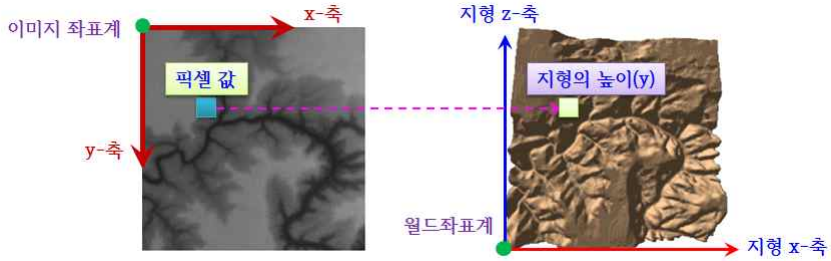
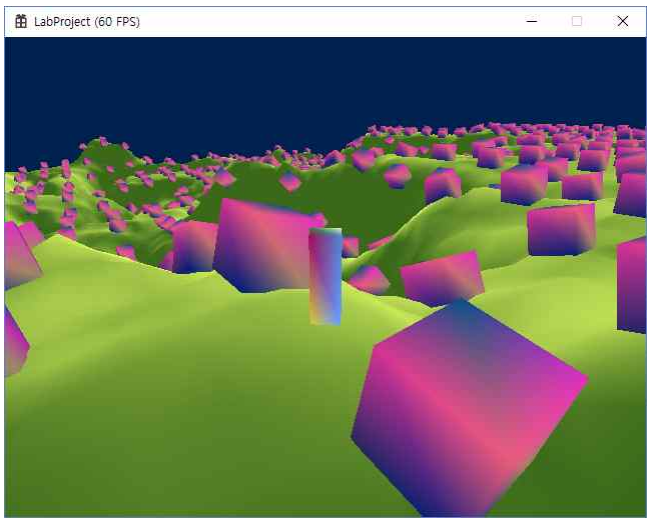
□ 예제 프로그램 15: LabProject14(실외 지형: Terrain)

예제 프로그램 LabProject11을 기반으로 지형과 여러 개의 직육면체들을 렌더링하도록 구현한다. 지형은 높이 맵(Height Map)을 사용한 실외 지형(Terrain)이다. 높이 맵은 각 픽셀이 지형의 높이를 나타내는 2차원 영상(이미지: Image)이다. 다음 그림의 왼쪽 그림은 높이 맵을 나타내는 2D 이미지이고 오른쪽은 이 높이 맵을 사용하여 생성한 지형이다.



높이 맵(왼쪽)과 높이 맵을 사용한 지형(오른쪽)의 예



① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject14를 생성한다. “LabProjects" 솔루션을 열고 솔루션 탐색기에서 마우스 오른쪽 버튼으로 『솔루션 LabProjects』를 선택하고 메뉴에서 『추가』, 『새 프로젝트』를 차례로 선택한다. 그러면 『새 프로젝트 대화상자』가 나타난다. 그러면 프로젝트 이름 “LabProject14”을 입력하고 『확인』을 선택한다.

❶ 파일 탐색기에서 프로젝트 “LabProject11” 폴더의 다음 파일을 선택하여 프로젝트 “LabProject14” 폴더에 복사한다.

- GameFramework.h

- GameFramework.cpp
- Mesh.h
- Mesh.cpp
- Camera.h
- Camera.cpp
- Player.h
- Player.cpp
- Object.h
- Object.cpp
- Scene.h
- Scene.cpp
- Shader.h
- Shader.cpp
- stdafx.h
- Timer.h
- Timer.cpp
- Shaders.hlsl

❷ 위에서 복사한 파일을 Visual Studio 솔루션 탐색기에서 프로젝트 “LabProject14”에 추가한다. 오른쪽 마우스 버튼으로 『LabProject14』를 선택하고 『추가』, 『기존 항목』를 차례로 선택한다. 그러면 “기존 항목 추가” 대화 상자가 나타난다. 위의 파일들을 마우스로 선택(Ctrl+선택)하여 『추가』를 누르면 선택된 파일들이 프로젝트 “LabProject14”에 추가된다.

② LabProject14.cpp 파일 수정하기

이제 “LabProject14.cpp" 파일의 내용을 “LabProject11.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject11.cpp" 파일의 내용 전체를 “LabProject14.cpp" 파일로 복사한다. 이제 “LabProject11.cpp" 파일에서 “LabProject11”을 “LabProject14”로 모두 바꾼다.
그리고 “LABPROJECT11”을 “LABPROJECT14”로 모두 바꾼다.

③ “Stdafx.h” 파일 수정하기

“Stdafx.h” 파일을 다음과 같이 수정한다.

❶ Vector3 네임 스페이스(Name Space)에 IsZero() 함수를 다음과 같이 정의한다.

```
namespace Vector3
{
```

```
//3-차원 벡터가 영벡터인 가를 반환하는 함수이다.
   inline bool IsZero(XMFLOAT3& xmf3Vector)
   {
      if (::IsZero(xmf3Vector.x) && ::IsZero(xmf3Vector.y) && ::IsZero(xmf3Vector.z))
return(true);
      return(false);
   }
   ...
}
```

❷ Vector4 네임 스페이스(Name Space)에 Multiply() 함수를 다음과 같이 정의한다.

```
namespace Vector4
{
//4-차원 벡터와 스칼라(실수)의 곱을 반환하는 함수이다.
   inline XMFLOAT4 Multiply(float fScalar, XMFLOAT4& xmf4Vector)
   {
      XMFLOAT4 xmf4Result;
      XMStoreFloat4(&xmf4Result, fScalar * XMLoadFloat4(&xmf4Vector));
      return(xmf4Result);
   }
   ...
}
```

④ “Mesh.h” 파일 수정하기

“Mesh.h” 파일을 다음과 같이 수정한다.

❶ 높이 맵을 표현하기 위한 “CHeightMapImage” 클래스를 다음과 같이 선언한다.

```
class CHeightMapImage
{
private:
//높이 맵 이미지 픽셀(8-비트)들의 이차원 배열이다. 각 픽셀은 0~255의 값을 갖는다.
   BYTE                        *m_pHeightMapPixels;
//높이 맵 이미지의 가로와 세로 크기이다.
   int                         m_nWidth;
   int                         m_nLength;
//높이 맵 이미지를 실제로 몇 배 확대하여 사용할 것인가를 나타내는 스케일 벡터이다.
   XMFLOAT3                    m_xmf3Scale;

public:
   CHeightMapImage(LPCTSTR pFileName, int nWidth, int nLength, XMFLOAT3 xmf3Scale);
   ~CHeightMapImage(void);

//높이 맵 이미지에서 (x, z) 위치의 픽셀 값에 기반한 지형의 높이를 반환한다.
   float GetHeight(float x, float z);
//높이 맵 이미지에서 (x, z) 위치의 법선 벡터를 반환한다.
   XMFLOAT3 GetHeightMapNormal(int x, int z);

   XMFLOAT3 GetScale() { return(m_xmf3Scale); }
```

```
    BYTE *GetHeightMapPixels() { return(m_pHeightMapPixels); }
    int GetHeightMapWidth() { return(m_nWidth); }
    int GetHeightMapLength() { return(m_nLength); }
};
```

❷ 높이 맵을 사용한 지형 메쉬를 표현하기 위한 “CHeightMapGridMesh” 클래스를 다음과 같이 선언한다.

```
class CHeightMapGridMesh : public CMesh
{
protected:
//격자의 크기(가로: x-방향, 세로: z-방향)이다.
    int                 m_nWidth;
    int                 m_nLength;
/*격자의 스케일(가로: x-방향, 세로: z-방향, 높이: y-방향) 벡터이다. 실제 격자 메쉬의 각 정점의 x-좌표, y-좌표, z-좌표는 스케일 벡터의 x-좌표, y-좌표, z-좌표로 곱한 값을 갖는다. 즉, 실제 격자의 x-축 방향의 간격은 1이 아니라 스케일 벡터의 x-좌표가 된다. 이렇게 하면 작은 격자(적은 정점)를 사용하더라도 큰 크기의 격자(지형)를 생성할 수 있다.*/
    XMFLOAT3            m_xmf3Scale;

public:
    CHeightMapGridMesh(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, int xStart, int zStart, int nWidth, int nLength, XMFLOAT3 xmf3Scale =
XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT4 xmf4Color = XMFLOAT4(1.0f, 1.0f, 0.0f, 0.0f), void
*pContext = NULL);
    virtual ~CHeightMapGridMesh();

    XMFLOAT3 GetScale() { return(m_xmf3Scale); }
    int GetWidth() { return(m_nWidth); }
    int GetLength() { return(m_nLength); }

//격자의 좌표가 (x, z)일 때 교점(정점)의 높이를 반환하는 함수이다.
    virtual float OnGetHeight(int x, int z, void *pContext);
//격자의 좌표가 (x, z)일 때 교점(정점)의 색상을 반환하는 함수이다.
    virtual XMFLOAT4 OnGetColor(int x, int z, void *pContext);
};
```

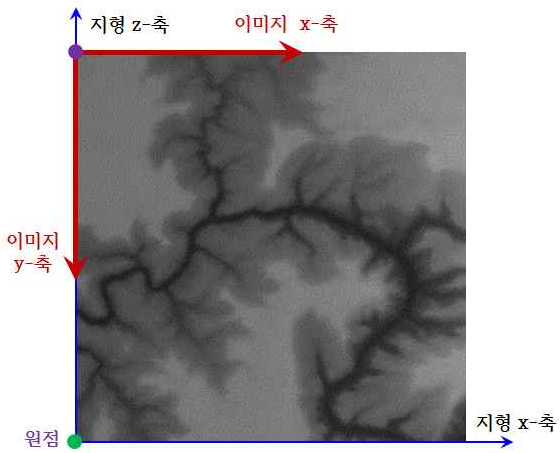
⑤ “Mesh.cpp” 파일 수정하기

“Mesh.cpp” 파일을 다음과 같이 수정한다.

❶ “CHeightMapImage” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

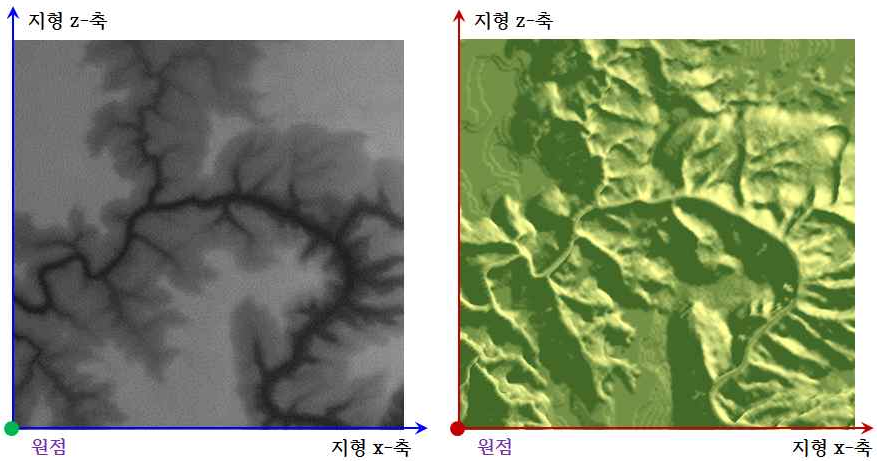
```
CHeightMapImage::CHeightMapImage(LPCTSTR pFileName, int nWidth, int nLength, XMFLOAT3
xmf3Scale)
{
    m_nWidth = nWidth;
    m_nLength = nLength;
    m_xmf3Scale = xmf3Scale;
```

```
   BYTE *pHeightMapPixels = new BYTE[m_nWidth * m_nLength];
//파일을 열고 읽는다. 높이 맵 이미지는 파일 헤더가 없는 RAW 이미지이다.
   HANDLE hFile = ::CreateFile(pFileName, GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL | FILE_ATTRIBUTE_READONLY, NULL);
   DWORD dwBytesRead;
   ::ReadFile(hFile, pHeightMapPixels, (m_nWidth * m_nLength), &dwBytesRead, NULL);
   ::CloseHandle(hFile);
```



높이 맵 이미지 좌표계

/*이미지의 y-축과 지형의 z-축이 방향이 반대이므로 이미지를 상하대칭 시켜 저장한다. 그러면 다음 그림과 같이 이미지의 좌표축과 지형의 좌표축의 방향이 일치하게 된다.*/

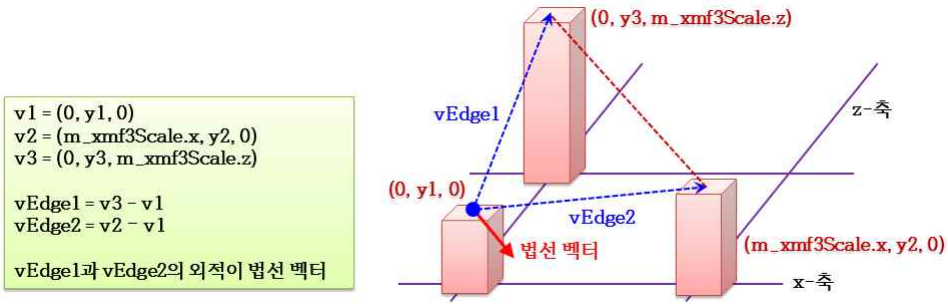


높이 맵 이미지과 y-축에서 바라 본 지형

```
   m_pHeightMapPixels = new BYTE[m_nWidth * m_nLength];
   for (int y = 0; y < m_nLength; y++)
   {
      for (int x = 0; x < m_nWidth; x++)
      {
         m_pHeightMapPixels[x + ((m_nLength - 1 - y)*m_nWidth)] = pHeightMapPixels[x +
(y*m_nWidth)];
      }
   }

   if (pHeightMapPixels) delete[] pHeightMapPixels;
```

```
}

CHeightMapImage::~CHeightMapImage()
{
   if (m_pHeightMapPixels) delete[] m_pHeightMapPixels;
   m_pHeightMapPixels = NULL;
}
```

❷ “CHeightMapImage” 클래스의 GetHeightMapNormal() 함수를 다음과 같이 정의한다.



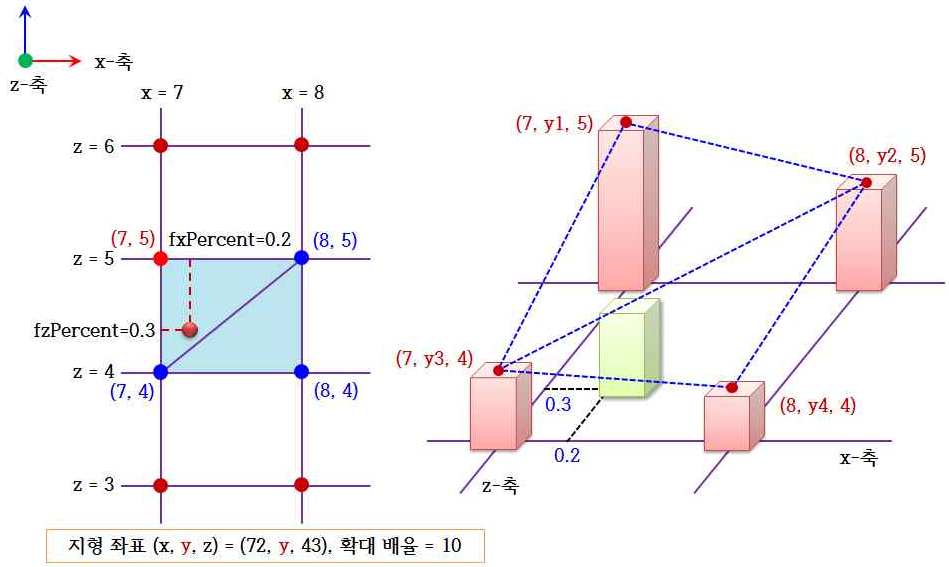
법선 벡터의 계산

```
XMFLOAT3 CHeightMapImage::GetHeightMapNormal(int x, int z)
{
//x-좌표와 z-좌표가 높이 맵의 범위를 벗어나면 지형의 법선 벡터는 y-축 방향 벡터이다.
   if ((x < 0.0f) || (z < 0.0f) || (x >= m_nWidth) || (z >= m_nLength))
return(XMFLOAT3(0.0f, 1.0f, 0.0f));

/*높이 맵에서 (x, z) 좌표의 픽셀 값과 인접한 두 개의 점 (x+1, z), (z, z+1)에 대한 픽셀 값을 사용하여 법선 벡터를 계산한다.*/
   int nHeightMapIndex = x + (z * m_nWidth);
   int xHeightMapAdd = (x < (m_nWidth - 1)) ? 1 : -1;
   int zHeightMapAdd = (z < (m_nLength - 1)) ? m_nWidth : -m_nWidth;
//(x, z), (x+1, z), (z, z+1)의 픽셀에서 지형의 높이를 구한다.
   float y1 = (float)m_pHeightMapPixels[nHeightMapIndex] * m_xmf3Scale.y;
   float y2 = (float)m_pHeightMapPixels[nHeightMapIndex + xHeightMapAdd] * m_xmf3Scale.y;
   float y3 = (float)m_pHeightMapPixels[nHeightMapIndex + zHeightMapAdd] * m_xmf3Scale.y;
//xmf3Edge1은 (0, y3, m_xmf3Scale.z) - (0, y1, 0) 벡터이다.
   XMFLOAT3 xmf3Edge1 = XMFLOAT3(0.0f, y3 - y1, m_xmf3Scale.z);
//xmf3Edge2는 (m_xmf3Scale.x, y2, 0) - (0, y1, 0) 벡터이다.
   XMFLOAT3 xmf3Edge2 = XMFLOAT3(m_xmf3Scale.x, y2 - y1, 0.0f);
//법선 벡터는 xmf3Edge1과 xmf3Edge2의 외적을 정규화하면 된다.
   XMFLOAT3 xmf3Normal = Vector3::CrossProduct(xmf3Edge1, xmf3Edge2, true);

   return(xmf3Normal);
}
```

❸ “CHeightMapImage” 클래스의 GetHeight() 함수를 다음과 같이 정의한다.

다음 그림은 지형의 확대 비율이 10일 때 지형의 좌표(xz-좌표) (72, 43)이 주어질 때 지형의 높이를 높이 맵에서 구하는 것을 보여준다. 지형의 확대 비율이 10이므로 높이 맵의 좌표는 (72/10, 43/10) = (7.2,

4.3)이다. 높이 맵의 좌표는 픽셀 단위(정수)이므로 좌표 (7.2, 4.3)는 정수로 바꾸어야 한다(높이 맵은 이차원 이미지이다). 좌표 (7.2, 4.3)를 정수로 바꾸면(소수 부분을 버림) (7, 4)가 된다. 이 좌표의 픽셀 값을 그대로 지형의 높이 값으로 사용해도 되지만 좀 더 정확한 지형의 높이를 계산하기 위하여 좌표 (7.2, 4.3)의 정수 부분과 소수 부분을 사용하여 지형의 높이(픽셀 값)를 보간(Interpolation)하도록 한다. 높이 맵을 사용한 지형의 확대 비율이 클수록 정수 좌표를 사용하면 지형의 실제 높이와 오차가 커지게 될 것이므로 조금 더 정확한 지형의 높이를 계산해야 한다.

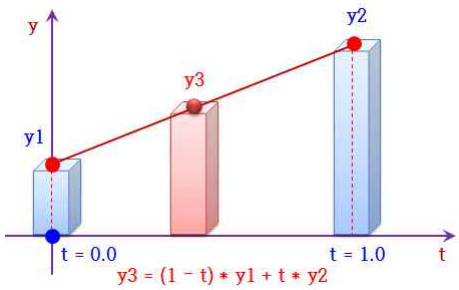


지형 높이의 보간(Interpolation)

다음 그림은 선형 보간(Linear Interpolation) 방법을 보여준다. 직선 위의 두 점에 대한 높이 y1과 y2가 주어질 때 두 점사이의 높이 값 y3는 0과 1사이의 실수 값 t를 사용하여 다음과 같이 구할 수 있다.

$$y3 = (1 - t) * y1 + t * y2 \quad (0.0 \leqq t \leqq 1.0)$$

(t = 0.0)이면 (y3 = y1)이고 (t = 1.0)이면 (y3 = y2)이다. (0.0 < t < 1.0)이면 y3는 y1에 ((y2 - y1) * t)를 더한 값이 된다. 즉, t값에 따라 y3는 y1에서 선형적으로(직선의 기울기 만큼) 증가하는 값을 갖게 된다.



선형 보간(Interpolation)

높이 맵에서 실수 좌표 (fx, fz)의 높이(픽셀 값)을 선형 보간을 사용하여 계산하는 방법은 다음과 같다. 좌표 (fx, fz)가 주어지면 fx의 정수 부분 x, 소수 부분 fxPercent, fz의 정수 부분 z, 소수 부분 fzPercent

를 계산한다. 그리고 이미지 좌표 (x, z), (x+1, z), (x, z+1), (x+1, z+1)의 높이(픽셀 값)를 높이 맵에서 구한다. 높이 맵에서 좌표 (x, z), (x+1, z), (x, z+1), (x+1, z+1)의 픽셀 값을 m[x, z], m[x+1, z], m[x, z+1], m[x+1, z+1]이라고 하자. 먼저, 좌표 (fx, z)의 픽셀 값(y1)을 구하자. 이 값은 좌표 (x, z)와 (x+1, z)의 픽셀 값 m[x, z]와 m[x+1, z]을 fxPercent를 사용하여 다음과 같이 보간한다.

```
y1 = m[x, z] * (1 - fxPercent) + m[x+1, z] * fxPercent
```

좌표 (fx, z+1)의 픽셀 값(y2)은 좌표 (x, z+1)와 (x+1, z+1)의 픽셀 값 m[x, z+1], m[x+1, z+1]을 fxPercent를 사용하여 다음과 같이 보간한다.

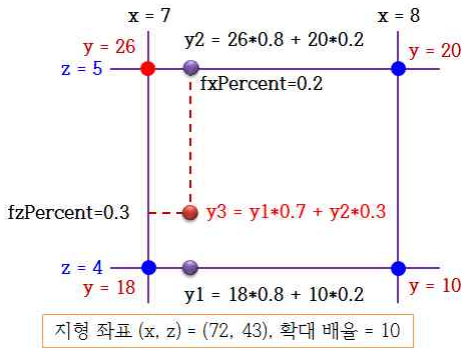
```
y2 = m[x, z+1] * (1 - fxPercent) + m[x+1, z+1] * fxPercent
```

그런 다음 좌표 (fx, fz)의 픽셀 값(y)은 y1과 y2를 fzPercent를 사용하여 다음과 같이 보간한다.

```
y = y1 * (1 - fzPercent) + y2 * fzPercent
```

다음 그림은 m[7, 4] = 18, m[8, 4] = 10, m[7, 5] = 26, m[8, 5] = 20, fxPercent = 0.2, fzPercent = 0.3일 때 좌표 (7.2, 4.3)의 픽셀 값을 보간하는 것을 보여준다.

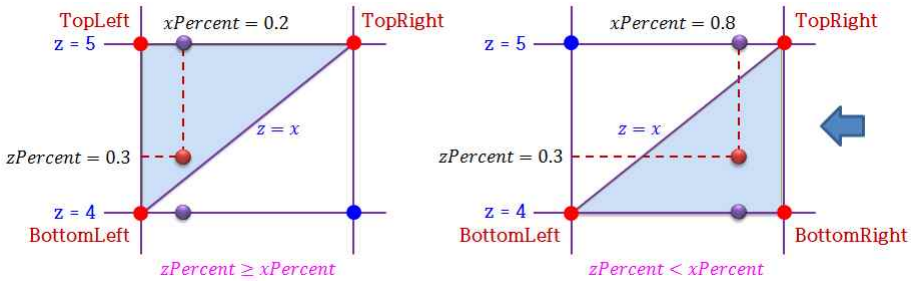
```
y1 = m[x,z] * (1-fxPercent) + m[x+1,z] * fxPercent = 18*0.8 + 10*0.2 = 16.4
y2 = m[x,z+1] * (1-fxPercent) + m[x+1,z+1] * fxPercent = 26*0.8 + 20*0.2 = 24.8
y = y1 * (1-fzPercent) + y2 * fzPercent = 16.4*0.7 + 24.8*0.3 = 18.92
```



사각형의 내부 점에 대한 보간

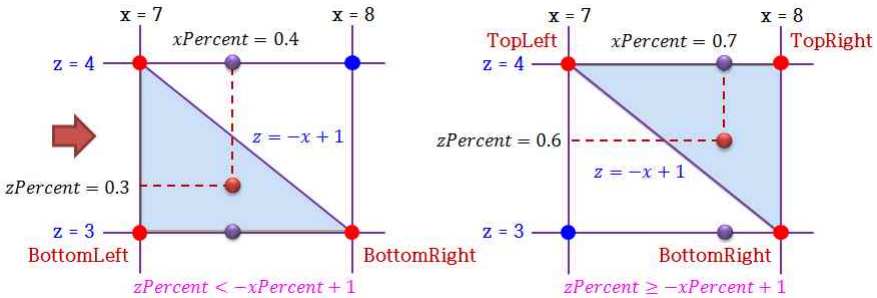
```
#define _WITH_APPROXIMATE_OPPOSITE_CORNER

float CHeightMapImage::GetHeight(float fx, float fz)
{
/*지형의 좌표 (fx, fz)는 이미지 좌표계이다. 높이 맵의 x-좌표와 z-좌표가 높이 맵의 범위를 벗어나면 지형의 높이는 0이다.*/
   if ((fx < 0.0f) || (fz < 0.0f) || (fx >= m_nWidth) || (fz >= m_nLength)) return(0.0f);

//높이 맵의 좌표의 정수 부분과 소수 부분을 계산한다.
   int x = (int)fx;
   int z = (int)fz;
   float fxPercent = fx - x;
   float fzPercent = fz - z;

   float fBottomLeft = (float)m_pHeightMapPixels[x + (z*m_nWidth)];
   float fBottomRight = (float)m_pHeightMapPixels[(x + 1) + (z*m_nWidth)];
   float fTopLeft = (float)m_pHeightMapPixels[x + ((z + 1)*m_nWidth)];
   float fTopRight = (float)m_pHeightMapPixels[(x + 1) + ((z + 1)*m_nWidth)];
#ifdef _WITH_APPROXIMATE_OPPOSITE_CORNER
//z-좌표가 1, 3, 5, ...인 경우 인덱스가 오른쪽에서 왼쪽으로 나열된다.
   bool bRightToLeft = ((z % 2) != 0);
```

```
    if (bRightToLeft)
    {
```
/*지형의 삼각형들이 오른쪽에서 왼쪽 방향으로 나열되는 경우이다. 다음 그림의 오른쪽은 (fzPercent < fxPercent)인 경우이다. 이 경우 TopLeft의 픽셀 값은 (fTopLeft = fTopRight + (fBottomLeft - fBottomRight))로 근사한다. 다음 그림의 왼쪽은 (fzPercent ≥ fxPercent)인 경우이다. 이 경우 BottomRight의 픽셀 값은 (fBottomRight = fBottomLeft + (fTopRight - fTopLeft))로 근사한다.*/



삼각형이 오른쪽에서 왼쪽으로 나열되는 경우

```
        if (fzPercent >= fxPercent)
            fBottomRight = fBottomLeft + (fTopRight - fTopLeft);
        else
            fTopLeft = fTopRight + (fBottomLeft - fBottomRight);
    }
    else
    {
```
/*지형의 삼각형들이 왼쪽에서 오른쪽 방향으로 나열되는 경우이다. 다음 그림의 왼쪽은 (fzPercent < (1.0f - fxPercent))인 경우이다. 이 경우 TopRight의 픽셀 값은 (fTopRight = fTopLeft + (fBottomRight - fBottomLeft))로 근사한다. 다음 그림의 오른쪽은 (fzPercent ≥ (1.0f - fxPercent))인 경우이다. 이 경우 BottomLeft의 픽셀 값은 (fBottomLeft = fTopLeft + (fBottomRight - fTopRight))로 근사한다.*/

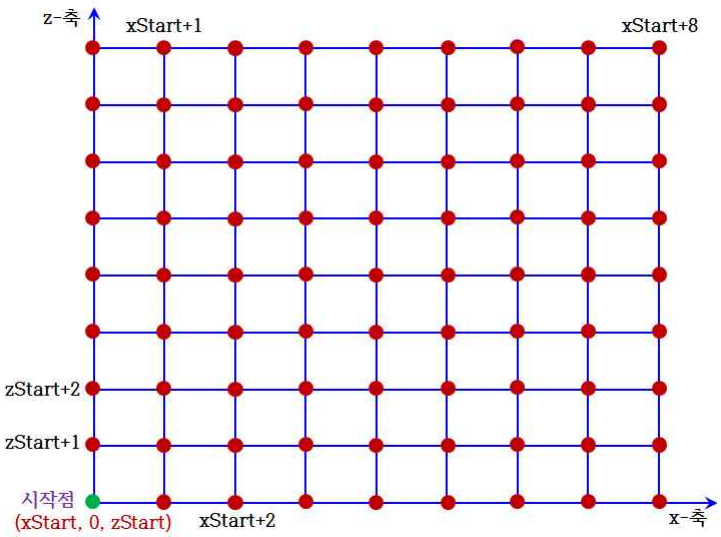


삼각형이 왼쪽에서 오른쪽으로 나열되는 경우

```
        if (fzPercent < (1.0f - fxPercent))
            fTopRight = fTopLeft + (fBottomRight - fBottomLeft);
        else
            fBottomLeft = fTopLeft + (fBottomRight - fTopRight);
    }
#endif
```
//사각형의 네 점을 보간하여 높이(픽셀 값)를 계산한다.
```
    float fTopHeight = fTopLeft * (1 - fxPercent) + fTopRight * fxPercent;
    float fBottomHeight = fBottomLeft * (1 - fxPercent) + fBottomRight * fxPercent;
    float fHeight = fBottomHeight * (1 - fzPercent) + fTopHeight * fzPercent;

    return(fHeight);
}
```

❸ “CHeightMapGridMesh” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

높이 맵을 사용하여 지형을 표현할 수 있는 격자(바둑판과 같은) 메쉬를 정의하자. CHeightMapGridMesh 클래스는 높이 맵을 사용하는 하나의 격자 메쉬를 표현한다. 이 격자는 좌표계의 xz-평면 위에 있다고 가정한다. 그리고 격자의 간격은 1이 되도록 만들 것이다(실제 지형에서 격자의 간격은 스케일 벡터에 따라 달라질 것이다). 그러면 격자의 각 교차점은 메쉬의 정점이 될 것이고 x-좌표와 z-좌표는 다음 그림과 같이 된다. 이제 이 격자의 각 교차점의 y-좌표(높이)는 교차점의 x-좌표가 a, z-좌표가 b일 때 높이 맵(이미지)에서 (a, b) 좌표의 픽셀 값을 사용한다.

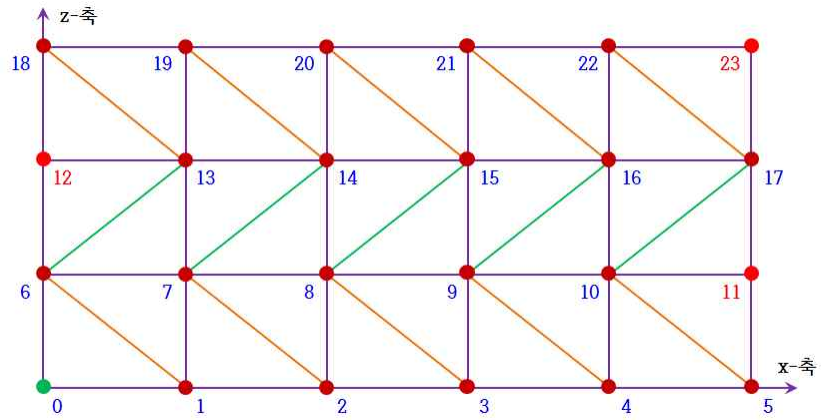


격자 메쉬

```
CHeightMapGridMesh::CHeightMapGridMesh(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList, int xStart, int zStart, int nWidth, int
nLength, XMFLOAT3 xmf3Scale, XMFLOAT4 xmf4Color, void *pContext) : CMesh(pd3dDevice,
pd3dCommandList)
{
//격자의 교점(정점)의 개수는 (nWidth * nLength)이다.
   m_nVertices = nWidth * nLength;
   m_nStride = sizeof(CDiffusedVertex);

//격자는 삼각형 스트립으로 구성한다.
   m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP;

   m_nWidth = nWidth;
   m_nLength = nLength;
   m_xmf3Scale = xmf3Scale;

   CDiffusedVertex *pVertices = new CDiffusedVertex[m_nVertices];

/*xStart와 zStart는 격자의 시작 위치(x-좌표와 z-좌표)를 나타낸다. 커다란 지형은 격자들의 이차원 배열로 만들 필요가 있기 때문에 전체 지형에서 각 격자의 시작 위치를 나타내는 정보가 필요하다.*/
   float fHeight = 0.0f, fMinHeight = +FLT_MAX, fMaxHeight = -FLT_MAX;
   for (int i = 0, z = zStart; z < (zStart + nLength); z++)
```

```
    {
        for (int x = xStart; x < (xStart + nWidth); x++, i++)
        {
//정점의 높이와 색상을 높이 맵으로부터 구한다.
            XMFLOAT3 xmf3Position = XMFLOAT3((x*m_xmf3Scale.x), OnGetHeight(x, z, pContext),
(z*m_xmf3Scale.z));
            XMFLOAT4 xmf3Color = Vector4::Add(OnGetColor(x, z, pContext), xmf4Color);
            pVertices[i] = CDiffusedVertex(xmf3Position, xmf3Color);
            if (fHeight < fMinHeight) fMinHeight = fHeight;
            if (fHeight > fMaxHeight) fMaxHeight = fHeight;
        }
    }
```

//다음 그림은 격자의 교점(정점)을 나열하는 순서를 보여준다.



격자 메쉬의 정점 순서

```
    m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pVertices,
m_nStride * 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;

    delete[] pVertices;
```

/*격자는 사각형들의 집합이고 사각형은 두 개의 삼각형으로 구성되므로 격자는 다음 그림과 같이 삼각형들의 집합이라고 할 수 있다. 격자를 표현하기 위하여 격자의 삼각형들을 정점 버퍼의 인덱스로 표현해야 한다. 삼각형 스트립을 사용하여 삼각형들을 표현하기 위하여 삼각형들은 사각형의 줄 단위로 아래에서 위쪽 방향으로(z-축) 나열한다. 첫 번째 사각형 줄의 삼각형들은 왼쪽에서 오른쪽으로(x-축) 나열한다. 두 번째 줄의 삼각형들은 오른쪽에서 왼쪽 방향으로 나열한다. 즉, 사각형의 줄이 바뀔 때마다 나열 순서가 바뀌도록 한다. 다음 그림의 격자에 대하여 삼각형 스트립을 사용하여 삼각형들을 표현하기 위한 인덱스의 나열은 다음과 같이 격자의 m번째 줄과 (m+1)번째 줄의 정점 번호를 사각형의 나열 방향에 따라 번갈아 아래, 위, 아래, 위, ... 순서로 나열하면 된다.

0, 6, 1, 7, 2, 8, 3, 9, 4, 10, 5, 11, 11, 17, 10, 16, 9, 15, 8, 14, 7, 13, 6, 12

이렇게 인덱스를 나열하면 삼각형 스트립을 사용할 것이므로 실제 그려지는 삼각형들의 인덱스는 다음과 같다.

(0, 6, 1), (1, 6, 7), (1, 7, 2), (2, 7, 8), (2, 8, 3), (3, 8, 9), ...

그러나 이러한 인덱스를 사용하면 첫 번째 줄을 제외하고 삼각형들이 제대로 그려지지 않는다. 왜냐하면 삼각형 스트립에서는 마지막 2개의 정점과 새로운 하나의 정점을 사용하여 새로운 삼각형을 그린다. 그리고 홀수 번째 삼각형의 정점 나열 순서(와인딩 순서)는 시계방향이고 짝수 번째 삼각형의 와인딩 순서는 반시계방향이어야 한다. 격자의 사각형이 한 줄에서 몇 개가 있던지 상관없이 한 줄의 마지막 삼각형은 짝수 번째 삼각형이고 와인딩 순서는 반시계 방향이다. 왜냐하면 사각형은 두 개의 삼각형으로 나누어지기 때문이다. 첫 번째 줄에서 두 번째 줄의 인덱스 나열과 실제

그려지는 삼각형들의 인덱스를 살펴보자.

..., 4, 10, 5, 11, 11, 17, 10, 16, 9, 15, 8, 14, 7, 13, 6, 12, ...

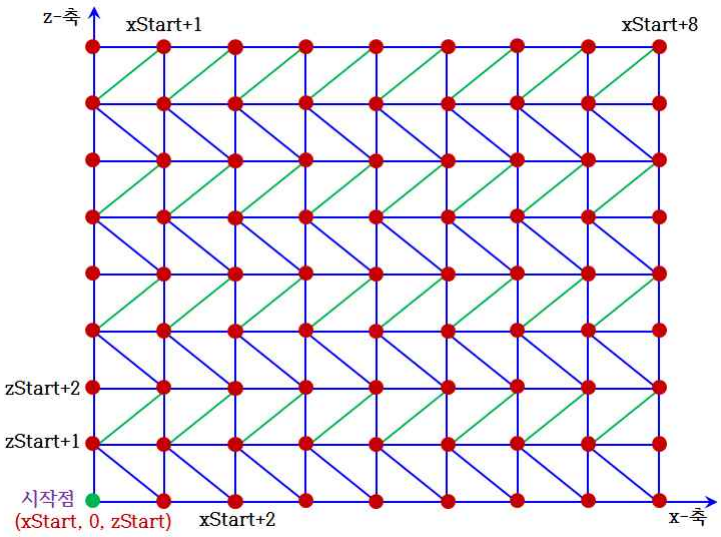
..., (4, 10, 5), **(5, 10, 11)**, (5, 11, 11), (11, 11, 17), **(11, 17, 10)**, ...

삼각형 (5, 10, 11)은 첫 번째 줄의 마지막 삼각형이고 짝수 번째이다. 삼각형 (11, 17, 10)은 두 번째 줄의 첫 번째 삼각형이고 홀수 번째이다. 홀수 번째이므로 와인딩 순서가 시계방향이어야 하는데 실제 와인딩 순서는 반시계방향이므로 그려지지 않을 것이다. 당연히 다음 삼각형도 와인딩 순서가 맞지 않으므로 그려지지 않을 것이다. 삼각형 (11, 17, 10)의 와인딩 순서가 반시계방향이므로 그려지도록 하려면 이 삼각형이 짝수 번째 삼각형이 되도록 해야 한다. 이를 위해서 줄이 바뀔 때마다 마지막 정점의 인덱스를 추가하도록 하자. 그러면 줄이 바뀐 첫 번째 삼각형은 짝수 번째 삼각형이 된다. 다음의 예에서는 11이 추가된 마지막 정점의 인덱스이다. 이렇게 하면 삼각형을 구성할 수 없어서 그려지지 않는 삼각형이 각 줄마다 3개씩 생기게 된다.

..., 4, 10, 5, 11, 11, 11, 17, 10, 16, 9, 15, 8, 14, 7, 13, 6, 12, ...

..., **(5, 10, 11)**, (5, 11, 11), (11, 11, 11), (11, 11, 17), **(11, 17, 10)**, ...

세 개의 삼각형 (5, 11, 11), (11, 11, 11), (11, 11, 17)은 삼각형을 구성할 수 없으므로 실제로 그려지지 않는다.*/



격자 메쉬의 삼각형

/*이렇게 인덱스를 나열하면 인덱스 버퍼는 ((nWidth*2)*(nLength-1))+((nLength-1)-1)개의 인덱스를 갖는다. 사각형 줄의 개수는 (nLength-1)이고 한 줄에서 (nWidth*2)개의 인덱스를 갖는다. 그리고 줄이 바뀔 때마다 인덱스를 하나 추가하므로 (nLength-1)-1개의 인덱스가 추가로 필요하다.*/

```
   m_nIndices = ((nWidth * 2)*(nLength - 1)) + ((nLength - 1) - 1);
   UINT *pnIndices = new UINT[m_nIndices];

   for (int j = 0, z = 0; z < nLength - 1; z++)
   {
      if ((z % 2) == 0)
      {
//홀수 번째 줄이므로(z = 0, 2, 4, ...) 인덱스의 나열 순서는 왼쪽에서 오른쪽 방향이다.
         for (int x = 0; x < nWidth; x++)
         {
//첫 번째 줄을 제외하고 줄이 바뀔 때마다(x == 0) 첫 번째 인덱스를 추가한다.
            if ((x == 0) && (z > 0)) pnIndices[j++] = (UINT)(x + (z * nWidth));
//아래(x, z), 위(x, z+1)의 순서로 인덱스를 추가한다.
            pnIndices[j++] = (UINT)(x + (z * nWidth));
            pnIndices[j++] = (UINT)((x + (z * nWidth)) + nWidth);
         }
      }
```

```
        else
        {
//짝수 번째 줄이므로(z = 1, 3, 5, ...) 인덱스의 나열 순서는 오른쪽에서 왼쪽 방향이다.
            for (int x = nWidth - 1; x >= 0; x--)
            {
//줄이 바뀔 때마다(x == (nWidth-1)) 첫 번째 인덱스를 추가한다.
                if (x == (nWidth - 1)) pnIndices[j++] = (UINT)(x + (z * nWidth));
//아래(x, z), 위(x, z+1)의 순서로 인덱스를 추가한다.
                pnIndices[j++] = (UINT)(x + (z * nWidth));
                pnIndices[j++] = (UINT)((x + (z * nWidth)) + nWidth);
            }
        }
    }

    m_pd3dIndexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pnIndices,
sizeof(UINT) * m_nIndices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER,
&m_pd3dIndexUploadBuffer);

    m_d3dIndexBufferView.BufferLocation = m_pd3dIndexBuffer->GetGPUVirtualAddress();
    m_d3dIndexBufferView.Format = DXGI_FORMAT_R32_UINT;
    m_d3dIndexBufferView.SizeInBytes = sizeof(UINT) * m_nIndices;

    delete[] pnIndices;
}

CHeightMapGridMesh::~CHeightMapGridMesh()
{
}
```

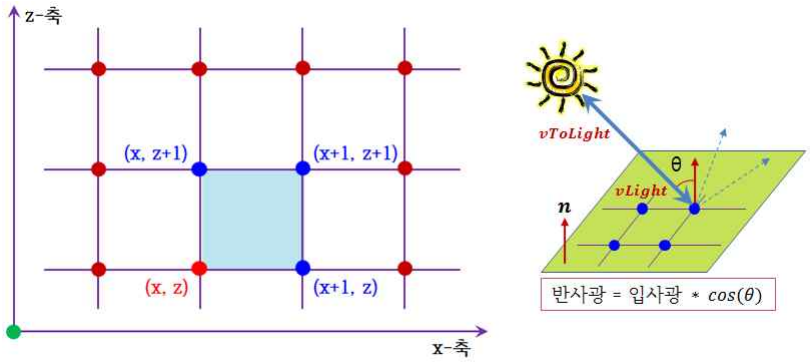
❹ “CHeightMapGridMesh” 클래스의 OnGetHeight() 함수를 다음과 같이 정의한다.

```
//높이 맵 이미지의 픽셀 값을 지형의 높이로 반환한다.
float CHeightMapGridMesh::OnGetHeight(int x, int z, void *pContext)
{
    CHeightMapImage *pHeightMapImage = (CHeightMapImage *)pContext;
    BYTE *pHeightMapPixels = pHeightMapImage->GetHeightMapPixels();
    XMFLOAT3 xmf3Scale = pHeightMapImage->GetScale();
    int nWidth = pHeightMapImage->GetHeightMapWidth();
    float fHeight = pHeightMapPixels[x + (z*nWidth)] * xmf3Scale.y;
    return(fHeight);
}
```

❺ “CHeightMapGridMesh” 클래스의 OnGetColor() 함수를 다음과 같이 정의한다. 가상적인 조명이 지형에 비춘다고 가정하고 조명이 지형의 정점에 반사되는 양을 계산하여 정점의 색상을 결정한다.

```
XMFLOAT4 CHeightMapGridMesh::OnGetColor(int x, int z, void *pContext)
{
//조명의 방향 벡터(정점에서 조명까지의 벡터)이다.
    XMFLOAT3 xmf3LightDirection = XMFLOAT3(-1.0f, 1.0f, 1.0f);
    xmf3LightDirection = Vector3::Normalize(xmf3LightDirection);
    CHeightMapImage *pHeightMapImage = (CHeightMapImage *)pContext;
```

```
   XMFLOAT3 xmf3Scale = pHeightMapImage->GetScale();
//조명의 색상(세기, 밝기)이다.
   XMFLOAT4 xmf4IncidentLightColor(0.9f, 0.8f, 0.4f, 1.0f);
/*정점 (x, z)에서 조명이 반사되는 양(비율)은 정점 (x, z)의 법선 벡터와 조명의 방향 벡터의 내적(cos)과 인접한 3개의 정점 (x+1, z), (x, z+1), (x+1, z+1)의 법선 벡터와 조명의 방향 벡터의 내적을 평균하여 구한다. 정점 (x, z)의 색상은 조명 색상(세기)과 반사되는 양(비율)을 곱한 값이다.*/
```



정점의 색상(정점의 법선 벡터와 조명의 방향 벡터의 내적)

```
   float fScale = Vector3::DotProduct(pHeightMapImage->GetHeightMapNormal(x, z),
xmf3LightDirection);
   fScale += Vector3::DotProduct(pHeightMapImage->GetHeightMapNormal(x + 1, z),
xmf3LightDirection);
   fScale += Vector3::DotProduct(pHeightMapImage->GetHeightMapNormal(x + 1, z + 1),
xmf3LightDirection);
   fScale += Vector3::DotProduct(pHeightMapImage->GetHeightMapNormal(x, z + 1),
xmf3LightDirection);
   fScale = (fScale / 4.0f) + 0.05f;
   if (fScale > 1.0f) fScale = 1.0f;
   if (fScale < 0.25f) fScale = 0.25f;
//fScale은 조명 색상(밝기)이 반사되는 비율이다.
   XMFLOAT4 xmf4Color = Vector4::Multiply(fScale, xmf4IncidentLightColor);
   return(xmf4Color);
}
```

⑥ “GameObject.h” 파일 변경하기

❶ “CGameObject” 클래스의 생성자를 다음과 같이 수정한다.

```
   CGameObject(int nMeshes=1);
```

❷ “CGameObject” 클래스의 m_pMesh 멤버 변수를 삭제한다.

```
   CMesh          *m_pMesh = NULL;
```

❸ “CGameObject” 클래스의 다음 멤버 변수를 선언한다.

```
//게임 객체는 여러 개의 메쉬를 포함하는 경우 게임 객체가 가지는 메쉬들에 대한 포인터와 그 개수이다.
   CMesh          **m_ppMeshes = NULL;
```

```
int             m_nMeshes = 0;
```

❹ “CGameObject” 클래스의 SetMesh() 함수를 다음과 같이 수정한다.

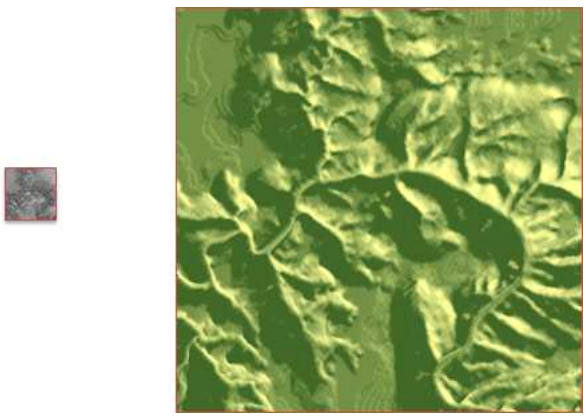
```
void SetMesh(int nIndex, CMesh *pMesh);
```

❺ “CRotatingObject” 클래스의 생성자를 다음과 같이 수정한다.

```
CRotatingObject(int nMeshes=1);
```

❻ “CHeightMapTerrain” 클래스를 다음과 같이 선언한다. 이 클래스는 높이 맵을 사용한 실외 지형을 표현하는 게임 객체이다.

```
class CHeightMapTerrain : public CGameObject
{
public:
   CHeightMapTerrain(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, ID3D12RootSignature *pd3dGraphicsRootSignature, LPCTSTR pFileName, int
nWidth, int nLength, int nBlockWidth, int nBlockLength, XMFLOAT3 xmf3Scale, XMFLOAT4
xmf4Color);
   virtual ~CHeightMapTerrain();

private:
//지형의 높이 맵으로 사용할 이미지이다.
   CHeightMapImage           *m_pHeightMapImage;

//높이 맵의 가로와 세로 크기이다.
   int                       m_nWidth;
   int                       m_nLength;

//지형을 실제로 몇 배 확대할 것인가를 나타내는 스케일 벡터이다.
   XMFLOAT3                  m_xmf3Scale;
```



높이 맵을 x-축, z-축으로 8배 확대한 지형

```
public:
//지형의 높이를 계산하는 함수이다(월드 좌표계). 높이 맵의 높이에 스케일의 y를 곱한 값이다.
   float GetHeight(float x, float z) { return(m_pHeightMapImage->GetHeight(x /
```

```
m_xmf3Scale.x, z / m_xmf3Scale.z) * m_xmf3Scale.y); }
//지형의 법선 벡터를 계산하는 함수이다(월드 좌표계). 높이 맵의 법선 벡터를 사용한다.
   XMFLOAT3 GetNormal(float x, float z) {
return(m_pHeightMapImage->GetHeightMapNormal(int(x / m_xmf3Scale.x), int(z /
m_xmf3Scale.z))); }

   int GetHeightMapWidth() { return(m_pHeightMapImage->GetHeightMapWidth()); }
   int GetHeightMapLength() { return(m_pHeightMapImage->GetHeightMapLength()); }

   XMFLOAT3 GetScale() { return(m_xmf3Scale); }
//지형의 크기(가로/세로)를 반환한다. 높이 맵의 크기에 스케일을 곱한 값이다.
   float GetWidth() { return(m_nWidth * m_xmf3Scale.x); }
   float GetLength() { return(m_nLength * m_xmf3Scale.z); }
};
```

⑦ “GameObject.cpp” 파일 변경하기

❶ “CGameObject” 클래스의 생성자와 소멸자를 다음과 같이 수정한다.

```
CGameObject::CGameObject(int nMeshes)
{
   m_xmf4x4World = Matrix4x4::Identity();

   m_nMeshes = nMeshes;
   m_ppMeshes = NULL;
   if (m_nMeshes > 0)
   {
      m_ppMeshes = new CMesh*[m_nMeshes];
      for (int i = 0; i < m_nMeshes; i++)     m_ppMeshes[i] = NULL;
   }
}

CGameObject::~CGameObject()
{
   if (m_ppMeshes)
   {
      for (int i = 0; i < m_nMeshes; i++)
      {
         if (m_ppMeshes[i]) m_ppMeshes[i]->Release();
         m_ppMeshes[i] = NULL;
      }
      delete[] m_ppMeshes;
   }

   if (m_pShader)
   {
      m_pShader->ReleaseShaderVariables();
      m_pShader->Release();
   }
}
```

❷ “CGameObject” 클래스의 SetMesh() 함수를 다음과 같이 수정한다.

```
void CGameObject::SetMesh(int nIndex, CMesh *pMesh)
{
   if (m_ppMeshes)
   {
      if (m_ppMeshes[nIndex]) m_ppMeshes[nIndex]->Release();
      m_ppMeshes[nIndex] = pMesh;
      if (pMesh) pMesh->AddRef();
   }
}
```

❸ “CGameObject” 클래스의 Render() 함수를 다음과 같이 수정한다.

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

   UpdateShaderVariables(pd3dCommandList);

   if (m_pShader) m_pShader->Render(pd3dCommandList, pCamera);

//게임 객체가 포함하는 모든 메쉬를 렌더링한다.
   if (m_ppMeshes)
   {
      for (int i = 0; i < m_nMeshes; i++)
      {
         if (m_ppMeshes[i]) m_ppMeshes[i]->Render(pd3dCommandList);
      }
   }
}
```

❹ “CGameObject” 클래스의 ReleaseUploadBuffers() 함수를 다음과 같이 수정한다.

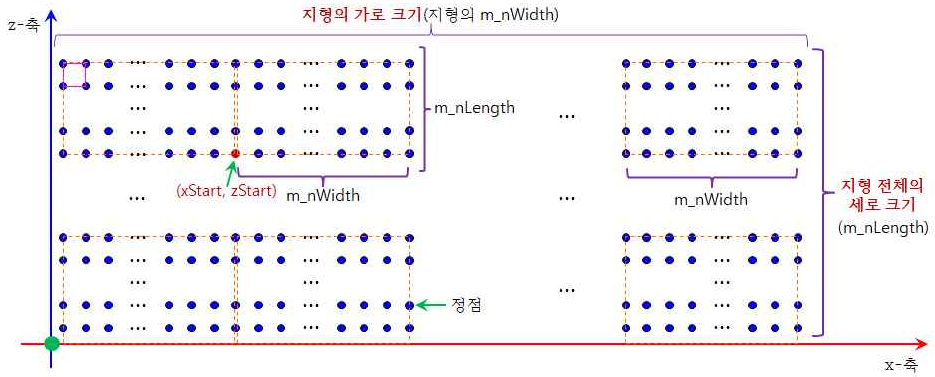
```
void CGameObject::ReleaseUploadBuffers()
{
   if (m_ppMeshes)
   {
      for (int i = 0; i < m_nMeshes; i++)
      {
         if (m_ppMeshes[i]) m_ppMeshes[i]->ReleaseUploadBuffers();
      }
   }
}
```

❺ “CRotatingObject” 클래스의 생성자를 다음과 같이 수정한다.

```
CRotatingObject::CRotatingObject(int nMeshes) : CGameObject(nMeshes)
{
   m_xmf3RotationAxis = XMFLOAT3(0.0f, 1.0f, 0.0f);
   m_fRotationSpeed = 15.0f;
}
```

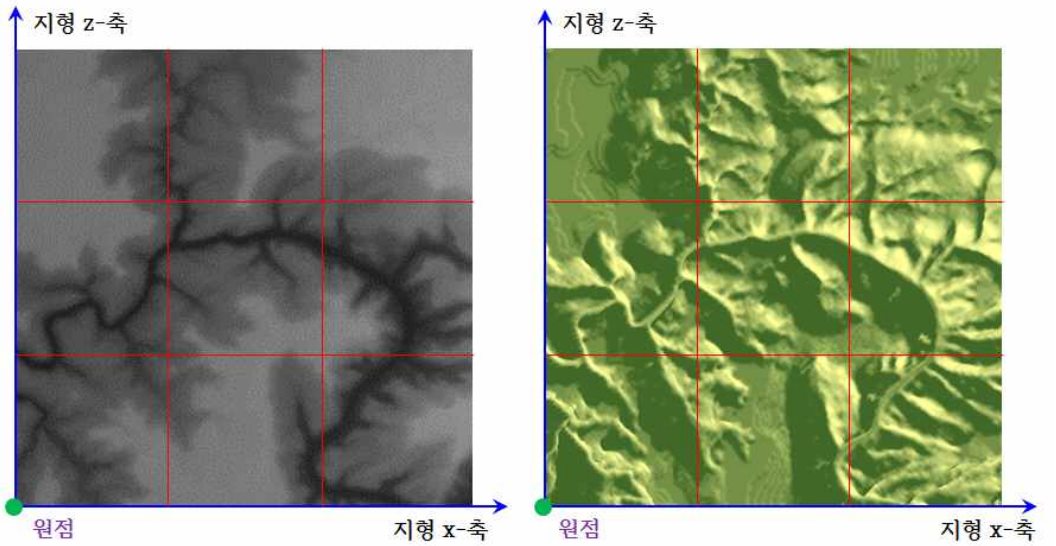
❻ “CHeightMapTerrain” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

지형의 (가로×세로) 크기가 (m×n)이라고 가정하자. 그리고 (m×n) 크기의 높이 맵(이미지)가 있다고 가정하자. 지형을 바둑판과 같은 커다란 격자라고 생각하자. 이 지형(격자)을 다음 그림과 같이 가로와 세로로 적당한 크기로 나눈다고 생각하자. 왜냐하면 지형이 아주 크다면 전체 지형을 한 번에 렌더링하는 것은 어렵기 때문이다(비디오 카드의 비디오 메모리의 양과 캐쉬(Cache) 메모리 크기에 따라 렌더링 시간이 달라질 수 있고 절두체 컬링에 영향을 줄 수 있다). 그러므로 한 번에 빠르게 그릴 수 있는 적당한 크기로 전체 지형을 나누도록 하자. 이렇게 작은 부분 지형들로 전체 지형을 나누면 전체 지형을 작은 지형(격자)들의 이차원 배열로 생각할 수 있다. 전체 지형을 렌더링하는 것은 이 작은 지형들을 모두 렌더링하는 것이다.



지형(작은 지형 메쉬들의 배열)

지형 전체를 xz-평면으로 적당한 크기로 나누어 각각을 메쉬(격자)로 표현하고 이들을 합하여 지형을 나타낼 것이다. 다음 그림은 지형을 (3×3)으로 나눈 경우를 보여준다.



전체 지형을 (3×3)의 작은 지형들로 나눈 경우

```
CHeightMapTerrain::CHeightMapTerrain(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, ID3D12RootSignature *pd3dGraphicsRootSignature, LPCTSTR pFileName, int
```

```
nWidth, int nLength, int nBlockWidth, int nBlockLength, XMFLOAT3 xmf3Scale, XMFLOAT4
xmf4Color) : CGameObject(0)
{
//지형에 사용할 높이 맵의 가로, 세로의 크기이다.
   m_nWidth = nWidth;
   m_nLength = nLength;

/*지형 객체는 격자 메쉬들의 배열로 만들 것이다. nBlockWidth, nBlockLength는 격자 메쉬 하나의 가로, 세로 크
기이다. cxQuadsPerBlock, czQuadsPerBlock은 격자 메쉬의 가로 방향과 세로 방향 사각형의 개수이다.*/
   int cxQuadsPerBlock = nBlockWidth - 1;
   int czQuadsPerBlock = nBlockLength - 1;

//xmf3Scale는 지형을 실제로 몇 배 확대할 것인가를 나타낸다.
   m_xmf3Scale = xmf3Scale;

//지형에 사용할 높이 맵을 생성한다.
   m_pHeightMapImage = new CHeightMapImage(pFileName, nWidth, nLength, xmf3Scale);

//지형에서 가로 방향, 세로 방향으로 격자 메쉬가 몇 개가 있는 가를 나타낸다.
   long cxBlocks = (m_nWidth - 1) / cxQuadsPerBlock;
   long czBlocks = (m_nLength - 1) / czQuadsPerBlock;

//지형 전체를 표현하기 위한 격자 메쉬의 개수이다.
   m_nMeshes = cxBlocks * czBlocks;
//지형 전체를 표현하기 위한 격자 메쉬에 대한 포인터 배열을 생성한다.
   m_ppMeshes = new CMesh*[m_nMeshes];
   for (int i = 0; i < m_nMeshes; i++) m_ppMeshes[i] = NULL;

   CHeightMapGridMesh *pHeightMapGridMesh = NULL;
   for (int z = 0, zStart = 0; z < czBlocks; z++)
   {
      for (int x = 0, xStart = 0; x < cxBlocks; x++)
      {
//지형의 일부분을 나타내는 격자 메쉬의 시작 위치(좌표)이다.
         xStart = x * (nBlockWidth - 1);
         zStart = z * (nBlockLength - 1);
//지형의 일부분을 나타내는 격자 메쉬를 생성하여 지형 메쉬에 저장한다.
         pHeightMapGridMesh = new CHeightMapGridMesh(pd3dDevice, pd3dCommandList, xStart,
zStart, nBlockWidth, nBlockLength, xmf3Scale, xmf4Color, m_pHeightMapImage);
         SetMesh(x + (z*cxBlocks), pHeightMapGridMesh);
      }
   }

//지형을 렌더링하기 위한 셰이더를 생성한다.
   CTerrainShader *pShader = new CTerrainShader();
   pShader->CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
   SetShader(pShader);
}

CHeightMapTerrain::~CHeightMapTerrain(void)
{
   if (m_pHeightMapImage) delete m_pHeightMapImage;
}
```

⑧ “Player.h” 파일 변경하기

❶ “CPlayer” 클래스의 생성자를 다음과 같이 수정한다.

```
CPlayer(int nMeshes = 1);
```

❷ “CAirplanePlayer” 클래스의 생성자를 다음과 같이 수정한다.

```
CAirplanePlayer(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCommandList,
ID3D12RootSignature *pd3dGraphicsRootSignature, int nMeshes = 1);
```

❸ “CTerrainPlayer” 클래스를 다음과 같이 선언한다.

```
class CTerrainPlayer : public CPlayer
{
public:
   CTerrainPlayer(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCommandList,
ID3D12RootSignature *pd3dGraphicsRootSignature, void *pContext, int nMeshes = 1);
   virtual ~CTerrainPlayer();

   virtual CCamera *ChangeCamera(DWORD nNewCameraMode, float fTimeElapsed);

   virtual void OnPlayerUpdateCallback(float fTimeElapsed);
   virtual void OnCameraUpdateCallback(float fTimeElapsed);
};
```

⑨ “Player.cpp” 파일 변경하기

❶ “CPlayer” 클래스의 생성자를 다음과 같이 수정한다.

```
CPlayer::CPlayer(int nMeshes) : CGameObject(nMeshes)
{
   ...
```

❷ “CAirplanePlayer” 클래스의 생성자를 다음과 같이 수정한다.

```
CAirplanePlayer::CAirplanePlayer(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, ID3D12RootSignature *pd3dGraphicsRootSignature, int nMeshes) :
CPlayer(nMeshes)
{
   CMesh *pAirplaneMesh = new CAirplaneMeshDiffused(pd3dDevice, pd3dCommandList, 20.0f,
20.0f, 4.0f, XMFLOAT4(0.5f, 0.0f, 0.0f, 0.0f));

   SetMesh(0, pAirplaneMesh);
   m_pCamera = ChangeCamera(SPACESHIP_CAMERA/*THIRD_PERSON_CAMERA*/, 0.0f);

   CreateShaderVariables(pd3dDevice, pd3dCommandList);
```

```
    SetPosition(XMFLOAT3(0.0f, 0.0f, -50.0f));

    CPlayerShader *pShader = new CPlayerShader();
    pShader->CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
    SetShader(pShader);
}
```

❸ “CTerrainPlayer” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CTerrainPlayer::CTerrainPlayer(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, ID3D12RootSignature *pd3dGraphicsRootSignature, void *pContext, int
nMeshes) : CPlayer(nMeshes)
{
    m_pCamera = ChangeCamera(THIRD_PERSON_CAMERA, 0.0f);

    CHeightMapTerrain *pTerrain = (CHeightMapTerrain *)pContext;
//플레이어의 위치를 지형의 가운데(y-축 좌표는 지형의 높이보다 1500 높게)로 설정한다. 플레이어 위치 벡터의 y-
좌표가 지형의 높이보다 크고 중력이 작용하도록 플레이어를 설정하였으므로 플레이어는 점차적으로 하강하게 된다.*/
    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));
//플레이어의 위치가 변경될 때 지형의 정보에 따라 플레이어의 위치를 변경할 수 있도록 설정한다.
    SetPlayerUpdatedContext(pTerrain);
//카메라의 위치가 변경될 때 지형의 정보에 따라 카메라의 위치를 변경할 수 있도록 설정한다.
    SetCameraUpdatedContext(pTerrain);

    CCubeMeshDiffused *pCubeMesh = new CCubeMeshDiffused(pd3dDevice, pd3dCommandList,
4.0f, 12.0f, 4.0f);
    SetMesh(0, pCubeMesh);

//플레이어를 렌더링할 셰이더를 생성한다.
    CPlayerShader *pShader = new CPlayerShader();
    pShader->CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
    SetShader(pShader);

    CreateShaderVariables(pd3dDevice, pd3dCommandList);
}

CTerrainPlayer::~CTerrainPlayer()
{
}
```

❹ “CTerrainPlayer” 클래스의 ChangeCamera() 함수를 다음과 같이 정의한다.

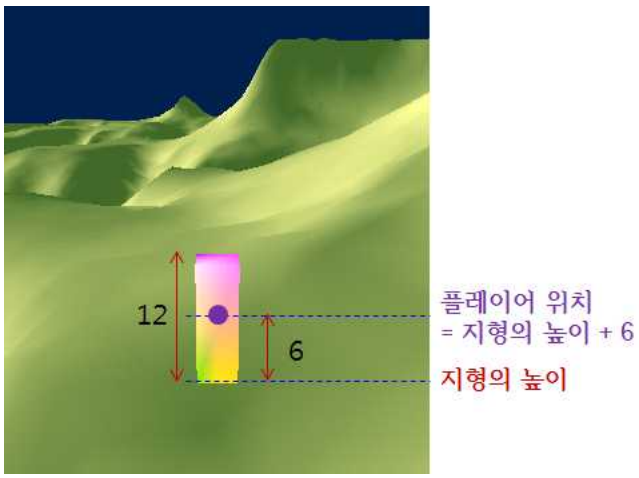
```
CCamera *CTerrainPlayer::ChangeCamera(DWORD nNewCameraMode, float fTimeElapsed)
{
    DWORD nCurrentCameraMode = (m_pCamera) ? m_pCamera->GetMode() : 0x00;
    if (nCurrentCameraMode == nNewCameraMode) return(m_pCamera);
    switch (nNewCameraMode)
    {
    case FIRST_PERSON_CAMERA:
```

```
        SetFriction(250.0f);
//1인칭 카메라일 때 플레이어에 y-축 방향으로 중력이 작용한다.
        SetGravity(XMFLOAT3(0.0f, -250.0f, 0.0f));
        SetMaxVelocityXZ(300.0f);
        SetMaxVelocityY(400.0f);
        m_pCamera = OnChangeCamera(FIRST_PERSON_CAMERA, nCurrentCameraMode);
        m_pCamera->SetTimeLag(0.0f);
        m_pCamera->SetOffset(XMFLOAT3(0.0f, 20.0f, 0.0f));
        m_pCamera->GenerateProjectionMatrix(1.01f, 50000.0f, ASPECT_RATIO, 60.0f);
        break;
    case SPACESHIP_CAMERA:
        SetFriction(125.0f);
 //스페이스 쉽 카메라일 때 플레이어에 중력이 작용하지 않는다.
        SetGravity(XMFLOAT3(0.0f, 0.0f, 0.0f));
        SetMaxVelocityXZ(300.0f);
        SetMaxVelocityY(400.0f);
        m_pCamera = OnChangeCamera(SPACESHIP_CAMERA, nCurrentCameraMode);
        m_pCamera->SetTimeLag(0.0f);
        m_pCamera->SetOffset(XMFLOAT3(0.0f, 0.0f, 0.0f));
        m_pCamera->GenerateProjectionMatrix(1.01f, 50000.0f, ASPECT_RATIO, 60.0f);
        break;
    case THIRD_PERSON_CAMERA:
        SetFriction(250.0f);
 //3인칭 카메라일 때 플레이어에 y-축 방향으로 중력이 작용한다.
        SetGravity(XMFLOAT3(0.0f, -250.0f, 0.0f));
        SetMaxVelocityXZ(300.0f);
        SetMaxVelocityY(400.0f);
        m_pCamera = OnChangeCamera(THIRD_PERSON_CAMERA, nCurrentCameraMode);
        m_pCamera->SetTimeLag(0.25f);
        m_pCamera->SetOffset(XMFLOAT3(0.0f, 20.0f, -50.0f));
        m_pCamera->GenerateProjectionMatrix(1.01f, 50000.0f, ASPECT_RATIO, 60.0f);
        break;
    default:
        break;
    }
    Update(fTimeElapsed);

    return(m_pCamera);
}
```

❺ “CTerrainPlayer” 클래스의 OnPlayerUpdateCallback() 함수를 다음과 같이 정의한다. 이 함수는 플레이어의 위치가 바뀔 때 마다 호출된다.

```
void CTerrainPlayer::OnPlayerUpdateCallback(float fTimeElapsed)
{
    XMFLOAT3 xmf3PlayerPosition = GetPosition();
    CHeightMapTerrain *pTerrain = (CHeightMapTerrain *)m_pPlayerUpdatedContext;
/*지형에서 플레이어의 현재 위치 (x, z)의 지형 높이(y)를 구한다. 그리고 플레이어 메쉬의 높이가 12이고 플레이어의 중심이 직육면체의 가운데이므로 y 값에 메쉬의 높이의 절반을 더하면 플레이어의 위치가 된다.*/
    float fHeight = pTerrain->GetHeight(xmf3PlayerPosition.x, xmf3PlayerPosition.z) +
6.0f;
/*플레이어의 위치 벡터의 y-값이 음수이면(예를 들어, 중력이 적용되는 경우) 플레이어의 위치 벡터의 y-값이 점점 작아지게 된다. 이때 플레이어의 현재 위치 벡터의 y 값이 지형의 높이(실제로 지형의 높이 + 6)보다 작으면 플레이어
```

```
의 일부가 지형 아래에 있게 된다. 이러한 경우를 방지하려면 플레이어의 속도 벡터의 y 값을 0으로 만들고 플레이어의 위치 벡터의 y-값을 지형의 높이(실제로 지형의 높이 + 6)로 설정한다. 그러면 플레이어는 항상 지형 위에 있게 된다.*/
    if (xmf3PlayerPosition.y < fHeight)
    {
        XMFLOAT3 xmf3PlayerVelocity = GetVelocity();
        xmf3PlayerVelocity.y = 0.0f;
        SetVelocity(xmf3PlayerVelocity);
        xmf3PlayerPosition.y = fHeight;
        SetPosition(xmf3PlayerPosition);
    }
}
```

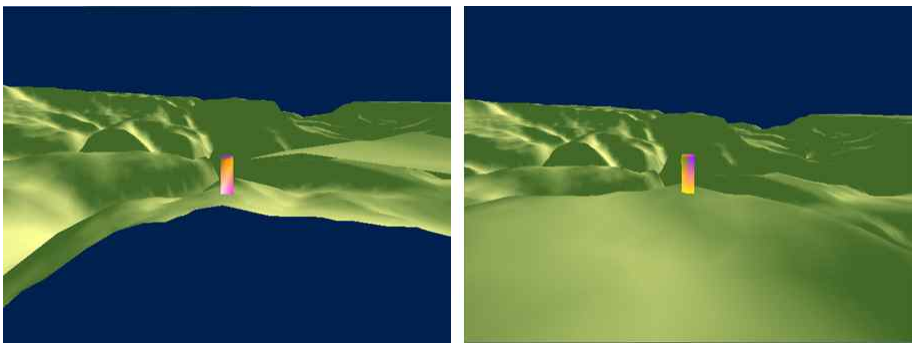


플레이어의 위치

❻ "CTerrainPlayer" 클래스의 OnCameraUpdateCallback() 함수를 다음과 같이 정의한다. 이 함수는 플레이어의 위치가 바뀔 때 마다 호출된다.

```
void CTerrainPlayer::OnCameraUpdateCallback(float fTimeElapsed)
{
    XMFLOAT3 xmf3CameraPosition = m_pCamera->GetPosition();
/*높이 맵에서 카메라의 현재 위치 (x, z)에 대한 지형의 높이(y 값)를 구한다. 이 값이 카메라의 위치 벡터의 y-값 보다 크면 카메라가 지형의 아래에 있게 된다. 이렇게 되면 다음 그림의 왼쪽과 같이 지형이 그려지지 않는 경우가 발생한다(카메라가 지형 안에 있으므로 삼각형의 와인딩 순서가 바뀐다). 이러한 경우가 발생하지 않도록 카메라의 위치 벡터의 y-값의 최소값은 (지형의 높이 + 5)로 설정한다. 카메라의 위치 벡터의 y-값의 최소값은 지형의 모든 위치에서 카메라가 지형 아래에 위치하지 않도록 설정해야 한다.*/
    CHeightMapTerrain *pTerrain = (CHeightMapTerrain *)m_pCameraUpdatedContext;
    float fHeight = pTerrain->GetHeight(xmf3CameraPosition.x, xmf3CameraPosition.z) +
5.0f;
    if (xmf3CameraPosition.y <= fHeight)
    {
        xmf3CameraPosition.y = fHeight;
        m_pCamera->SetPosition(xmf3CameraPosition);
        if (m_pCamera->GetMode() == THIRD_PERSON_CAMERA)
        {
//3인칭 카메라의 경우 카메라 위치(y-좌표)가 변경되었으므로 카메라가 플레이어를 바라보도록 한다.
            CThirdPersonCamera *p3rdPersonCamera = (CThirdPersonCamera *)m_pCamera;
            p3rdPersonCamera->SetLookAt(GetPosition());
```

```
        }
    }
}
```



카메라가 지형 아래에 있는 경우(왼쪽)와 보정한 경우(오른쪽)

⑩ “Scene.h” 파일 변경하기

❶ “CScene” 클래스에 다음을 추가한다.

```
protected:
    CHeightMapTerrain  *m_pTerrain = NULL;

public:
    CHeightMapTerrain *GetTerrain() { return(m_pTerrain); }
```

⑪ “Scene.cpp” 파일 변경하기

❶ “CScene” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

```
void CScene::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
    m_pd3dGraphicsRootSignature = CreateGraphicsRootSignature(pd3dDevice);

//지형을 확대할 스케일 벡터이다. x-축과 z-축은 8배, y-축은 2배 확대한다.
    XMFLOAT3 xmf3Scale(8.0f, 2.0f, 8.0f);
    XMFLOAT4 xmf4Color(0.0f, 0.2f, 0.0f, 0.0f);
//지형을 높이 맵 이미지 파일(HeightMap.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);
```

```
#endif

   m_nShaders = 1;
   m_pShaders = new CObjectsShader[m_nShaders];
   m_pShaders[0].CreateShader(pd3dDevice, m_pd3dGraphicsRootSignature);
   m_pShaders[0].BuildObjects(pd3dDevice, pd3dCommandList, m_pTerrain);
}
```

❷ “CScene” 클래스의 ReleaseObjects() 함수에 다음을 추가한다.

```
   if (m_pTerrain) delete m_pTerrain;
```

❸ “CScene” 클래스의 ReleaseUploadBuffers() 함수에 다음을 추가한다.

```
   if (m_pTerrain) m_pTerrain->ReleaseUploadBuffers();
```

❹ “CScene” 클래스의 Render() 함수를 다음과 같이 수정한다.

```
void CScene::Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera)
{
   pCamera->SetViewportsAndScissorRects(pd3dCommandList);
   pd3dCommandList->SetGraphicsRootSignature(m_pd3dGraphicsRootSignature);
   pCamera->UpdateShaderVariables(pd3dCommandList);

   if (m_pTerrain) m_pTerrain->Render(pd3dCommandList, pCamera);

   for (int i = 0; i < m_nShaders; i++)
   {
      m_pShaders[i].Render(pd3dCommandList, pCamera);
   }
}
```

⑫ “Shader.h” 파일 변경하기

❶ “CObjectsShader” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

```
   virtual void BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, void *pContext);
```

❷ “CTerrainShader” 클래스를 다음과 같이 선언한다.

```
class CTerrainShader : public CShader
{
public:
   CTerrainShader();
   virtual ~CTerrainShader();

   virtual D3D12_INPUT_LAYOUT_DESC CreateInputLayout();
   virtual D3D12_SHADER_BYTECODE CreateVertexShader(ID3DBlob **ppd3dShaderBlob);
```

```
   virtual D3D12_SHADER_BYTECODE CreatePixelShader(ID3DBlob **ppd3dShaderBlob);

   virtual void CreateShader(ID3D12Device *pd3dDevice, ID3D12RootSignature
*pd3dGraphicsRootSignature);
};
```

⑬ “Shader.cpp” 파일 변경하기

❶ “CObjectsShader” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

```
void CObjectsShader::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, void *pContext)
{
   CHeightMapTerrain *pTerrain = (CHeightMapTerrain *)pContext;
   float fTerrainWidth = pTerrain->GetWidth(), fTerrainLength = pTerrain->GetLength();

   float fxPitch = 12.0f * 3.5f;
   float fyPitch = 12.0f * 3.5f;
   float fzPitch = 12.0f * 3.5f;
//직육면체를 지형 표면에 그리고 지형보다 높은 위치에 일정한 간격으로 배치한다.
   int xObjects = int(fTerrainWidth / fxPitch), yObjects = 2, zObjects =
int(fTerrainLength / fzPitch);
   m_nObjects = xObjects * yObjects * zObjects;
   m_ppObjects = new CGameObject*[m_nObjects];

   CCubeMeshDiffused *pCubeMesh = new CCubeMeshDiffused(pd3dDevice, pd3dCommandList,
12.0f, 12.0f, 12.0f);

   XMFLOAT3 xmf3RotateAxis, xmf3SurfaceNormal;
   CRotatingObject *pRotatingObject = NULL;
   for (int i = 0, x = 0; x < xObjects; x++)
   {
      for (int z = 0; z < zObjects; z++)
      {
         for (int y = 0; y < yObjects; y++)
         {
            pRotatingObject = new CRotatingObject(1);
            pRotatingObject->SetMesh(0, pCubeMesh);
            float xPosition = x * fxPitch;
            float zPosition = z * fzPitch;
            float fHeight = pTerrain->GetHeight(xPosition, zPosition);
            pRotatingObject->SetPosition(xPosition, fHeight + (y * 10.0f * fyPitch) +
6.0f, zPosition);
            if (y == 0)
            {
/*지형의 표면에 위치하는 직육면체는 지형의 기울기에 따라 방향이 다르게 배치한다. 직육면체가 위치할 지형의 법선
벡터 방향과 직육면체의 y-축이 일치하도록 한다.*/
               xmf3SurfaceNormal = pTerrain->GetNormal(xPosition, zPosition);
               xmf3RotateAxis = Vector3::CrossProduct(XMFLOAT3(0.0f, 1.0f, 0.0f),
xmf3SurfaceNormal);
               if (Vector3::IsZero(xmf3RotateAxis)) xmf3RotateAxis = XMFLOAT3(0.0f, 1.0f,
0.0f);
               float fAngle = acos(Vector3::DotProduct(XMFLOAT3(0.0f, 1.0f, 0.0f),
xmf3SurfaceNormal));
```

```
                pRotatingObject->Rotate(&xmf3RotateAxis, XMConvertToDegrees(fAngle));
            }
            pRotatingObject->SetRotationAxis(XMFLOAT3(0.0f, 1.0f, 0.0f));
            pRotatingObject->SetRotationSpeed(36.0f * (i % 10) + 36.0f);
            m_ppObjects[i++] = pRotatingObject;
        }
    }
  }

  CreateShaderVariables(pd3dDevice, pd3dCommandList);
}
```

❷ “CTerrainShader” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CTerrainShader::CTerrainShader()
{
}

CTerrainShader::~CTerrainShader()
{
}
```

❸ “CTerrainShader” 클래스의 CreateInputLayout() 함수를 다음과 같이 정의한다.

```
D3D12_INPUT_LAYOUT_DESC CTerrainShader::CreateInputLayout()
{
  UINT nInputElementDescs = 2;
  D3D12_INPUT_ELEMENT_DESC *pd3dInputElementDescs = new
D3D12_INPUT_ELEMENT_DESC[nInputElementDescs];

  pd3dInputElementDescs[0] ={ "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 };
  pd3dInputElementDescs[1] ={ "COLOR", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 12,
D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 };

  D3D12_INPUT_LAYOUT_DESC d3dInputLayoutDesc;
  d3dInputLayoutDesc.pInputElementDescs = pd3dInputElementDescs;
  d3dInputLayoutDesc.NumElements = nInputElementDescs;

  return(d3dInputLayoutDesc);
}
```

❹ “CTerrainShader” 클래스의 CreateVertexShader(), CreatePixelShader(), CreateShader() 함수를 다음과 같이 정의한다.

```
D3D12_SHADER_BYTECODE CTerrainShader::CreateVertexShader(ID3DBlob **ppd3dShaderBlob)
{
  return(CShader::CompileShaderFromFile(L"Shaders.hlsl", "VSDiffused", "vs_5_1",
ppd3dShaderBlob));
}
```

```
D3D12_SHADER_BYTECODE CTerrainShader::CreatePixelShader(ID3DBlob **ppd3dShaderBlob)
{
   return(CShader::CompileShaderFromFile(L"Shaders.hlsl", "PSDiffused", "ps_5_1",
ppd3dShaderBlob));
}

void CTerrainShader::CreateShader(ID3D12Device *pd3dDevice, ID3D12RootSignature
*pd3dGraphicsRootSignature)
{
   m_nPipelineStates = 1;
   m_ppd3dPipelineStates = new ID3D12PipelineState*[m_nPipelineStates];

   CShader::CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
}
```

⑭ “GameFramework.cpp” 파일 수정하기

❶ “CGameFramework” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

```
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(m_pd3dDevice, m_pd3dCommandList,
m_pScene->GetGraphicsRootSignature(), m_pScene->GetTerrain(), 1);
   m_pCamera = m_pPlayer->GetCamera();

   m_pd3dCommandList->Close();
   ID3D12CommandList *ppd3dCommandLists[] = { m_pd3dCommandList };
   m_pd3dCommandQueue->ExecuteCommandLists(1, ppd3dCommandLists);

   WaitForGpuComplete();
   if (m_pScene) m_pScene->ReleaseUploadBuffers();

   m_GameTimer.Reset();
}
```