Terrain

305890
2009년 봄학기
5/20/2009
박경신

Terrain Mesh

- 지형메쉬 (Terrain Mesh)
  - 격자 내에 버텍스에 높이를 부여하여 생성된 메쉬
  - 실제 지형과 같이 산에서 계곡으로의 부드러운 전환 가능
  - 텍스처를 추가하면, 모래로 덮인 해변이나 풀로 덮인 언덕, 눈 덮인 산 등의 표현 가능
- Terrain 클래스 구현
  - 아주 작은 지형을 이용하는 게임의 경우, 버텍스 프로세싱을 지원하는 그래픽 카드를 이용해 충분히 적용 가능

Basic Terrain Rendering

- Heightmaps
  - 산이나 계곡과 같은 자연스러운 지형을 표현하기 위해 지형의 높이 정보 생성 방법
- Generating the Terrain Geometry
  - 지형을 위한 버텍스와 삼각형 데이터의 생성 방법
- Texturing & Lighting
  - 지형에 조명과 텍스처의 적용 방법
- Walking on the Terrain
  - 카메라를 지형 표면에 위치시키고, 지형을 따라 걷거나 뛰는 방법
- Terrain 예제

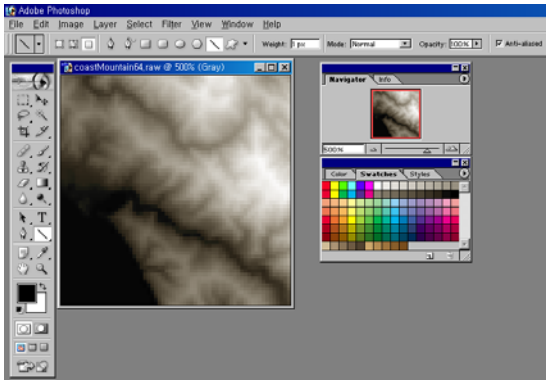
Heightmap

- 높이맵 (heightmap)
  - 지형 격자 (terrain grid) 내의 정점의 높이 값을 요소로 가지고 있는 배열 (높이맵-지형의 특정 정점 높이 일대일 대응관계)
  - 보통 각 요소(element)에 한 바이트(byte)의 메모리 할당 (0~255)
    - 필요에 따라 원하는 배율로 값을 조정
    - Gray-scale map (0: 낮은 고도, 255: 높은 고도)
  - 한 배율 값을 사용하여 데이터 읽기 시점에서 실제 높이를 조정

Gray-scale로 표현된 높이맵

Creating Heightmap

- 이미지 편집기 사용 (예: Adobe Photoshop)
  - 이미지 포맷을 gray-scale로 지정해야 함
  - 편집 후 8-bit RAW 파일로 저장하는 것이 편리함 (RAW 파일은 무압축으로 순차적으로 저장함. 헤더 없이 저장할 것.)



Loading RAW Heightmap File

- Byte 로 읽어서 integer로 바꿈 (0 .. 255 범위 밖으로 배율을 변경할 수 있도록)
- Byte 수 만큼의 정점을 생성함
  - 256x256 RAW 파일이면 256x256 vertex의 지형을 구성함
  - 정점 수 보다 RAW 데이터의 byte수가 작으면 안됨.

Loading RAW Heightmap File

```
std::vector<int> _heightmap; /* terrain.h */
bool Terrain::readRawFile(std::string fileName) /* terrain.cpp */
{
    // Restriction: RAW file dimensions must be >= to the dimensions of the terrain.
    // That is a 128x128 RAW file can only be used with a terrain constructed with
    // at most 128x128 vertices.
    // A height for each vertex
    std::vector<BYTE> in( _numVertices );

    std::ifstream inFile(fileName.c_str(), std::ios_base::binary);

    if( inFile == 0 ) return false;

    inFile.read((char*)&in[0], // buffer
            in.size());// number of bytes to read into buffer

    inFile.close();

    // copy BYTE vector to INT vector
    _heightmap.resize( _numVertices );

    for(int i = 0; i < in.size(); i++)
            _heightmap[i] = in[i];

    return true;
}
```

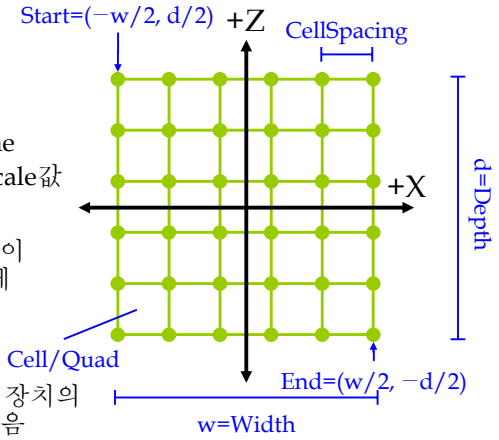
Accessing and Modifying Heightmap

- 행과 열을 지정하여 항목을 참조

```
int Terrain::getHeightmapEntry(int row, int col)
{
    return _heightmap[row * _numVertsPerRow + col];
}


void Terrain::setHeightmapEntry(int row, int col, int value)
{
    _heightmap[row * _numVertsPerRow + col] = value;
}
```

Generating the Terrain Geometry

- 지형의 크기 정의
  - # of cells per row
  - # of cells per column
  - Cell spacing
  - Heightmap data filename
  - 배율 조정을 위한 높이 scale값
- 주의
  - 지형을 위한 모든 정점들이 하나의 큰 vertex buffer에 보관됨
  - D3DCAPS9 구조체의 MaxPrimitiveCount, MaxVertexIndex를 통해 장치의 한계를 확인할 필요가 있음



Generating the Terrain Geometry

```
class Terrain {
public:
    Terrain(IDirect3DDevice9* device,    std::string heightmapFileName,
            int numVertsPerRow,  int numVertsPerCol, int cellSpacing,
            float heightScale);
    ~Terrain();
    float getHeight(float x, float z);
    bool loadTexture(std::string fileName);
    bool genTexture(D3DXVECTOR3* directionToLight);
    bool draw(D3DXMATRIX* world, bool drawTris);

private:
    IDirect3DDevice9* _device;          IDirect3DTexture9* _tex;
    IDirect3DVertexBuffer9* _vb;        IDirect3DIndexBuffer9* _ib;

    int _numVertsPerRow;    int _numVertsPerCol;        int _cellSpacing;
    int _numCellsPerRow;    int _numCellsPerCol;
    int _width;             int _depth;
    int _numVertices;       int _numTriangles;          float _heightScale;

    bool computeVertices();    bool computeIndices();
    bool lightTerrain(D3DXVECTOR3* directionToLight);
    float computeShade(int cellRow, int cellCol, D3DXVECTOR3* directionToLight);

    // TerrainVertex structure

    std::vector<int> _heightmap;
};
```

TerrainVertex Structure

- TerrainVertex 구조체
  - Terrain 클래스 외부에서는 필요치 않기 때문에 클래스 내부에 중첩되게 정의함

```
struct TerrainVertex
{
   TerrainVertex(){}
   TerrainVertex(float x, float y, float z, float u, float v)
   {
    _x = x; _y = y; _z = z; _u = u; _v = v;
   }
   float _x, _y, _z;
   float _u, _v;

   static const DWORD FVF;
};

const DWORD Terrain::TerrainVertex::FVF = D3DFVF_XYZ | D3DFVF_TEX1;
```

Terrain Constructor

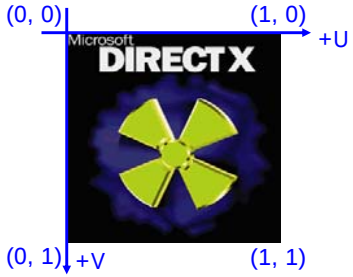
```
Terrain::Terrain(IDirect3DDevice9* device,      std::string heightmapFileName,
                 int numVertsPerRow,  int numVertsPerCol,  int cellSpacing,  float heightScale)
{
    _device       = device;
    _numVertsPerRow = numVertsPerRow;
    _numVertsPerCol = numVertsPerCol;
    _cellSpacing    = cellSpacing;
    _numCellsPerRow = _numVertsPerRow - 1;
    _numCellsPerCol = _numVertsPerCol - 1;
    _width = _numCellsPerRow * _cellSpacing;
    _depth = _numCellsPerCol * _cellSpacing;
    _numVertices  = _numVertsPerRow * _numVertsPerCol;
    _numTriangles = _numCellsPerRow * _numCellsPerCol * 2;
    _heightScale = heightScale;

    if (!readRawFile(heightmapFileName)) { // load heightmap
       ::MessageBox(0, "readRawFile – FAILED", 0, 0);
       ::PostQuitMessage(0); }
    for (int i = 0; i < _heightmap.size(); i++) // scale heights
       _heightmap[i] *= heightScale;
    if (!computeVertices()) { // compute the vertices
       ::MessageBox(0, "computeVertices – FAILED", 0, 0);
       ::PostQuitMessage(0); }
    if (!computeIndices()) { // compute the indices
       ::MessageBox(0, "computeIndices – FAILED", 0, 0);
       ::PostQuitMessage(0); }
}
```

Computing Terrain Vertex

- (x, y, z) 좌표
  - x 와 z 좌표: 시작점(start)에서 버텍스를 생성하여 끝점(end)에 이를 때까지 셀 간격(cell spacing)만큼 간격을 비우면서 버텍스들 생성
  - y 좌표: 읽어 들인 높이맵(hightmap) 데이터 구조체 내의 대응되는 항목에서 얻음
- 텍스처 좌표 (u, v)

$$u = i \cdot uCoordIncrementSize$$
$$v = j \cdot vCoordIncrementSize$$

$$uCoordIncrementSize = \frac{1}{numCellCols}$$
$$vCoordIncrementSize = \frac{1}{numCellRows}$$



Computing Terrain Vertex

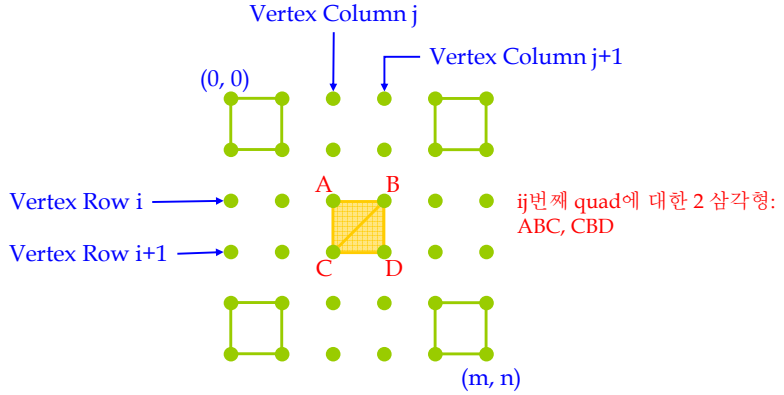
```
bool Terrain::computeVertices()
{
    HRESULT hr = 0;

    hr = _device->CreateVertexBuffer(
            _numVertices * sizeof(TerrainVertex),
            D3DUSAGE_WRITEONLY, TerrainVertex::FVF,
            D3DPOOL_MANAGED, &_vb, 0);

    if(FAILED(hr))
            return false;

    // coordinates to start generating vertices at
    int startX = -_width / 2;
    int startZ =  _depth / 2;

    // coordinates to end generating vertices at
    int endX =  _width / 2;
    int endZ = -_depth / 2;

    // compute the increment size of the texture coordinates
    // from one vertex to the next.
    float uCoordIncrementSize = 1.0f / (float)_numCellsPerRow;
    float vCoordIncrementSize = 1.0f / (float)_numCellsPerCol;
```

Computing Terrain Vertex

```
    TerrainVertex* v = 0;
    _vb->Lock(0, 0, (void**)&v, 0);

    int i = 0;
    for(int z = startZ; z >= endZ; z -= _cellSpacing) {
        int j = 0;
        for(int x = startX; x <= endX; x += _cellSpacing)  {
            // 현재 vertex의 vertex buffer와 heightmap에서의 index를 계산
            int index = i * _numVertsPerRow + j;

            v[index] = TerrainVertex(
                        (float)x,
                        (float)_heightmap[index],
                        (float)z,
                        (float)j * uCoordIncrementSize,
                        (float)i * vCoordIncrementSize);

            j++; // next column
        }
        i++; // next row
    }

    _vb->Unlock();

    return true;
}
```

Computing Terrain Indices

- 각 사각형을 구성하는 두 개의 삼각형을 차례로 계산



$$\Delta ABC = \{i \cdot numVertsPerRow + j,\ i \cdot numVertsPerRow + j + 1,\ (i+1) \cdot numVertsPerRow + j\}$$
$$\Delta CBD = \{(i+1) \cdot numVertsPerRow + j,\ i \cdot numVertsPerRow + j + 1,\ (i+1) \cdot numVertsPerRow + j + 1\}$$

Computing Terrain Indices

```
bool Terrain::computeIndices()
{
    HRESULT hr = 0;

    hr = _device->CreateIndexBuffer(
            _numTriangles * 3 * sizeof(WORD), // 3 indices per tri.
            D3DUSAGE_WRITEONLY,    D3DFMT_INDEX16,
            D3DPOOL_MANAGED,       &_ib,       0);

    if(FAILED(hr))
            return false;
```

Computing Terrain Indices

```
    WORD* indices = 0;
    _ib->Lock(0, 0, (void**)&indices, 0);

    // 1 quad를 만드는 2개의 삼각형 (즉, 6개의 index)으로 구성함
    int baseIndex = 0;

    // loop through and compute the triangles of each quad
    for(int i = 0; i < _numCellsPerCol; i++)
    {
            for(int j = 0; j < _numCellsPerRow; j++)
            {
                indices[baseIndex]     =   i   * _numVertsPerRow + j;
                indices[baseIndex + 1] =   i   * _numVertsPerRow + j + 1;
                indices[baseIndex + 2] = (i+1) * _numVertsPerRow + j;

                indices[baseIndex + 3] = (i+1) * _numVertsPerRow + j;
                indices[baseIndex + 4] =   i   * _numVertsPerRow + j + 1;
                indices[baseIndex + 5] = (i+1) * _numVertsPerRow + j + 1;

                // next quad
                baseIndex += 6;
            }
    }

    _ib->Unlock();
    return true;
}
```

Terrain Texturing

- 지형에 텍스처를 입히는 두 가지 방법
  1. 미리 만들어둔 텍스처 파일을 읽어 들여 이용
  2. 절차적으로 텍스처를 계산
- 텍스처 파일로부터 _tex 데이터 멤버로 텍스처를 읽어 들임

```
bool Terrain::loadTexture(std::string fileName)
{
    HRESULT hr = 0;

    hr = D3DXCreateTextureFromFile(_device, fileName.c_str(), &_tex);

    if(FAILED(hr)) return false;

    return true;
}
```

Terrain Texturing

- 절차적 방식 (Terrain::genTexture)
  - "빈" 텍스처를 만든다 – D3DXCreateTexture
  - 인자를 바탕으로 텍셀 (texture cell, texel)의 색(color)를 계산
    - 예) 지형의 높이를 인자로 이용
      - 낮은 고도의 지형 - 모래 해변 색상
      - 중간 고도의 지형 - 풀 덮인 언덕 색상
      - 높은 고도의 지형 - 눈 덮인 산 색상
  - 방법
    - 빈 텍스처를 만들고 최상위 레벨(밉맵)을 잠금
    - 각 사각형의 높이에 따라 텍셀의 컬러를 결정
    - 텍셀을 비추는 빛의 각도에 따라 텍셀의 밝기를 조정 ➔ 조명 (Terrain::lightTerrain)
    - 하위 밉맵 레벨의 텍셀을 계산 – D3DXFilterTexture

Terrain Texturing

```
bool Terrain::genTexture(D3DXVECTOR3* directionToLight)
{
    // Method fills the top surface of a texture procedurally.  Then
    // lights the top surface.  Finally, it fills the other mipmap
    // surfaces based on the top surface data using D3DXFilterTexture.
    HRESULT hr = 0;

    // texel for each quad cell
    int texWidth  = _numCellsPerRow;
    int texHeight = _numCellsPerCol;

    // create an empty texture
    hr = D3DXCreateTexture(_device,
            texWidth, texHeight,
            0, // create a complete mipmap chain
            0, // usage
            D3DFMT_X8R8G8B8,// 32 bit XRGB format
            D3DPOOL_MANAGED, &_tex);

    if(FAILED(hr)) return false;

    D3DSURFACE_DESC textureDesc;
    _tex->GetLevelDesc(0 /*level*/, &textureDesc);

    // make sure we got the requested format because our code
    // that fills the texture is hard coded to a 32 bit pixel depth.
    if( textureDesc.Format != D3DFMT_X8R8G8B8 ) return false;
```

Terrain Texturing

```
    D3DLOCKED_RECT lockedRect;
    _tex->LockRect(0 /*lock top surface*/, &lockedRect,
                   0 /*lock entire tex*/, 0 /*flags*/);

    DWORD* imageData = (DWORD*)lockedRect.pBits;
    for(int i = 0; i < texHeight; i++) {
      for(int j = 0; j < texWidth; j++)  {
            D3DXCOLOR c;

            // get height of upper left vertex of quad.
            float height = (float)getHeightmapEntry(i, j) / _heightScale;

            if( (height) < 42.5f )          c = d3d::BEACH_SAND;
            else if( (height) < 85.0f )     c = d3d::LIGHT_YELLOW_GREEN;
            else if( (height) < 127.5f )    c = d3d::PUREGREEN;
            else if( (height) < 170.0f )    c = d3d::DARK_YELLOW_GREEN;
            else if( (height) < 212.5f )    c = d3d::DARKBROWN;
            else                            c = d3d::WHITE;

            // fill locked data, note we divide the pitch by four because the
            // pitch is given in bytes and there are 4 bytes per DWORD.
            imageData[i * lockedRect.Pitch / 4 + j] = (D3DCOLOR)c;
      }
    }

    _tex->UnlockRect(0);
```

Terrain Texturing

```
    if(!lightTerrain(directionToLight))
    {
            ::MessageBox(0, "lightTerrain() - FAILED", 0, 0);
            return false;
    }

    hr = D3DXFilterTexture(_tex,
            0, // default palette
            0, // use top level as source level
            D3DX_DEFAULT); // default filter

    if(FAILED(hr))
    {
            ::MessageBox(0, "D3DXFilterTexture() - FAILED", 0, 0);
            return false;
    }

    return true;
}
```

Terrain Texturing

```
Bool Terrain::lightTerrain(D3DXVECTOR3* directionToLight)
{
    HRESULT hr = 0;
    D3DSURFACE_DESC textureDesc;
    _tex->GetLevelDesc(0 /* level */, &textureDesc);
    if(textureDesc.Format != D3DFMT_X8R8G8B8)
            return false;
    D3DLOCKED_RECT lockedRect;
    _tex->LockRect(0, &lockedRect, 0, 0);

    DWORD* imageData = (DWORD*) lockedRect.pBits;
    for (int i = 0; i < textureDesc.Height; i++) {
        for (int j = 0; j < textureDesc.Width; j++) {
            int index = i * lockedRect.Pitch / 4 + j;
            D3DXCOLOR c(imageData[index]); // get current color of quad
            c *= computeShade(i, j, directionToLight); // shade current quad
            imageData[index] = (D3DCOLOR) c; // save shaded color
        }
    }

    _tex->UnlockedRect(0);
    return true;
}
```

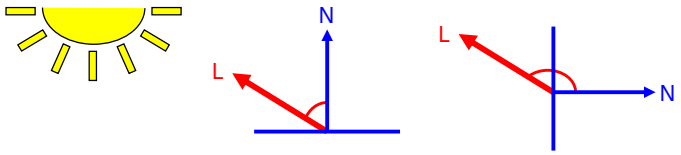
Lighting

- 광원에 따른 지형의 음영 계산
  - Terrain::genTexture는 지형에 조명을 추가하여 사실감을 더하고자 Terrain::lightTerrain을 호출함
  - 지형의 color는 이미 계산되어 있으므로, 지형의 음영만 계산하면 됨

- Direct3D에게 맡기지 않고 직접하는 세 가지 이유
  - 버텍스 법선 (vertex normal)을 저장하지 않아도 되므로 메모리 절약
  - Direct3D가 지형에 실시간으로 조명을 처리하는 부담을 덜 수 있음
    - 지형은 정적이며 조명을 움직이지 않을 것이므로, 미리 조명 계산이 가능
  - 약간의 수학 연습을 할 수 있으며, 기본적인 조명의 개념과 Direct3D 함수에 익숙해질 수 있음

Lighting

- "난반사광" (diffuse lighting)
  - 방향성 광원 (빛의 방향이 평행한 평행 광원)
    - 빛이 발산되는 방향: lightRaysDirection = (0, -1, 0)
    - 빛의 위치로의 방향 (빛을 향하는 벡터 L): directionToLight = (0, 1, 0)
  - 각 표면에서의 빛의 양은 입사각 (빛을 향하는 벡터 L과 표면의 법선 벡터 N 사이의 각도)를 계산
    - [0, 1] 범위의 음영 스칼라 – 표면이 받는 빛의 양
    - 컬러에 음영 스칼라를 곱하면, 어두운 또는 밝은 값이 만들어짐



Shading

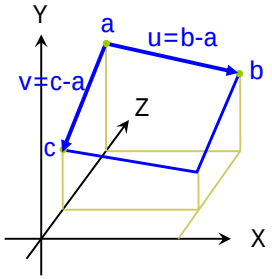
- 정규화된 빛을 향하는 벡터 L 과 사각형의 법선 벡터 N 사이의 각도
  - 외적을 이용하여 벡터 **N**계산

$$\mathbf{u} = (cellSpacing,\ b_y - a_y,\ 0)$$

$$\mathbf{v} = (0,\ c_y - a_y,\ cellSpacing)$$

$$\mathbf{N} = \mathbf{u} \times \mathbf{v}$$

$$\mathbf{N} = \frac{\mathbf{N}}{\|\mathbf{N}\|}$$



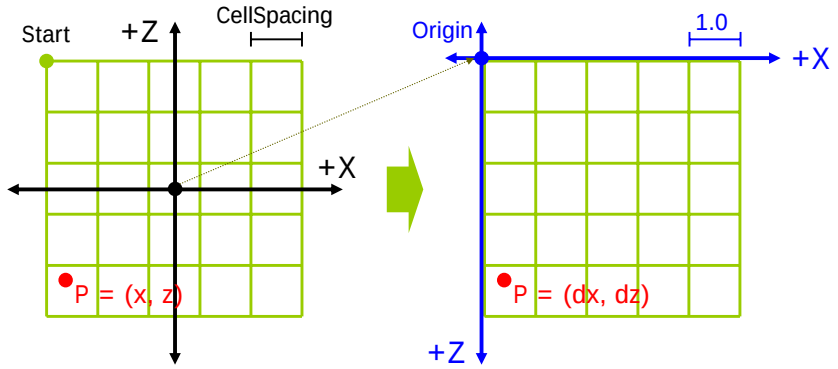
  - 입사각 계산: $s = L \cdot N$ (두 벡터간 각도의 cosine임)
    - s가 [-1, 1] 범위이나 [-1, 0)인 경우는 빛을 받지 못하므로 s=0 으로 하여 s가 [0, 1] 범위가 되도록 함: s[-1, 1] ➔ [0, 1]

```
float cosine = D3DXVec3Dot(&n, directionToLight);
if(cosine < 0.0f) cosine = 0.0f;
```

Shading

```
float Terrain::computeShade(int cellRow, int cellCol, D3DXVECTOR3* directionToLight)
{
    // get heights of three vertices on the quad
    float heightA = getHeightmapEntry(cellRow,   cellCol);
    float heightB = getHeightmapEntry(cellRow,   cellCol+1);
    float heightC = getHeightmapEntry(cellRow+1, cellCol);

    // build two vectors on the quad
    D3DXVECTOR3 u(_cellSpacing, heightB - heightA, 0.0f);
    D3DXVECTOR3 v(0.0f,         heightC - heightA, -_cellSpacing);

    // find the normal by taking the cross product of two
    // vectors on the quad.
    D3DXVECTOR3 n;
    D3DXVec3Cross(&n, &u, &v);
    D3DXVec3Normalize(&n, &n);

    float cosine = D3DXVec3Dot(&n, directionToLight);

    if(cosine < 0.0f)  cosine = 0.0f;

    return cosine;
}
```

Walking on the Terrain



- start → origin
- cell spacing → 1.0
- -z → +z

Walking on the Terrain

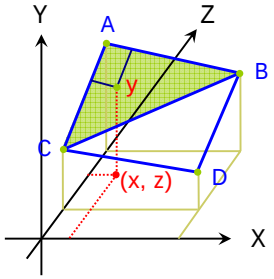
- 지형에 따라 카메라의 높이(y 좌표)를 조정
  - 카메라의 위치의 x와 z좌표를 이용하여, 카메라가 놓인 셀을 찾아내서 높이를 얻어냄

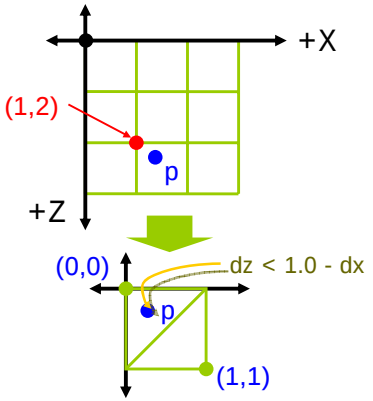
```
float Terrain::getHeight(float x, float z) {
    // xz-평면 상에서, terrain의 start위치를 원점으로 이동한다.
    x = ((float)_width / 2.0f) + x;
    z = ((float)_depth / 2.0f) - z;

    // cellspacing을 1로 scaling한다 (1 / cellspacing)
    x /= (float)_cellSpacing;
    z /= (float)_cellSpacing;

    // row, col 계산이 쉽도록, +z축이 반대방향 (아래방향)이라고 생각하자.
    int col = ::floorf(x);
    int row = ::floorf(z);

    // 현재 있는 곳에 해당하는 quad의 높이를 얻는다.
    float A = getHeightmapEntry(row,   col);
    float B = getHeightmapEntry(row,   col+1);
    float C = getHeightmapEntry(row+1, col);
    float D = getHeightmapEntry(row+1, col+1);
```

floor(*t*) = 가장 큰 정수 ≤ *t*
셀을 구성하는 네 버텍스의 높이를 얻어 낼 수 있음

Walking on the Terrain

- 셀의 어떤 삼각형에 있는지?

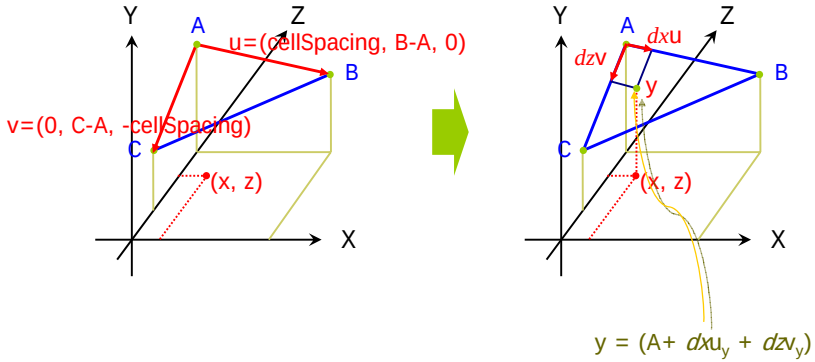





```
// 현재 있는 곳의 triangle을 찾는다.
// 현재 있는 곳의 cell의 upper left corner를 원점으로 이동한다.
// 따라서 cellspacing이 1로 되어 있으므로, +x/+z가 오른쪽/아래이며
// 원점이 정사각형이 있게 된다.
// +x -> 'right' and +z -> 'down' system.
float dx = x - col;
float dz = z - row;
```

Walking on the Terrain

- 선형 보간 (linear interpolation)

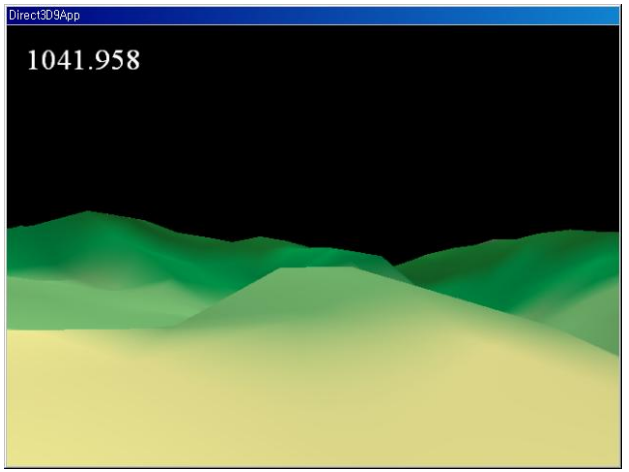


Walking on the Terrain

```
    // u, v 벡터 계산
    float height = 0.0f;
    if(dz < 1.0f - dx)  // upper triangle ABC
    {
        float uy = B - A; // A->B
        float vy = C - A; // A->C

    // Linearly interpolate on each vector.  The height is the vertex
    // height the vectors u and v originate from {A}, plus the heights
    // found by interpolating on each vector u and v.
        height = A + d3d::Lerp(0.0f, uy, dx) + d3d::Lerp(0.0f, vy, dz);
    }
    else // lower triangle DCB
    {
        float uy = C - D; // D->C
        float vy = B - D; // D->B

    // Linearly interpolate on each vector.  The height is the vertex
    // height the vectors u and v originate from {D}, plus the heights
    // found by interpolating on each vector u and v.
        height = D + d3d::Lerp(0.0f, uy, 1.0f-dx) + d3d::Lerp(0.0f, vy, 1.0f-dz);
    }
  return height;
}
```

```
float d3d::Lerp(float a, float b, float t)
{
        return a - (a*t) + (b*t);
}
```

Example: Terrain



Example: Terrain

```
Terrain* TheTerrain = 0;
Camera   TheCamera(Camera::LANDOBJECT);
FPSCounter* FPS = 0;
bool Setup() {
    // Create the terrain.
    D3DXVECTOR3 lightDirection(0.0f, 1.0f, 0.0f);
    TheTerrain = new Terrain(Device, "coastMountain64.raw", 64, 64, 10, 0.5f);
    TheTerrain->genTexture(&lightDirection);

    // Set texture filters.
    Device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

    // Set projection matrix.
    D3DXMATRIX proj;
    D3DXMatrixPerspectiveFovLH(
                    &proj,
                    D3DX_PI * 0.25f, // 45 - degree
                    (float)Width / (float)Height,
                    1.0f,
                    1000.0f);
    Device->SetTransform(D3DTS_PROJECTION, &proj);

    return true;
}
void Cleanup() {
    d3d::Delete<Terrain*>(TheTerrain);
    d3d::Delete<FPSCounter*>(FPS);
}
```

Example: Terrain

```
bool Display(float timeDelta) {
    if( Device ) {
        ... // input checking
        D3DXVECTOR3 pos;
        TheCamera.getPosition(&pos);
        float height = TheTerrain->getHeight( pos.x, pos.z );
        pos.y = height + 5.0f; // add height because we're standing up
        TheCamera.setPosition(&pos);

        D3DXMATRIX V;
        TheCamera.getViewMatrix(&V);
        Device->SetTransform(D3DTS_VIEW, &V);

        // Draw the scene
        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xff000000, 1.0f, 0);
        Device->BeginScene();

        D3DXMATRIX I;
        D3DXMatrixIdentity(&I);

        if( TheTerrain )
            TheTerrain->draw(&I, false);

        if( FPS )
            FPS->render(0xffffffff, timeDelta);

        Device->EndScene();
        Device->Present(0, 0, 0, 0);
    }
    return true;
}
```