의

```
CGridMesh::CGridMesh(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, int xStart, int zStart, int nWidth, int nLength, XMFLOAT3 xmf3Scale, XMFLOAT4 xmf4Color, void* pContext) : CMesh(pd3dDevice, pd3dCommandList)
```

이 부분을 수정하였습니다.

```
-   m_pd3dVertexBuffer = CreateBufferResource(pd3dDevice, pd3dCommandList, pVertices, m_nStride * m_nVertices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, &m_pd3dVertexUploadBuffer);
    //m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, m_pxmf3Positions, sizeof(XMFLOAT3) * m_nVertices,
        //D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, &m_pd3dVertexUploadBuffer);
    m_pd3dVertexBuffer = CreateBufferResource(pd3dDevice, pd3dCommandList, pVertices, sizeof(CDiffusedTexturedVertex) * m_nVertices,
        D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, &m_pd3dVertexUploadBuffer);

    m_d3dVertexBufferView.BufferLocation = m_pd3dVertexBuffer->GetGPUVirtualAddress();
-   m_d3dVertexBufferView.StrideInBytes = m_nStride;
-   m_d3dVertexBufferView.SizeInBytes = m_nStride * m_nVertices;
+   m_d3dVertexBufferView.StrideInBytes = sizeof(CDiffusedTexturedVertex);
+   m_d3dVertexBufferView.SizeInBytes = sizeof(CDiffusedTexturedVertex) * m_nVertices;
```

그리고 이 부분을 수정하였습니다.

```
-       //&m_ppd3dSubSetIndexUploadBuffers[i]
-       //m_pd3dIndexBuffer = CreateBufferResource(pd3dDevice, pd3dCommandList, pnIndices, sizeof(UINT) * m_nIndices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER, &m_pd3dIndexUploadBuffer);
-       m_pd3dIndexBuffer = CreateBufferResource(pd3dDevice, pd3dCommandList, pnIndices, sizeof(UINT) * m_nIndices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER, &m_ppd3dSubSetIndexUploadBuffers
 [0]);
+       ////&m_ppd3dSubSetIndexUploadBuffers[i]
+       ////m_pd3dIndexBuffer = CreateBufferResource(pd3dDevice, pd3dCommandList, pnIndices, sizeof(UINT) * m_nIndices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER, &m_pd3dIndexUploadBuffer);
+       //m_pd3dIndexBuffer = CreateBufferResource(pd3dDevice, pd3dCommandList, pnIndices,
+       //      sizeof(UINT) * m_nIndices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER, &m_ppd3dSubSetIndexUploadBuffers[0]);

-       m_d3dIndexBufferView.BufferLocation = m_pd3dIndexBuffer->GetGPUVirtualAddress();
-       m_d3dIndexBufferView.Format = DXGI_FORMAT_R32_UINT;
-       m_d3dIndexBufferView.SizeInBytes = sizeof(UINT) * m_nIndices;
+       //m_d3dIndexBufferView.BufferLocation = m_pd3dIndexBuffer->GetGPUVirtualAddress();
+       //m_d3dIndexBufferView.Format = DXGI_FORMAT_R32_UINT;
+       //m_d3dIndexBufferView.SizeInBytes = sizeof(UINT) * m_nIndices;
+
+       m_nSubMeshes = 1;
+       m_pnSubSetIndices = new int[m_nSubMeshes];
+       m_ppnSubSetIndices = new UINT * [m_nSubMeshes];
+
+       m_ppd3dSubSetIndexBuffers = new ID3D12Resource * [m_nSubMeshes];
+       m_ppd3dSubSetIndexUploadBuffers = new ID3D12Resource * [m_nSubMeshes];
+       m_pd3dSubSetIndexBufferViews = new D3D12_INDEX_BUFFER_VIEW[m_nSubMeshes];
+       m_ppnSubSetIndices[0] = pnIndices;
+       m_pnSubSetIndices[0] = m_nIndices;
+       m_ppd3dSubSetIndexBuffers[0] = CreateBufferResource(pd3dDevice, pd3dCommandList, /*pnIndices*/m_ppnSubSetIndices[0],
+              sizeof(UINT) * /*m_nIndices*/ m_pnSubSetIndices[0], D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER, &m_ppd3dSubSetIndexUploadBuffers[0]);
+
+       m_pd3dSubSetIndexBufferViews[0].BufferLocation = m_ppd3dSubSetIndexBuffers[0]->GetGPUVirtualAddress();
+       m_pd3dSubSetIndexBufferViews[0].Format = DXGI_FORMAT_R32_UINT;
+       m_pd3dSubSetIndexBufferViews[0].SizeInBytes = sizeof(UINT) * /*m_nIndices*/m_pnSubSetIndices[0];
```

4. 충돌처리

```
490     if (m_pPlayer->attack)
491     {
492         XMFLOAT3 Cur_LookVector = m_pPlayer->GetLookVector();
493         XMFLOAT3 Cur_Pos = m_pPlayer->GetPosition();
494
495         XMVECTOR Bullet_Origin = XMLoadFloat3(&Cur_Pos);
496         XMVECTOR Bullet_Direction = XMLoadFloat3(&Cur_LookVector);
497
498         for (int i{}; i < pObjectsShader->m_nObjects; ++i)
499         {
500             float bullet_monster_distance = Vector3::Length(Vector3::Subtract(pObjectsShader->obj[i]->aabb.Center, Cur_Pos));
501             if (pObjectsShader->obj[i]->aabb.Intersects(Bullet_Origin, Bullet_Direction, bullet_monster_distance))
502             {
503
504                 cout << i << "명중" << endl;
505                 pMultiSpriteObjectShader->hit = pObjectsShader->obj[i]->GetPosition();
506                 pObjectsShader->obj.erase(pObjectsShader->obj.begin() + i);
507
508                 m_pPlayer->attack = false;
509
510             }
511         }
```

(1) if(m_pPlayer->attack) :

이 조건은 플레이어가 공격을 수행 중인지 여부를 확인합니다.

(2) XMFLOAT3 Cur_LookVector = m_pPlayer->GetLookVector(); :

플레이어의 시선(바라보는 방향)을 나타내는 벡터를 얻습니다. 이 방향은 플레이어가 총을 쏘는 방향을 나타냅니다.

(3) XMFLOAT3 Cur_Pos = m_pPlayer->GetPosition(); :

플레이어의 현재 위치를 얻습니다.

(4) XMVECTOR Bullet_Origin = XMLoadFloat3(&Cur_Pos); 및 XMVECTOR Bullet_Direction = XMLoadFloat3(&Cur_LookVector); :

Cur_Pos와 Cur_LookVector를 사용하여 총알의 시작 위치와 방향을 나타내는 XMVECTOR를 생성합니다.

(5) for(int i{}; i<pObjectShader->m_nObjects; ++i) :

게임 오브젝트를 순회하는 반복문을 시작합니다. pObjectShader 객체의 m_nObjects 속성은 게임 오브젝트의 총 수를 나타냅니다.

(6) float bullet_monster_distance =

Vector3::Length(Vector3::Subtract(pObjectsShader->obj[i]->aabb.Center, Cur_Pos)); :

현재 적 헬리콥터와 플레이어 사이의 거리를 계산합니다. 이 거리는 적 헬리콥터와 총알 간의 충돌을 감지하는 데 사용됩니다.

(7) if (pObjectShader->obj[i]->aabb.Intersects(Bullet_Origin, Bullet_Direction, bullet_monster_distance)) :

적 헬리콥터의 경계 상자와 총알 간의 충돌을 검사합니다.

Intersects 메서드를 사용하여 충돌 여부를 확인하고, 충돌이 발생하면 아래 코드 블록이 실행됩니다.

(8) pObjectsShader->obj.erase(pObjectsShader->obj.begin() + i); :

명중한 적 헬리콥터를 벡터에서 삭제합니다.

5. 적 헬리콥터 이동

```
526  void CObjectsShader::BuildObjects(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList,
527      ID3D12RootSignature* pd3dGraphicsRootSignature, void* pContext)
528  {
529      m_nObjects = 120;
530      m_ppObjects = new CGameObject * [m_nObjects];
531
532      CreateCbvSrvDescriptorHeaps(pd3dDevice, 0, 17 + 50); //SuperCobra(17), Gunship(2)
533
534      CGameObject* pSuperCobraModel = CGameObject::LoadGeometryFromFile(pd3dDevice, pd3dCommandList, pd3dGraphicsRootSignature, "Model/SuperCobra.bin", this);
535      CGameObject* pGunshipModel = CGameObject::LoadGeometryFromFile(pd3dDevice, pd3dCommandList, pd3dGraphicsRootSignature, "Model/Gunship.bin", this);
536
537      int nColumnSpace = 5, nColumnSize = 30;
538      int nFirstPassColumnSize = (m_nObjects % nColumnSize) > 0 ? (nColumnSize - 1) : nColumnSize;
539
540      int nObjects = 0;
541      for (int h{}; h < nFirstPassColumnSize; ++h)
542      {
543          for (int i{}; i < floor(float(m_nObjects) / float(nColumnSize)); ++i)
544          {
545              if (nObjects % 2)
546              {
547                  m_ppObjects[nObjects] = new CSuperCobraObject(pd3dDevice, pd3dCommandList, pd3dGraphicsRootSignature);
548                  m_ppObjects[nObjects]->SetChild(pSuperCobraModel);
549                  pSuperCobraModel->AddRef();
550
551              }
552              else
553              {
554                  m_ppObjects[nObjects] = new CGunshipObject(pd3dDevice, pd3dCommandList, pd3dGraphicsRootSignature);
555                  m_ppObjects[nObjects]->SetChild(pGunshipModel);
556                  pGunshipModel->AddRef();
557              }
558              XMFLOAT3 xmf3RandomPosition = RandomPositionInSphere(XMFLOAT3(920.0f, 0.0f, 1200.0f), Random(20.0f, 150.0f), h - int(floor(nColumnSize / 2.0f)), nColumnSpace);
559              m_ppObjects[nObjects]->SetPosition(xmf3RandomPosition.x, xmf3RandomPosition.y + 750.0f, xmf3RandomPosition.z);
560              m_ppObjects[nObjects]->Rotate(0.0f, 90.0f, 0.0f);
561              obj.push_back(m_ppObjects[nObjects]);
```

(1) obj.push_back(m_ppObjects[nObjects]); :

obj는 CObjectsShader 클래스의 멤버 변수로, 적 헬리콥터를 저장하는 벡터입니다. 마지막 줄의 이 코드는 생성한 적 헬리콥터를 obj에 추가합니다.

```
620  void CObjectsShader::Render(ID3D12GraphicsCommandList* pd3dCommandList, CCamera* pCamera, int nPipelineState)
621  {
622
623
624      if (120 == m_nObjects || 2 == m_nObjects)
625      {
626          if (CStandardShader::m_ppd3dPipelineStates)
627              pd3dCommandList->SetPipelineState(CStandardShader::m_ppd3dPipelineStates[0]);
628
629          if (CStandardShader::m_pd3dCbvSrvDescriptorHeap)
630              pd3dCommandList->SetDescriptorHeaps(1, &(CStandardShader::m_pd3dCbvSrvDescriptorHeap));
631      }
632
633      if (2 == m_nObjects)
634          UpdateShaderVariables(pd3dCommandList);
635
636      if (!obj.empty())
637      {
638          for (const auto& o : obj)
639          {
640              o->Animate(0.16f);
641              o->UpdateTransform(NULL);
642
643              o->Render2(pd3dCommandList, pCamera);
644          }
645      }
```

(1) if (!obj.empty()) :

이 조건은 obj 벡터가 비어 있지 않은 경우, 즉 적 헬리콥터에서 렌더링 함수를 호출했을 경우에 작업을 수행합니다.

(2) for (const auto& o : obj) :

obj 벡터에 저장된 각 헬리콥터에 대해 반복합니다. auto 키워드를 사용하여 범용 자료형을 사용하며, 범위 기반 for 루프를 사용하여 컨테이너를 순회합니다. o는 현재 순회 중인 헬리콥터를 나타냅니다.

(3) o->Render2(pd3dCommandList, pCamera); :

Render2 메서드를 호출하여 적 헬리콥터를 화면에 렌더링합니다. 이 메서드는 pd3dCommandList와 카메라를 사용하여 적 헬리콥터를 그래픽스 파이프라인을 통해 렌더링합니다. 따라서 적 헬리콥터는 화면에 그려집니다.

```
470     random_device rd;
471     mt19937 gen(rd());
472     uniform_int_distribution<> dist(1, 6);
473
474     for (const auto& o : pObjectsShader->obj)
475     {
476         if (1 == dist(gen))
477             o->MoveForward(+1.0f);//forward
478         else if (2 == dist(gen))
479             o->MoveForward(-1.0f);//back
480         else if (3 == dist(gen))
481             o->MoveStrafe(-1.0f);//left
482         else if (4 == dist(gen))
483             o->MoveStrafe(+1.0f);//right
484         else if (5 == dist(gen))
485             o->MoveUp(+1.0f);//up
486         else if (6 == dist(gen))
487             o->MoveUp(-1.0f);//down
488     }
```

(1) if(1==dist(gen)) ~ else if(6==dist(gen)) :

각 헬리콥터에 대해, 1부터 6까지의 난수를 생성하고, 그 값에 따라 다양한 이동을 적용합니다. 각 조건문은 난수가 특정 값과 일치할 경우에 해당 이동을 수행합니다. 예를 들어, if(1==dist(gen))은 난수가 1인 경우에 해당 헬리콥터를 위로 이동시킵니다. 나머지 경우에는 다른 방향으로 이동합니다.

o->MoveForward(+1.0f) : 오브젝트를 앞으로 이동시킵니다.

o->MoveForward(-1.0f) : 오브젝트를 뒤로 이동시킵니다.

o->MoveStrafe(-1.0f) : 오브젝트를 좌로 이동시킵니다.

o->MoveStrafe(+1.0f) : 오브젝트를 우로 이동시킵니다.

o->MoveUp(+1.0f) : 오브젝트를 상승시킵니다.

o->MoveUp(-1.0f) : 오브젝트를 하강시킵니다.

6. 플레이어 가속

```
462  void CGameFramework::ProcessInput()
463  {
464      static UCHAR pKeysBuffer[256];
465      bool bProcessedByScene = false;
466      if (GetKeyboardState(pKeysBuffer) && m_pScene) bProcessedByScene = m_pScene->ProcessInput(pKeysBuffer);
467      if (!bProcessedByScene)
468      {
469          DWORD dwDirection{};
470          if (pKeysBuffer[0x57] & 0xF0) dwDirection |= DIR_FORWARD;//W
471          if (pKeysBuffer[0x53] & 0xF0) dwDirection |= DIR_BACKWARD;//S
472          if (pKeysBuffer[0x41] & 0xF0) dwDirection |= DIR_LEFT;//A
473          if (pKeysBuffer[0x44] & 0xF0) dwDirection |= DIR_RIGHT;//D
474          if (pKeysBuffer[0x58] & 0xF0) dwDirection |= DIR_UP;//X
475          if (pKeysBuffer[0x43] & 0xF0) dwDirection |= DIR_DOWN;//C
476          if (pKeysBuffer[0x10] & 0xF0 && dwDirection) dwDirection |= DIR_RUN;// Shift run
```

(1) pKeysBuffer[0x10] & 0xF0 :

pKeysBuffer 배열에서 0x10 위치의 값과 0xF0을 비트 단위로 비교합니다. 이렇게 비트 AND 연산자를 사용하면 0x10 위치의 비트 중에서 0xF0과 비트 AND를 수행합니다.

(2) pKeysBuffer[0x10] :

pKeysBuffer 배열에서 0x10 위치에 있는 값을 나타냅니다. 이 값은 키 입력 상태로 사용됩니다.

(3) 0xF0 :

0xF0은 16진수 상수로, 이진으로 나타내면 11110000입니다. 이 값은 비트 마스크로 사용됩니다.

(4) pKeyBuffer[0x10] & 0xF0 :

pKeyBuffer 배열에서 0x10 위치의 값과 0xF0을 비트 AND 연산하여 특정 비트 패턴을 확인합니다. 만약 pKeysBuffer[0x10]의 비트 중 상위 4비트가 모두 1로 설정되어 있을 때, 이 연산 결과는 0xF0이 됩니다.

(5) dwDirection :

dwDirection은 비트 연산을 통해 조작할 상태 정보를 저장하는 변수로 사용됩니다.

(6) dwDirection |= DIR_RUN :

이 부분은 dwDirection에 DIR_RUN 상수를 추가합니다. 비트 OR 연산자를 사용하여 DIR_RUN의 비트 패턴을 dwDirection에 추가합니다. 이것은 특정 상태를 나타내며, DIR_RUN이 상수일 때 이 비트를 설정하고 나머지 비트는 변경하지 않습니다.

(7) dwDirection |= DIR_RUN 실행 조건 :

조건 pKeysBuffer[0x10] & 0xF0의 결과가 0이 아니고, dwDirection이 0이 아닐 때 dwDirection에 DIR_RUN을 추가합니다.

```
346         case THIRD_PERSON_CAMERA:
347             SetFriction(20.5f);
348             SetGravity(XMFLOAT3(0.0f, 0.0f, 0.0f));
349             //SetMaxVelocityXZ(25.5f);
350             //SetMaxVelocityY(20.0f);
351             m_pCamera = OnChangeCamera(THIRD_PERSON_CAMERA, nCurrentCameraMode);
352             m_pCamera->SetTimeLag(0.25f);
353             m_pCamera->SetOffset(XMFLOAT3(0.0f, 15.0f, -30.0f));
354             m_pCamera->SetPosition(Vector3::Add(m_xmf3Position, m_pCamera->GetOffset()));
355             m_pCamera->GenerateProjectionMatrix(1.01f, 5000.0f, ASPECT_RATIO, 60.0f);
356             m_pCamera->SetViewport(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT, 0.0f, 1.0f);
357             m_pCamera->SetScissorRect(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT);
358             break;
```

(1) SetMaxVelocityXZ(25.5f);와 SetMaxVelocityY(20.0f);를 주석 처리 했습니다.

```
146 void CPlayer::Update(float fTimeElapsed)
147 {
148     if (direction & DIR_UP || direction & DIR_DOWN) {
149         if (direction & DIR_RUN)
150             // Above or below with run acceleration
151             SetMaxVelocityY(150.0f);
152         else
153             // Above or below with walk acceleration
154             SetMaxVelocityY(50.0f);
155     }
156     else {
157         if (direction & DIR_RUN)
158             // Left or right with run acceleration
159             SetMaxVelocityXZ(150.0f);
160         else
161             // Left or right with walk acceleration
162             SetMaxVelocityXZ(50.5f);
163     }
```

(1) direction 변수가 DIR_DOWN 상수와 일치하는지 확인하여 오브젝트가 수직 방향으로 이동 중인지 확인합니다.

(2) 만약 수직으로 이동 중이고 RUN 상태라면, 최대 Y 방향 속도를 150.0으로 설정합니다. 그렇지 않다면 50.0으로 설정합니다.

(3) 만약 수직 이동 중이 아니라면, 좌우로 이동 중인지 확인합니다.

(4) 만약 좌우로 이동 중이고 RUN 상태라면, 최대 수평 방향(XZ)속도를 150.0으로 설정합니다. 그렇지 않다면 50.5로 설정합니다.

7. 폭발 애니메이션과 점수판

GameFramework.cpp의 다음 함수에서

```
287  void CGameFramework::OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam, LPARAM lParam)
288  {
289      random_device rd;
290      mt19937 gen(rd());
291      uniform_int_distribution<> dist(3, 5);
292
293      if (m_pScene) m_pScene->OnProcessingMouseMessage(hWnd, nMessageID, wParam, lParam);
294      switch (nMessageID)
295      {
296      case WM_LBUTTONDOWN:
297          ::SetCapture(hWnd);
298          ::GetCursorPos(&m_ptOldCursorPos);
299          break;
300      case WM_RBUTTONDOWN:
301
302          m_pScene->pMultiSpriteObjectShader->m_ppObjects[0]->m_ppMaterials[0]->m_pTexture->m_bActive = true;
303          m_pPlayer->attack = true;
304
305
306          m_pScene->pMultiSpriteObjectShader->score = dist(gen);
307          cout << m_pScene->pMultiSpriteObjectShader->score << endl;
308
309          break;
```

(1) 마우스 우클릭 시

m_pScene->pMultiSpriteObjectShader->m_ppObjects[0]->m_ppMaterials[0]->m_pTexture->m_bActive = true; 이 변수에 true를 할당했습니다.

(2) 3부터 5까지의 난수를 생성해 3일 경우 B+ 점수판, 4일 경우 C+ 점수판, 5일 경우 A+ 점수판의 텍스쳐를 할당하게끔 했습니다.

Shader.h의 다음 클래스에

```
class CMultiSpriteObjectsShader : public CObjectsShader
```

다음 함수들을 추가하였습니다.

```
virtual D3D12_SHADER_BYTECODE CreateVertexShader();

virtual D3D12_SHADER_BYTECODE CreatePixelShader();

virtual D3D12_INPUT_LAYOUT_DESC CreateInputLayout();
```

Shader.cpp의 다음 함수를

```
1246    void CTexture::AnimateRowColumn(float fTime)
```

이렇게 수정하였습니다.

```
void CTexture::AnimateRowColumn(float fTime)
{
    texMat.x = float(m_nRow) / texMat.z;//가로

    if (4 != texMat.z)
        texMat.y = float(m_nCol) / texMat.z;//세로
    else
        texMat.y = float(m_nCol) / (texMat.z * 1.5f);//세로

    if (0.0f == fTime)
    {
        if (++m_nCol == texMat.z)
        {
            ++m_nRow;//가로 증가
            m_nCol = 0; //세로 0

            m_bActive = false;
        }
        if (4 != texMat.z)
        {
            if (m_nRow == texMat.z)
                m_nRow = 0;//가로 0
        }
        else
        {
            if (m_nRow == texMat.z * 1.5f)
                m_nRow = 0;//가로 0
        }
    }
}
```

위의 코드는 빌보드 스프라이트 애니메이션을 재생하는 부분입니다.

다음 함수는

```
void CMultiSpriteObjectsShader::Render(ID3D12GraphicsCommandList* pd3dCommandList, CCamera* pCamera)
```

이 부분을 수정하였습니다.

```
for (int j{}; j < 3; ++j)
{
    if (m_ppObjects[j])
    {
        if (0 == j || 1 == j)
        {
            m_ppObjects[j]->SetPosition(xmf3Position);
            m_ppObjects[j]->SetLookAt(xmf3CameraPosition, XMFLOAT3(0.0f, 1.0f, 0.0f));
        }
        else if (2 == j)
        {
            xmf3MonPos = hit;

            xmf3Position = Vector3::Add(xmf3MonPos, Vector3::ScalarProduct(xmf3PlayerLook, 50.0f, false));

            m_ppObjects[j]->SetPosition(xmf3Position);
            m_ppObjects[j]->SetLookAt(xmf3PlayerPosition, XMFLOAT3(0.0f, 1.0f, 0.0f));

            m_ppObjects[j]->m_ppMaterials[0]->SetTexture(ppSpriteTextures[score]);
        }
    }
}
```

j가 0, 1일 때는 폭발 애니메이션을 로드하고 j가 2일 때는 적 헬리콥터의 위치 정보를 갖고 있는 hit 변수를 이용하여 점수판을 로드 합니다.

8. 실행 화면