



3D게임프로그래밍1

과제 2 설명

제출일: 2022.06.20

전공: 게임공학과

학번: 2019184020

성명: 윤은지

목차

1. 조작법(키)에 대한 설명
2. 플레이어가 일정한 높이 이하로만 날아다니게
3. 플레이어와 적군의 출발점을 설정
4. 플레이어와 적군의 색상 변경
5. 일정한 시간이 지나면 적군이 플레이어를 향하여 이동
6. 플레이어의 이동 속력 조절
7. 하늘과 땅의 색상 변경
8. 적당한 크기의 지형 제작

1. 조작법(키)에 대한 설명

1.1 키보드

```
void CGameFramework::ProcessInput()
{
    static UCHAR pKeyBuffer[256];
    DWORD dwDirection = 0;
    /*키보드의 상태 정보를 반환한다. 화살표 키( '→', '←', '↑', '↓')를 누르면 플레이어를 오른쪽/왼쪽(로컬 x-축), 앞/
    뒤(로컬 z-축)로 이동한다. 'Page Up' 과 'Page Down' 키를 누르면 플레이어를 위/아래(로컬 y-축)로 이동한다.*/
    if (::GetKeyboardState(pKeyBuffer))
    {
        if (pKeyBuffer[VK_UP] & 0x01) dwDirection |= DIR_FORWARD;
        if (pKeyBuffer[VK_DOWN] & 0x01) dwDirection |= DIR_BACKWARD;
        if (pKeyBuffer[VK_LEFT] & 0x01) dwDirection |= DIR_LEFT;
        if (pKeyBuffer[VK_RIGHT] & 0x01) dwDirection |= DIR_RIGHT;
        if (pKeyBuffer[VK_PRIOR] & 0x01) dwDirection |= DIR_UP;
        if (pKeyBuffer[VK_NEXT] & 0x01) dwDirection |= DIR_DOWN;
    }
}
```

위쪽 화살표 키: 앞으로 이동

아래쪽 화살표 키: 뒤로 이동

왼쪽 화살표 키: 왼쪽으로 이동

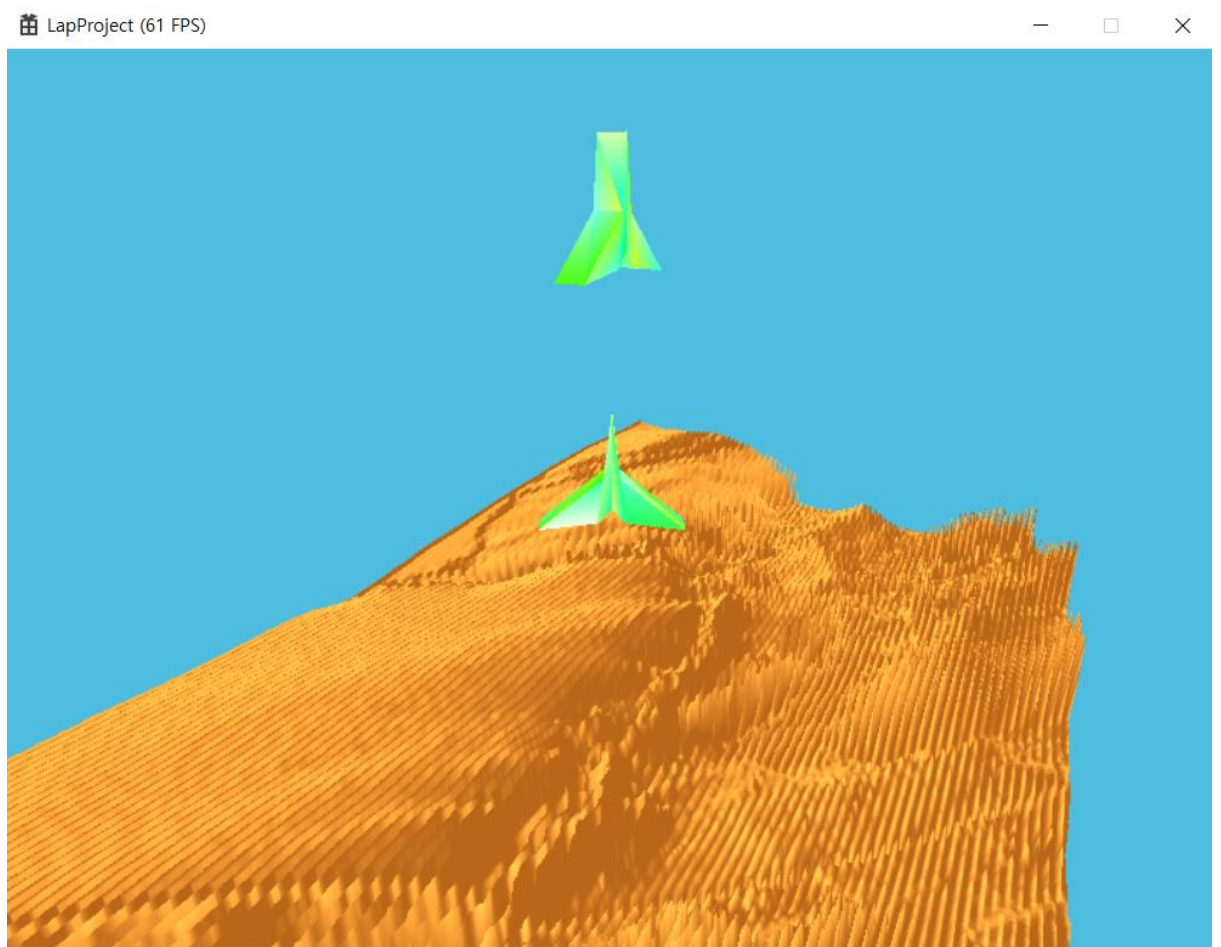
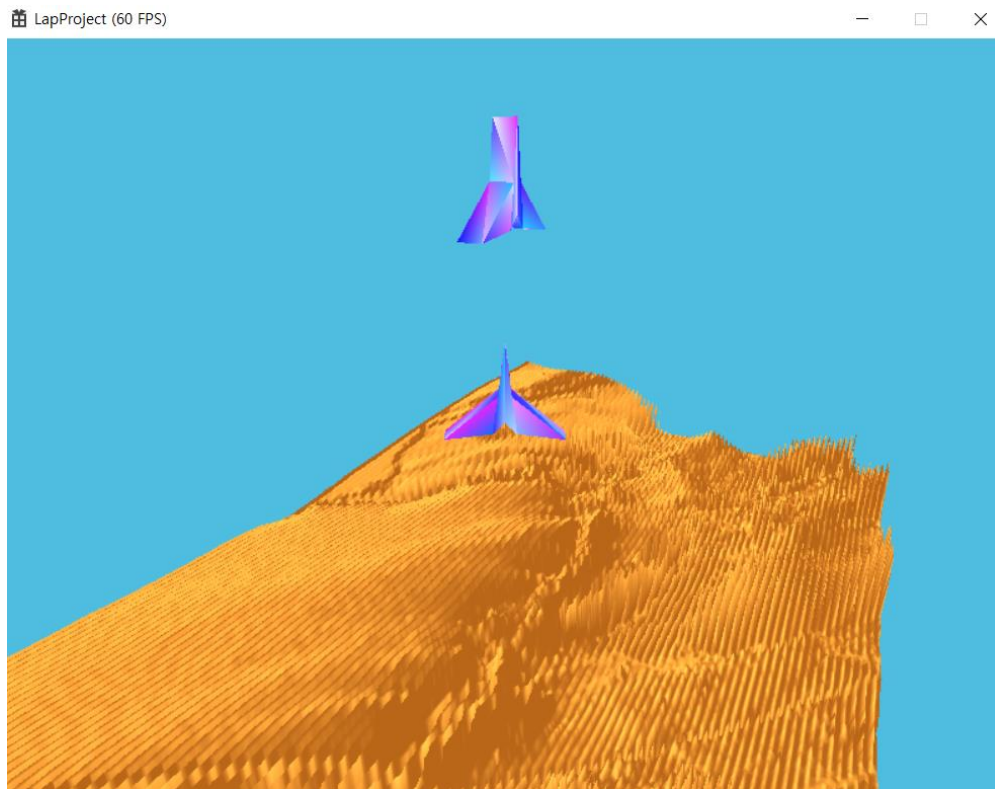
오른쪽 화살표 키: 오른쪽으로 이동

PAGE UP 키: 위로 이동

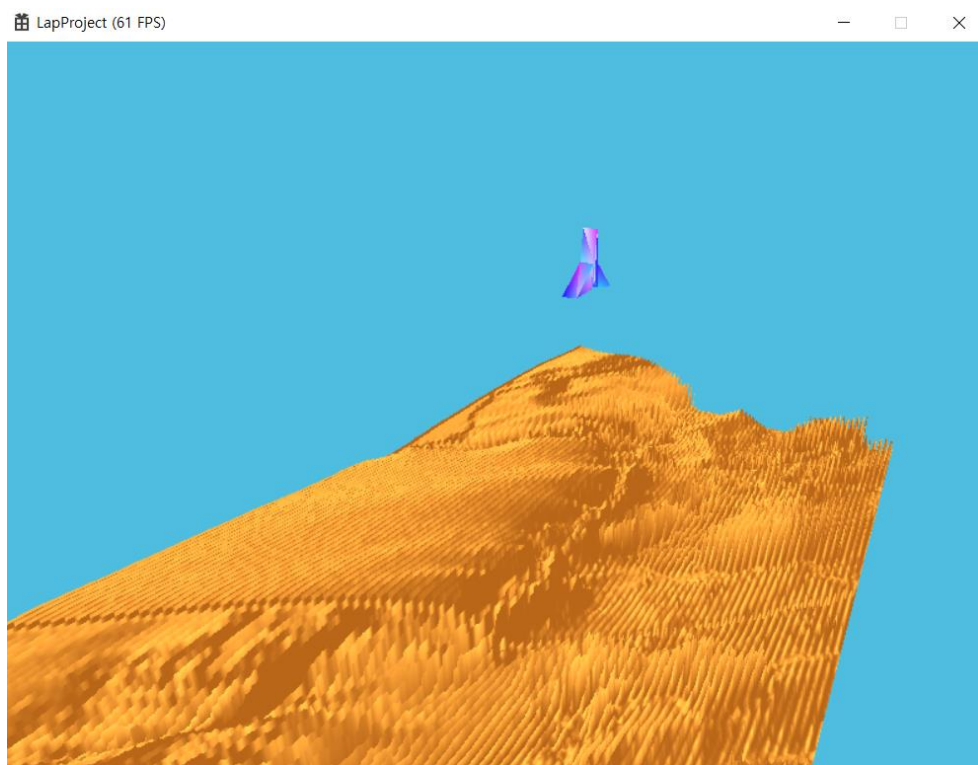
PAGE DOWN 키: 아래로 이동

```
void CGameFramework::OnProcessingKeyboardMessage(HWND hWind, UINT nMessageID, WPARAM
wParam, LPARAM lParam)
{
    switch (nMessageID)
    {
        case WM_KEYUP:
        {
            switch (wParam)
            {
                /* 'F1' 키를 누르면 1인칭 카메라, 'F2' 키를 누르면 스페이스-쉽 카메라로 변경한다, 'F3' 키를 누르면 3인칭 카메라로 변경한다.*/
                case VK_F1:
                case VK_F2:
                case VK_F3:
                {
                    if (m_pPlayer) m_pCamera = m_pPlayer->ChangeCamera((DWORD)(wParam - VK_F1 + 1), m_GameTimer.GetTimeElapsed());
                    if (m_pPlayer2) m_pCamera = m_pPlayer2->ChangeCamera((DWORD)(wParam - VK_F1 + 1), m_GameTimer.GetTimeElapsed());
                    break;
                }
                case VK_F4: //플레이어가 적들에 가까이 가서 전투장면을 볼 수 있게
                {
                    m_pPlayer->SetPosition(badGuy->GetPosition());
                    m_pPlayer2->SetPosition(badGuy2->GetPosition());
                    break;
                }
                //22.06.19
                //플레이어 색깔 변경
                case 'C':
                case 'c':
                {
                    if (1 == planeColor)
                        planeColor = 0;
                    else if (0 == planeColor)
                        planeColor = 1;
                    //
                }
            }
        }
    }
}
```

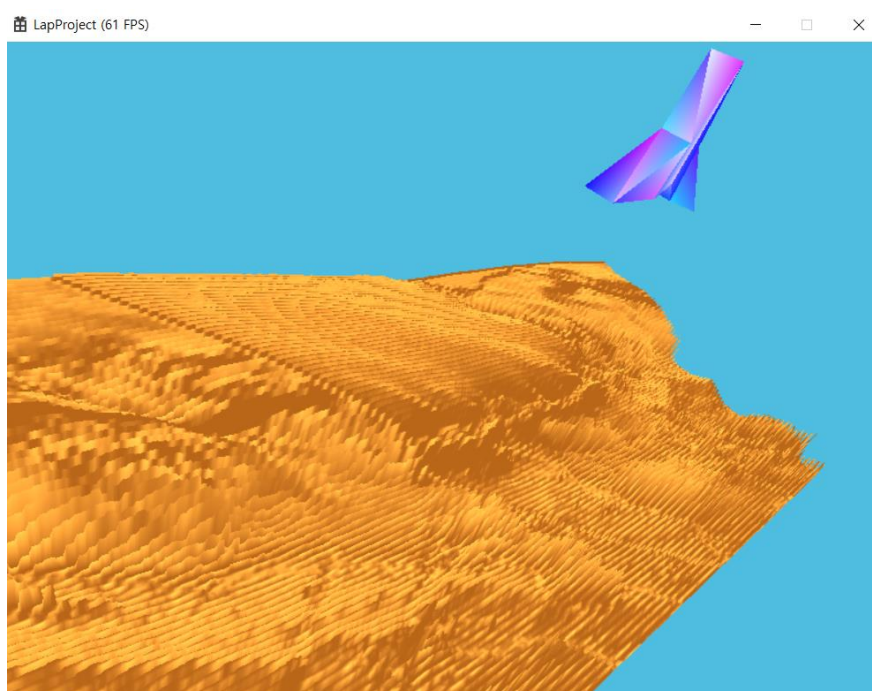
C, c: 비행기들의 색상 변경



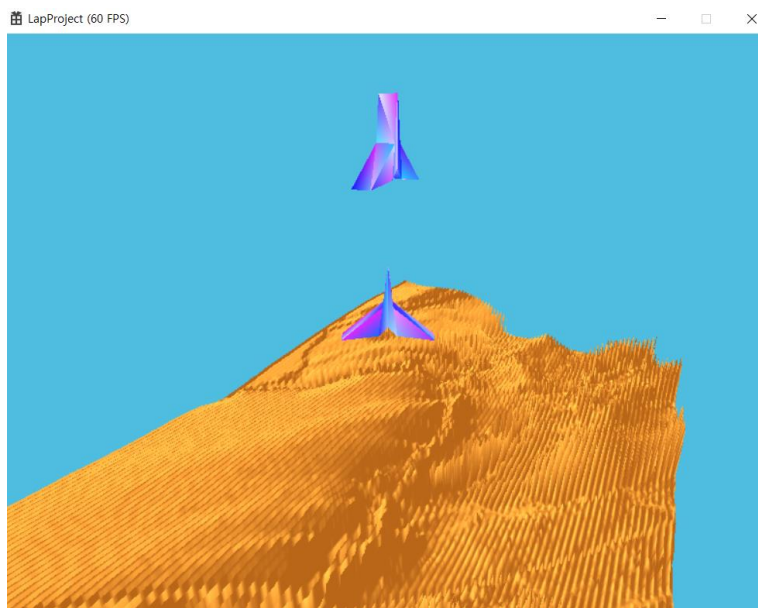
F1: 1인칭 시점 카메라로 전환



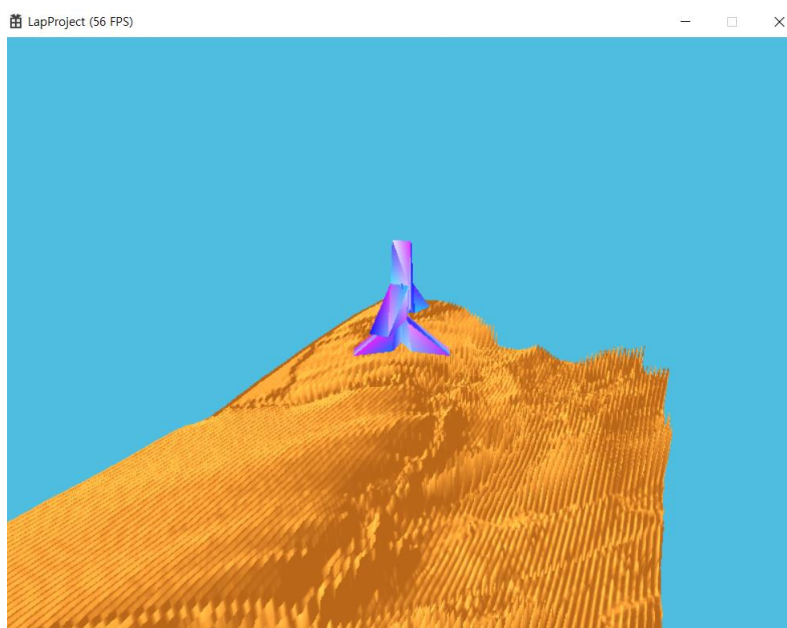
F2: 스페이스쉽 시점 카메라로 전환



F3: 3인칭 시점 카메라로 전환



F4: 플레이어가 적의 좌표로 이동



```
HRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        //22.05.09
        case WM_CHAR:
            if (wParam == 'P' || wParam == 'p')
                PostQuitMessage(0);
            break;
        //
    }
}
```

P, p: 게임 종료

1.2 마우스

마우스 버튼: 회전

```
}
float cxDelta = 0.0f, cyDelta = 0.0f;
POINT ptCursorPos;
/*마우스를 캡처했으면 마우스가 얼마만큼 이동하였는 가를 계산한다. 마우스 왼쪽 또는 오른쪽 버튼이 눌러질 때의
메시지(WM_LBUTTONDOWN, WM_RBUTTONDOWN)를 처리할 때 마우스를 캡처하였다. 그러므로 마우스가 캡처된
것은 마우스 버튼이 눌러진 상태를 의미한다. 마우스 버튼이 눌러진 상태에서 마우스를 좌우 또는 상하로 움직이면 플
레이어를 x-축 또는 y-축으로 회전한다.*/
if (::GetCapture() == m_hWnd)
{
    //마우스 커서를 화면에서 없앤다(보이지 않게 한다).
    ::SetCursor(NULL);
    //현재 마우스 커서의 위치를 가져온다.
    ::GetCursorPos(&ptCursorPos);
    //마우스 버튼이 눌린 상태에서 마우스가 움직인 양을 구한다.
    cxDelta = (float)(ptCursorPos.x - m_ptOldCursorPos.x) / 3.0f;
    cyDelta = (float)(ptCursorPos.y - m_ptOldCursorPos.y) / 3.0f;
    //마우스 커서의 위치를 마우스가 눌러졌던 위치로 설정한다.
    ::SetCursorPos(m_ptOldCursorPos.x, m_ptOldCursorPos.y);
}

//마우스 또는 키 입력이 있으면 플레이어를 이동하거나(dwDirection) 회전한다(cxDelta 또는 cyDelta).
if ((dwDirection != 0) || (cxDelta != 0.0f) || (cyDelta != 0.0f))
{
    if (cxDelta || cyDelta)
    {
        /*+cxDelta는 y-축의 회전을 나타내고 cyDelta는 x-축의 회전을 나타낸다. 오른쪽 마우스 버튼이 눌러진 경우
        cxDelta는 z-축의 회전을 나타낸다.*/

        if (pKeyBuffer[VK_RBUTTON] & 0xF0)
            m_pPlayer->Rotate(cyDelta, 0.0f, -cxDelta);
        else
            m_pPlayer->Rotate(cyDelta, cxDelta, 0.0f);

        if (pKeyBuffer[VK_RBUTTON] & 0xF0)
            m_pPlayer2->Rotate(cyDelta, 0.0f, -cxDelta);
        else
            m_pPlayer2->Rotate(cyDelta, cxDelta, 0.0f);
    }
}
```


2. 플레이어가 일정한 높이 이하로만 날아다니게

```
/*플레이어의 위치를 변경하는 함수이다. 플레이어의 위치는 기본적으로 사용자가 플레이어를 이동하기 위한 키보드를 누를 때 변경된다. 플레이어의 이동 방향(dwDirection)에 따라 플레이어를 fDistance 만큼 이동한다.*/
void CPlayer::Move(DWORD dwDirection, float fDistance, bool bUpdateVelocity)
{
    if (dwDirection)
    {
        XMFLLOAT3 xmf3Shift = XMFLLOAT3(0, 0, 0);
        //화살표 키 '↑'를 누르면 로컬 z-축 방향으로 이동(전진)한다. '↓'를 누르면 반대 방향으로 이동한다.
        if (dwDirection & DIR_FORWARD) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Look, fDistance);
        if (dwDirection & DIR_BACKWARD) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Look, -fDistance);
        //화살표 키 '→'를 누르면 로컬 x-축 방향으로 이동한다. '←'를 누르면 반대 방향으로 이동한다.
        if (dwDirection & DIR_RIGHT) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Right, fDistance);
        if (dwDirection & DIR_LEFT) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Right, -fDistance);
        // 'Page Up'을 누르면 로컬 y-축 방향으로 이동한다. 'Page Down'을 누르면 반대 방향으로 이동한다.
        if (dwDirection & DIR_UP)
        {
            //22.06.18
            // 헬리콥터는 일정한 높이 이하로만 날아다니게
            if (m_xmf3Position.y < 1000)
                xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Up, fDistance);
            //
        }
        if (dwDirection & DIR_DOWN) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Up, -fDistance);
        //플레이어를 현재 위치 벡터에서 xmf3Shift 벡터만큼 이동한다.
    }
}
```

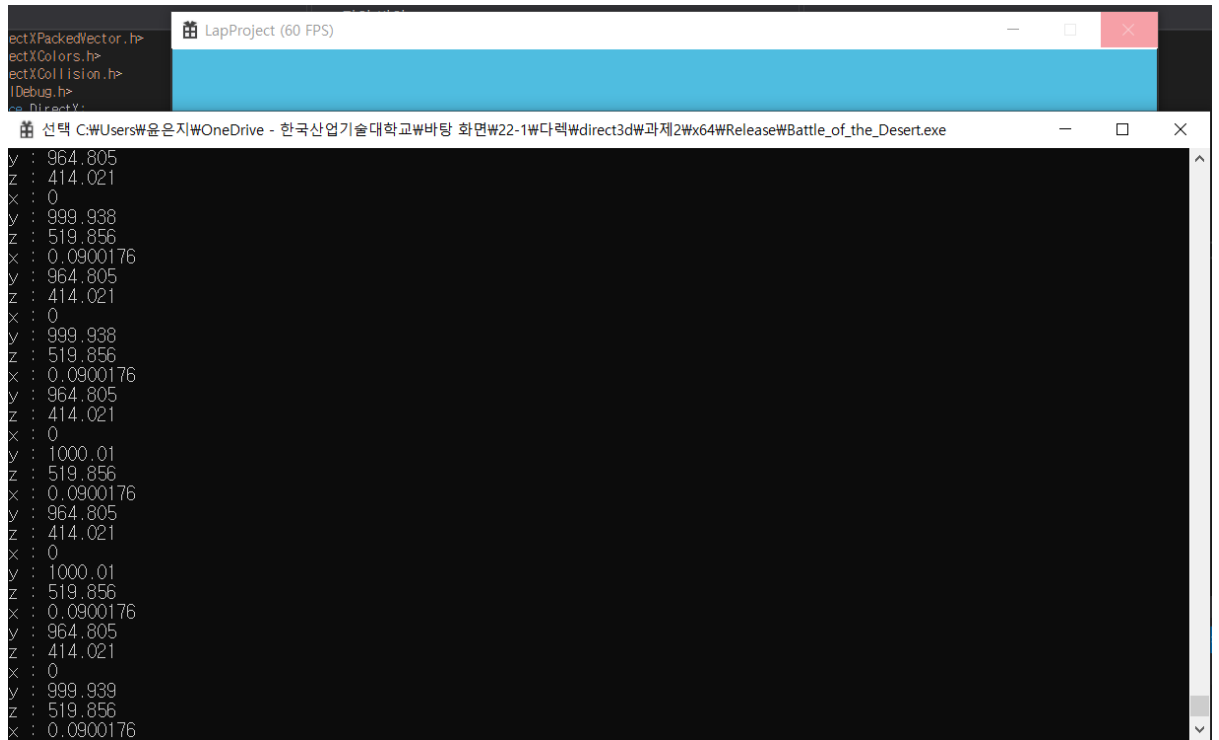
조건문을 사용하여 플레이어의 y좌표가 1000보다 작을 경우에만 위로 이동하는 것을 허용하였습니다.

```
//22.06.13////////////////////////////////////
#ifdef UNICODE
#pragma comment(linker, "/entry:WinMainCRTStartup /subsystem:console")
#else
#pragma comment(linker, "/entry:WinMainCRTStartup /subsystem:console")
#endif
////////////////////////////////////
```

stdafx.h에 위와 같은 코드를 추가하여 콘솔창을 띄웠습니다.

```
else
{
    //플레이어를 현재 위치 벡터에서 xmf3Shift 벡터만큼 이동한다.
    m_xmf3Position = Vector3::Add(m_xmf3Position, xmf3Shift);
    cout << "x : " << m_xmf3Position.x << endl;
    cout << "y : " << m_xmf3Position.y << endl;
    cout << "z : " << m_xmf3Position.z << endl;
    //플레이어의 위치가 변경되었으므로 카메라의 위치도 xmf3Shift 벡터만큼 이동한다.
    if (m_pCamera) m_pCamera->Move(xmf3Shift);
}
```


`void CPlayer::Move(const XMFLOAT3& xmf3Shift, bool bUpdateVelocity)` 함수가 호출될 때마다 플레이어의 좌표를 콘솔창에 출력했습니다.



The screenshot shows a Visual Studio IDE with a console window open. The console window title is "선택 C:\Users\윤은지\OneDrive - 한국산업기술대학교\바탕 화면\22-1\다렉\direct3d\과제2\wx64\Release\Battle_of_the_Desert.exe". The console output shows the following coordinates:

```
y : 964.805
z : 414.021
x : 0
y : 999.938
z : 519.856
x : 0.0900176
y : 964.805
z : 414.021
x : 0
y : 999.938
z : 519.856
x : 0.0900176
y : 964.805
z : 414.021
x : 0
y : 1000.01
z : 519.856
x : 0.0900176
y : 964.805
z : 414.021
x : 0
y : 1000.01
z : 519.856
x : 0.0900176
y : 964.805
z : 414.021
x : 0
y : 999.939
z : 519.856
x : 0.0900176
```

다음은 위쪽 화살표 키를 계속 눌렀을 때입니다.

플레이어의 좌표의 y값이 999.938에서 1000.01을 거쳐 더 커지지 못하고 다시 999.939로 작아진 것을 확인할 수 있습니다.

3. 플레이어와 적군의 출발점을 설정

```
CTerrainPlayer::CTerrainPlayer(IGD3DDevice* pd3dDevice, IG3DGraphicsCommandList* pd3dCommandList, IG3DRootSignature* pd3dGraphicsRootSignature, void* pContext, int meshes) : OPlayer(pMeshes)
{
    m_pCamera = ChangeCamera(THIRD_PERSON_CAMERA, 0.0f);
    CHeightMapTerrain* pTerrain = (CHeightMapTerrain*)pContext;
    //플레이어의 위치를 지형의 가운데(y=0) 좌표는 지형의 높이보다 1500 높게)로 설정한다.
    //플레이어 위치 벡터의 y 좌표가 지형의 높이보다 크고 음력이 작을수록 플레이어를 설정하였으므로 플레이어는 점차적으로 하강하게 된다.*/
    //22.06.18
    //나의 출발점 설정
    float fHeight = pTerrain->GetHeight(pTerrain->GetWidth() * 0.5f, pTerrain->GetLength() * 0.5f);
    SetPosition(XMFLOAT3(pTerrain->GetWidth() * 0.5f, fHeight + 1500.0f, pTerrain->GetLength() * 0.5f));

    float fHeight = pTerrain->GetHeight(pTerrain->GetWidth() * 0.5f, pTerrain->GetLength() * 0.5f);
    SetPosition(XMFLOAT3(0, fHeight + 1500.0f, 0));
    //
}
```

다음 함수에서 SetPosition함수를 호출하여 플레이어의 출발점을 설정하였습니다.

```
void CGameFramework::BuildObjects()
{
    m_pd3dCommandList->Reset(m_pd3dCommandAllocator, NULL);
    m_pScene = new CScene();
    if (m_pScene) m_pScene->BuildObjects(m_pd3dDevice, m_pd3dCommandList);
    m_pPlayer = new CTerrainPlayer(0, m_pd3dDevice, m_pd3dCommandList, m_pScene->GetGraphicsRootSignature(), m_pScene->GetTerrain(), 1);
    //22.06.18
    //적군 생성
    badGuy = new CTerrainPlayer(1, m_pd3dDevice, m_pd3dCommandList, m_pScene->GetGraphicsRootSignature(), m_pScene->GetTerrain(), 1);
    badGuy->SetPosition(XMFLOAT3(4184, 1000, 2056));
}
```

다음 함수에서 SetPosition함수를 호출하여 적군의 출발점을 설정하였습니다.

4. 플레이어와 적군의 색상 변경

```
void CGameFramework::BuildObjects()
{
    m_pd3dCommandList->Reset(m_pd3dCommandAllocator, NULL);
    m_pScene = new CScene();
    if (m_pScene) m_pScene->BuildObjects(m_pd3dDevice, m_pd3dCommandList);
    m_pPlayer = new CTerrainPlayer(0, m_pd3dDevice, m_pd3dCommandList, m_pScene->GetGraphicsRootSignature(), m_pScene->GetTerrain(), 1);
    //22.06.18
    //적군 생성
    badGuy = new CTerrainPlayer(1, m_pd3dDevice, m_pd3dCommandList, m_pScene->GetGraphicsRootSignature(), m_pScene->GetTerrain(), 1);
    badGuy->SetPosition(XMFLOAT3(4184, 1000, 2056));
    //
    m_pCamera = m_pPlayer->GetCamera();

    //22.06.19
    //플레이어 색상 변경
    m_pPlayer2 = new CTerrainPlayer(m_pd3dDevice, m_pd3dCommandList, m_pScene->GetGraphicsRootSignature(), m_pScene->GetTerrain(), 1);
    //22.06.18
    //적군 생성
    badGuy2 = new CTerrainPlayer(m_pd3dDevice, m_pd3dCommandList, m_pScene->GetGraphicsRootSignature(), m_pScene->GetTerrain(), 1);
    badGuy2->SetPosition(XMFLOAT3(4184, 1000, 2056));
    //
    //
}
```

각각 다른 색을 가진 플레이어를 2개 생성하고, 적군 또한 서로 색깔이 다르게 2개를 생성했습니다.

```
class CTerrainPlayer : public CPlayer
{
public:
    CTerrainPlayer(int who, ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, ID3D12RootSignature* pd3dGraphicsRootSignature, void* pContext, int nMeshes = 1);
    //플레이어 색상 변경
    CTerrainPlayer(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, ID3D12RootSignature* pd3dGraphicsRootSignature, void* pContext, int nMeshes = 1);
    //
};
```

그러기 위해 CTerrainPlayer클래스의 생성자를 오버로딩 했습니다.

```
CTerrainPlayer::CTerrainPlayer(int who, ID3D10Device* pd3dDevice, ID3D10GraphicsCommandList* pd3dCommandList, ID3D10RootSignature* pd3dGraphicsRootSignature, void* pContext,
int nMeshes) : CPlayer(nMeshes)
```

먼저 위 생성자에서는

```

XMFL0AT4 tmp;
//22.06.19
//플레이어 색상
if (1 == planeColor)
{
    //pCubeMesh->xf4Color = XMFL0AT4(1, 1, 1, 1);
}
{
    tmp = XMFL0AT4(0, 0, 1, 1);
}
else
{
    tmp = XMFL0AT4(0, 0, 1, 1);
}
//Update();
//Update(FullTrust)
//
pCubeMesh = new CAirplaneMeshDiffused(tmp, pd3dDevice, pd3dCommandList, 10.0f, 10.0f, 2.0f);

```

XMFLOAT4(0,0,1,1)의 색상 값을 설정했습니다.

그 색상 값을 CAirplaneMeshDiffused에 인자로 전달하고,

```
CAirplaneMeshDiffused::CAirplaneMeshDiffused(XMFLOAT4 color, ID3D12Device* pd3dDevice,
ID3D12GraphicsCommandList* pd3dCommandList, float fWidth, float fHeight, float fDepth) : CMesh(pd3dDevice, pd3dCommandList)
{
    xmf4Color = color;
}
```

CAirplaneMeshDiffused 클래스의 멤버 변수 xmf4Color에 매개변수 color값을 넣었습니다.

```
//비행기 메쉬의 위쪽 면
pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), -fz), Vector4::Add(xmf4Color, RANDOM_COLOR));
pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, -fz), Vector4::Add(xmf4Color, RANDOM_COLOR));
pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, -fz), Vector4::Add(xmf4Color, RANDOM_COLOR));
pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, -(fy + y3), -fz), Vector4::Add(xmf4Color, RANDOM_COLOR));
```

xmf4Color를 CDiffusedVertex의 인자로 전달했습니다.

```
CTerrainPlayer::CTerrainPlayer(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList* pd3dCommandList, ID3D12RootSignature* pd3dGraphicsRootSignature, void* pContext, int nMeshes) : CPlayer(nMeshes)
```

오버로딩된 두번째 생성자에서는

```
XMFLOAT4 tmp;
//22.06.19
//플레이어 색상
if (1 == planeColor)
{
    //pCubeMesh->xmf4Color = XMFLOAT4(1, 1, 1, 1);
}
tmp = XMFLOAT4(0, 1, 0, 1);
else
tmp = XMFLOAT4(0, 1, 0, 1);

pCubeMesh = new CAirplaneMeshDiffused(tmp, pd3dDevice, pd3dCommandList, 10.0f, 10.0f, 2.0f);
```

XMFLOAT4(0,1,0,1)의 색상 값을 설정했습니다.

그 색상 값을 CAirplaneMeshDiffused에 인자로 전달하고,

CAirplaneMeshDiffused 생성자에서 일어나는 일은 CTerrainPlayer클래스의 첫번째 생성자를 호출했을 때의 경우와 같습니다.

```

void CGameFramework::OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM
wParam, LPARAM lParam)
{
    switch (nMessageID)
    {
    case WM_KEYUP:
        switch (wParam)
        {
            /* 'F1' 키를 누르면 1인칭 카메라, 'F2' 키를 누르면 스페이스-쉽 카메라로 변경한다. 'F3' 키를 누르면 3인칭 카메라로 변경한다.*/
            case VK_F1:
            case VK_F2:
            case VK_F3:
                if (!m_pPlayer) m_pCamera = m_pPlayer->ChangeCamera((DWORD)(wParam - VK_F1 + 1), m_GameTimer.GetTimeElapsed());
                if (!m_pPlayer2) m_pCamera = m_pPlayer2->ChangeCamera((DWORD)(wParam - VK_F1 + 1), m_GameTimer.GetTimeElapsed());
                break;
            case VK_F4: //플레이어가 적들에 가까이 가서 전투장면을 볼 수 있게
                m_pPlayer->SetPosition(badGuy->GetPosition());
                m_pPlayer2->SetPosition(badGuy2->GetPosition());
                break;
            //22.06.19
            //플레이어 색깔 변경
            case 'c':
            case 'C':
                if (1 == planeColor)
                    planeColor = 0;
                else if (0 == planeColor)
                    planeColor = 1;
                //
                break;
        }
    }
}

```

C,c를 눌렀을 때 플레이어와 적군의 색상 변화가 일어납니다.

planeColor변수는 0과 1의 값만 가지므로 bool자료형으로 선언하였습니다.

```

//마우스 또는 키 입력이 있으면 플레이어를 이동하거나(dwDirection) 회전한다(cxDelta 또는 cyDelta).
if ((dwDirection != 0) || (cxDelta != 0.0f) || (cyDelta != 0.0f))
{
    if (cxDelta || cyDelta)
    {
        /*-cxDelta는 y-축의 회전을 나타내고 cyDelta는 x-축의 회전을 나타낸다. 오른쪽 마우스 버튼이 눌러진 경우
        cxDelta는 z-축의 회전을 나타낸다.*/

        if (pKeyBuffer[VK_RBUTTON] & 0xF0)
            m_pPlayer->Rotate(cyDelta, 0.0f, -cxDelta);
        else
            m_pPlayer->Rotate(cyDelta, cxDelta, 0.0f);

        if (pKeyBuffer[VK_RBUTTON] & 0xF0)
            m_pPlayer2->Rotate(cyDelta, 0.0f, -cxDelta);
        else
            m_pPlayer2->Rotate(cyDelta, cxDelta, 0.0f);
    }

    //22.06.18 플레이어 속도
    /*플레이어를 dwDirection 방향으로 이동한다(실제로는 속도 벡터를 변경한다). 이동 거리는 시간에 비례하도록 한다. 플레이어의 이동 속력은 (50/초)로 가정한다.*/
    if (dwDirection) m_pPlayer->Move(dwDirection, 50.0f * m_GameTimer.GetTimeElapsed(), true);
    if (dwDirection) m_pPlayer2->Move(dwDirection, 500.0f * m_GameTimer.GetTimeElapsed(), true);
    if (dwDirection) m_pPlayer2->Move(dwDirection, 500.0f * m_GameTimer.GetTimeElapsed(), true);
    //
}

//플레이어를 실제로 이동하고 카메라를 갱신한다. 중력과 마찰력의 영향을 속도 벡터에 적용한다.
m_pPlayer->Update(m_GameTimer.GetTimeElapsed());
//22.06.18
//플레이어를 적군 진영으로 이동
XMVECTOR3 xmf3Shift;
= XMVECTOR3(m_pPlayer->GetPosition().x - badGuy->GetPosition().x, m_pPlayer->GetPosition().y - badGuy->GetPosition().y, m_pPlayer->GetPosition().z - badGuy->GetPosition().z);
badGuy->Update(xmf3Shift, m_GameTimer.GetTimeElapsed());
//
//22.06.19
m_pPlayer2->Update(m_GameTimer.GetTimeElapsed());
xmf3Shift;
= XMVECTOR3(m_pPlayer2->GetPosition().x - badGuy2->GetPosition().x, m_pPlayer2->GetPosition().y - badGuy2->GetPosition().y, m_pPlayer2->GetPosition().z - badGuy2->GetPosition().z);
badGuy2->Update(xmf3Shift, m_GameTimer.GetTimeElapsed());
//

```

색상 변화를 주기 위해 플레이어와 적군 모두 두 개씩 생성하고 둘 모두에게 rotate, move, update를 적용하였습니다.

```

DWORD nCameraMode = (m_pCamera) ? m_pCamera->GetMode() : 0x00;
///카메라 모드가 3인치이면 플레이어 객체를 렌더링한다.
if (planeColor == 0)
{
    if (nCameraMode == THIRD_PERSON_CAMERA)
    {
        ///3인칭 카메라일 때 플레이어를 렌더링한다.
        {
            if (m_pPlayer) m_pPlayer->Render(0, m_pd3dCommandList, m_pCamera);
            ///22.08.18
            if (badGuy) badGuy->Render(1, m_pd3dCommandList, m_pCamera);
            //
        }
    }
    else if (nCameraMode == SPACESHIP_CAMERA)
    {
        ///3인칭 카메라일 때 플레이어를 렌더링한다.
        {
            if (badGuy) badGuy->Render(1, m_pd3dCommandList, m_pCamera);
        }
    }
    else if (nCameraMode == FIRST_PERSON_CAMERA)
    {
        ///3인칭 카메라일 때 플레이어를 렌더링한다.
        {
            if (badGuy) badGuy->Render(1, m_pd3dCommandList, m_pCamera);
        }
    }
}
else if (planeColor == 1)
{
    m_pCamera = m_pPlayer2->GetCamera();
    if (nCameraMode == THIRD_PERSON_CAMERA)
    {
        ///3인칭 카메라일 때 플레이어를 렌더링한다.
        {
            if (m_pPlayer2) m_pPlayer2->Render(0, m_pd3dCommandList, m_pCamera);
            ///22.08.18
            if (badGuy2) badGuy2->Render(1, m_pd3dCommandList, m_pCamera);
        }
    }
    else if (nCameraMode == SPACESHIP_CAMERA)
    {
        ///3인칭 카메라일 때 플레이어를 렌더링한다.
        {
            if (badGuy2) badGuy2->Render(1, m_pd3dCommandList, m_pCamera);
        }
    }
    else if (nCameraMode == FIRST_PERSON_CAMERA)
    {
        ///3인칭 카메라일 때 플레이어를 렌더링한다.
        {
            if (badGuy2) badGuy2->Render(1, m_pd3dCommandList, m_pCamera);
        }
    }
}
}

```

C, c를 누를 때마다 값이 변경되는 planeColor변수가 0이면 첫 번째 색깔의 플레이어와 적군을 렌더했고, planeColor변수가 1이면 두 번째 색깔의 플레이어와 적군을 렌더했습니다.

5. 일정한 시간이 지나면 적군이 플레이어를 향하여 이동

```
//22.06.18
//플레이어를 적군 진영으로 이동
XMFL0AT3 xmf3Shift
= XMFL0AT3(m_pPlayer->GetPosition().x - badGuy->GetPosition().x, m_pPlayer->GetPosition().y - badGuy->GetPosition().y, m_pPlayer->GetPosition().z - badGuy->GetPosition().z);
badGuy->Update(xmf3Shift, m_GameTimer.GetTimeElapsed());
//

//22.06.19
m_pPlayer2->Update(m_GameTimer.GetTimeElapsed());
xmf3Shift
= XMFL0AT3(m_pPlayer2->GetPosition().x - badGuy2->GetPosition().x, m_pPlayer2->GetPosition().y - badGuy2->GetPosition().y, m_pPlayer2->GetPosition().z - badGuy2->GetPosition().z);
badGuy2->Update(xmf3Shift, m_GameTimer.GetTimeElapsed());
//
```

플레이어 쪽에서 적군 쪽을 가리키는 벡터를 XMFL0AT3자료형 xmf3Shift에서 넣었습니다.

그리고 xmf3Shift를 Update함수의 인자로 전달하였습니다.

적군이 플레이어 쪽으로 이동하게 하기 위해 Update함수를 오버로딩하여 플레이어와 적군에 Update를 각각 다르게 적용하였습니다.

```
//22.06.18
//적군 위치 갱신
void CPlayer::Update(XMFL0AT3 xmf3Shift, float fTimeElapsed)
{
    //22.06.18
    //일정한 시간이 지나면 적이 나를 향해서 오게: 델타 t
    srand((unsigned int)time(NULL));
    mpTime += fTimeElapsed;
    XMFL0AT3 tmp = XMFL0AT3(xmf3Shift.x / 10, xmf3Shift.y / 10, xmf3Shift.z / 10);
    if (mpTime > 0.5f)
    {
        //플레이어를 현재 위치 벡터에서 xmf3Shift 벡터만큼 이동한다.

        m_xmf3Position = Vector3::Add(m_xmf3Position, tmp);
        mpTime = 0.f;
    }

    cout << "x : " << m_xmf3Position.x << endl;
    cout << "y : " << m_xmf3Position.y << endl;
    cout << "z : " << m_xmf3Position.z << endl;
    //플레이어의 위치가 변경되었으므로 카메라의 위치도 xmf3Shift 벡터만큼 이동한다.
}
```

fTimeElapsed변수를 계속해서 더한 mpTime변수가 0.5f를 초과할 때마다 적군의 좌표를 플레이어 쪽으로 설정한 값만큼 이동시켰습니다.

그리고 mpTime변수를 0으로 초기화하고 이를 반복했습니다.


```
x : 4184
y : 1000
z : 2056
x : 4184
y : 1000
z : 2056
x : 4184
y : 1000
z : 2056
```

```
x : 4184
y : 1000
z : 2056
x : 3765.6
y : 1078.26
z : 1850.4
x : 3765.6
y : 1078.26
z : 1850.4
```

```
x : 3050.14
y : 1211.59
z : 1498.82
x : 2745.12
y : 1268.11
z : 1348.94
x : 2745.12
y : 1268.11
z : 1348.94
```

콘솔창에 출력된 적군의 좌표 정보를 통해 적군이 플레이어 쪽으로 이동하고 있음을 확인할 수 있습니다.

6. 플레이어의 이동 속도 조절

```
//22.06.16 플레이어 속도
/*플레이어를 dwDirection 방향으로 이동한다(실제로는 속도 벡터를 변경한다). 이동 거리는 시간에 비례하도록 한다. 플레이어의 이동 속력은 (50/초)로 가정한다.*/
//if (dwDirection) m_pPlayer->Move(dwDirection, 50.0f * m_GameTimer.GetTimeElapsed(), true);
if (dwDirection) m_pPlayer->Move(dwDirection, 500.0f * m_GameTimer.GetTimeElapsed(), true);
if (dwDirection) m_pPlayer2->Move(dwDirection, 500.0f * m_GameTimer.GetTimeElapsed(), true);
//
```

Move함수의 두 번째 인자를 변경해 플레이어의 이동 속력을 높였습니다.

7. 하늘과 땅의 색상 변경

```
//22.06.14
//float pfClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };// 하늘 색깔
//float pfClearColor[4] = { 0.0f, 0.f, 0.4f, 1.0f };// 하늘 색깔
float pfClearColor[4] = { 0.31f, 0.74f, 0.88f, 1.0f };// 하늘 색깔
//
m_pd3dCommandList->ClearRenderTargetView(d3dRtvCPUDescriptorHandle,pfClearColor/*Colors::Azure*/, 0, NULL);
```

pfClearColor 배열의 인덱스 값을 변경한 후

ClearRenderTargetView함수의 두 번째 인자로 전달하여 하늘색으로 배경색을 바꿨습니다.

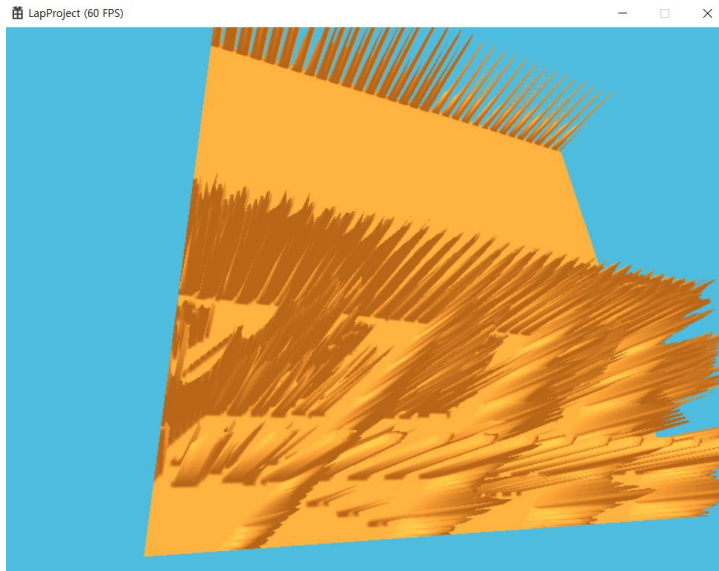
```
void CScene::BuildObjects(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList
* pd3dCommandList)
{
    m_pd3dGraphicsRootSignature = CreateGraphicsRootSignature(pd3dDevice);
    //지형을 확대할 스케일 벡터이다. x-축과 z-축은 8배, y-축은 2배 확대한다.
    XMFLAOT3 xmf3Scale(8.0f, 2.0f, 8.0f);

    //22.06.14
    //XMFLAOT4 xmf4Color(0.0f, 0.2f, 0.0f, 0.0f);
    XMFLAOT4 xmf4Color(0.5f, 0.2f, 0.0f, 0.0f);//사막 색깔
    //
    //지형을 높이 맵 이미지 파일(HightMap.raw)을 사용하여 생성한다. 높이 맵의 크기는 가로x세로(257x257)이다
    #ifdef _WITH_TERRAIN_PARTITION
    /*하나의 격자 메쉬의 크기는 가로x세로(17x17)이다. 지형 전체는 가로 방향으로 16개, 세로 방향으로 16의 격자 메
    쉬를 가진다. 지형을 구성하는 격자 메쉬의 개수는 총 256(16x16)개가 된다.*/
    m_pTerrain = new CHeightMapTerrain(pd3dDevice, pd3dCommandList,
        m_pd3dGraphicsRootSignature, _T("../Assets/Image/Terrain/HeightMap.raw"), 257, 257, 17,
        17, xmf3Scale, xmf4Color);
    #else
    //지형을 하나의 격자 메쉬(257x257)로 생성한다.
    m_pTerrain = new CHeightMapTerrain(pd3dDevice, pd3dCommandList,
        m_pd3dGraphicsRootSignature,
        _T("../Assets/Image/Terrain/HeightMap.raw"), 257, 257, 257,257, xmf3Scale, xmf4Color);

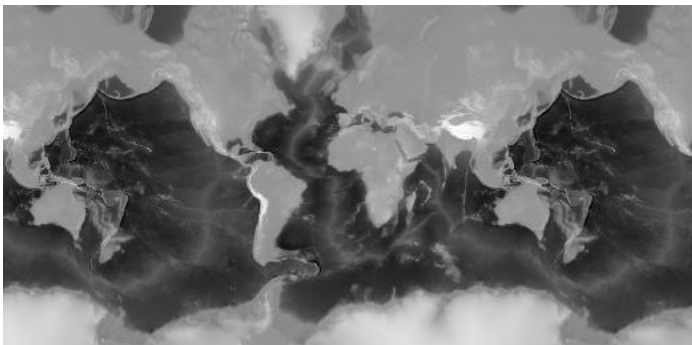
    //22.06.14
    _T("../LabProject14\테러인\리소스\map\render22.bmp"), 523, 257, 257, 257, xmf3Scale, xmf4Color);
    _T("../LabProject14\리소스\map\colormap.bmp"),1025, 1025, 257, 257, xmf3Scale, xmf4Color);
    _T("Executable\리소스\map\render22.bmp"), 523, 257, 257, 257, xmf3Scale, xmf4Color);
```

Xmf4Color변수의 값을 바꾸고 CHeightMapTerrain 생성자의 마지막 인자로 전달하여 지형의 색을 사막과 비슷하게 바꿨습니다.

8. 적당한 크기의 지형 제작



높이맵을 로드하기 전입니다.



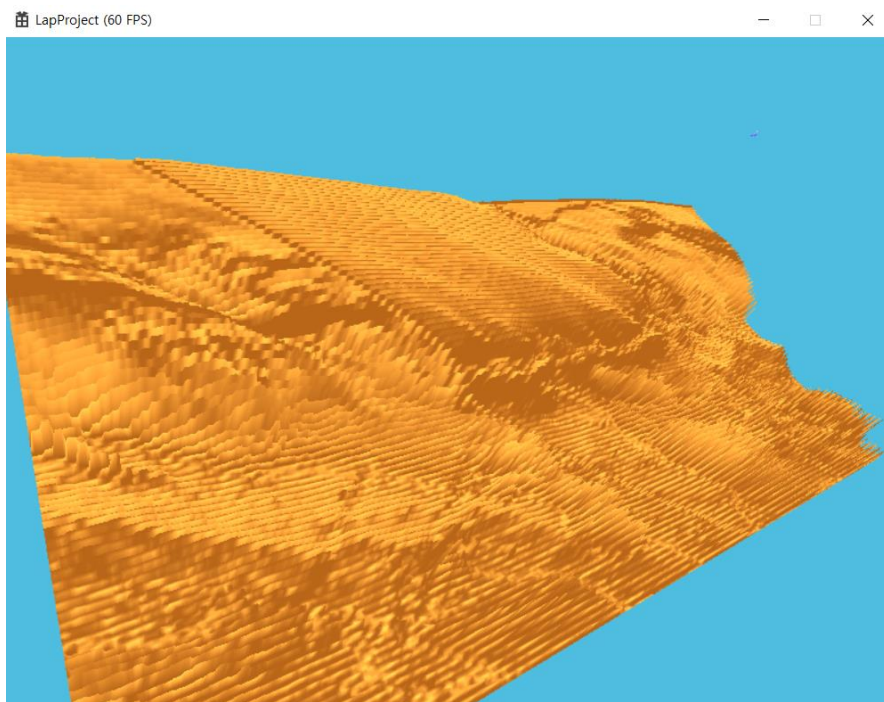
<https://tangrams.github.io/heightmapper/#1.5625/-84.9/989.8>

다음 사이트에서 다운 받은 높이맵 파일입니다.

```
//else
//지형을 하나의 격자 매쉬(257x257)로 생성한다.
m_pTerrain = new CHightMapTerrain(pd3dDevice, pd3dCommandList,
    m_pd3dGraphicsRootSignature,
    _T("../Assets/Image/Terrain/HeightMap.raw"), 257, 257, 257, 257, xmf3Scale, xmf4Color);

//22.06.14
//_T("../Assets/Image/Terrain/HeightMap.raw"), 523, 257, 257, 257, xmf3Scale, xmf4Color);
//_T("../Assets/Image/Terrain/HeightMap.raw"), 1025, 1025, 257, 257, xmf3Scale, xmf4Color);
//_T("../Assets/Image/Terrain/HeightMap.raw"), 523, 257, 257, 257, xmf3Scale, xmf4Color);
//
```

다음과 같이 높이맵 파일을 로드 하였습니다.



높이맵을 로드한 지형입니다.