스노우보드 게임을 위한 3D Height Map Tool 구현

김은주
한림대학교 컴퓨터공학과
e-mail : ejkim628@hallym.ac.kr

Implementation of 3D Height Map Tool for SnowBoard Game

Eun-Ju Kim
Dept. of Computer Engineering, Hallym University

요 약

체감형 게임을 하기 위한 지형 인터페이스 모델로써 스노우보드 게임에서 사용되는 지형 Map Tool 을 구현한다. 게임에 쓰이게 될 3D 지형을 하이트 2D 맵 편집기만을 이용해서 만들기는 힘들며 이러한 방법은 지형의 세밀함이 떨어지고 게임상에서 계속 불러들여 확인하는 것은 비효율적이다. 하이트맵을 3D 상에서 직접 불러와 편집을 하고 저장할 수 있게 된다면 훨씬 효율적이며 퀄리티를 높일 수 있을 것이다.

1.서론

게임의 발전은 기술적인 면에서 점차 상호작용을 위한 인터페이스 몰입형 및 체감형 게임으로 진행이 되고 있다. 2000 년 이후에는 3D 그래픽을 이용한 가상현실 게임이 발전하는 추세이다. 이러한 변화는 기존 게임의 한계를 극복하고 기술의 발전과 새로운 패러다임의 변화를 준비하는 것이다. 기존 2D 기술이 3D 기술로 발전하면서 일반 사용자들은 3D 의 기술을 요구하고 있는데 반해 게임의 인터페이스 기술은 단순히 2D 차원에서 머물고 있는 실정이다.

본 논문은 이러한 체감형 게임을 하기 위한 인터페이스 기술 모델로써 스노우보드 게임에서 사용되는 지형 Map Tool 을 구현한다. 게임에 쓰이게 될 3D 지형을 하이트 2D 맵 편집기만 이용해서 만들기는 힘들며 이러한 방법은 세밀함이 떨어지고 게임상에서 계속 불러들여 확인하는 것은 비효율적이다. 따라서 하이트맵을 불러 들여 3D 상에서 직접 편집을 하고 저장할 수 있게 된다면 훨씬 효율적으로 퀄리티를 높일 수 있을 것이다. 특히 스노우보드 게임은 지형에 관한 인터페이스 모델이 필요하다. 구현된 맵 툴을 이용하여 지형에 빛을 미리 삽입하여 계산된 텍스처를 입힐 수 있고, 흰색과 빛을 미리 계산(음영 넣기)함으로써 3D 상에서 높낮이를 구분하기가 쉽게 할 수 있다. 지형을 쉽고 간편하게 생성하고 편집할 수 있으며 편집할 수 있는 주위의 픽셀 개수를 줄였다 늘였다 할 수 있다.

2D 맵 디자인 방법과 3D 맵 디자인 방법, 그리고 Heightmap 구현, 마지막으로 구현된 결과와 향후 연구과제에 대해서 기술한다.

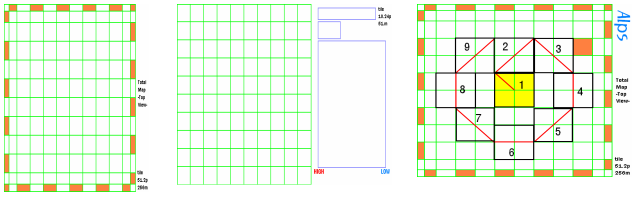
본 연구는 2006 년 지방대학 혁신역량 강화사업(NURI) 문화콘텐츠(CT) 인력양성 2 차년도 사업의 연구비 지원으로 수행되었습니다.

2.본론

본 장에서는 2D 맵 디자인에서 3D HeightMap 을 구현하기까지의 개발 방법과 프로그램을 설명한다.

2.1 2D 맵 디자인

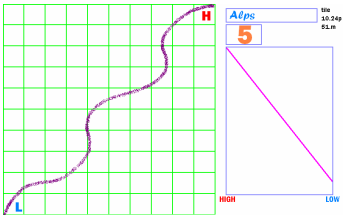
2D 맵 디자인은 Adobe Photoshop 7.0.1 과 그림판을 이용하였다.



(a) (b) (c)

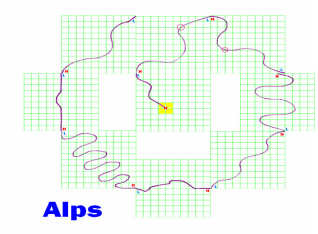
(그림 1) 2D 상에서의 맵 틀 디자인

(그림 1)에서 (a)그림은 전체 맵의 틀이고, (b)그림은 구간별 지형의 틀이다. 전체 맵의 틀은 512*512 를 표현하기 위해 만들어진 그림이다. 총 10*10 개의 타일을 디자인 하였고 하나의 정사각형의 타일은 51.2pixel 을 가리킨다. 외각에 표시된 색들은 지형의 경계면을 나타냄으로써 캐릭터가 경계면 밖으로 움직이는 것을 방지하기 위해 표시해 둔다. (c)그림은 완성된 전체 맵으로 지형 맵이 1 번부터 9 번 구간까지 나선형으로 연결된 것을 한눈에 알 수가 있다. 구간별 지형의 틀은 2*2 타일을 표현하기 위해 만들어진 그림이다. 이것은 10*10 개의 타일로 51.2/5 = 10.24pixel 을 가리킨다. 오른쪽 상단의 칸에는 나라의 이름을, 그 아래에는 구간별 지형의 이름, 마지막 아래에는 구간별로 시작점(높은 지점)과 끝점(낮은 지점)을 표현하였다.



(그림 2) 부분 맵 구간

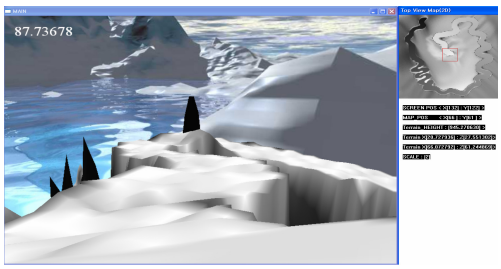
(그림 2)는 부분 맵 구간으로 5 번째 구간을 표시한다. 10*10 타일 중에서 오른쪽 상단의 'H' 점이 시작점이고 'L' 점이 끝점을 가리킨다. 오른쪽 높낮이 그림에서는 HIGH 가 'H' 를 가리키고 LOW 가 'L' 을 가리키게 된다. 구간별 맵을 모두 모아 보면 (그림 3)과 같다.



(그림 3) 전체 맵

2.2 3D 맵 디자인

3D 맵 디자인은 3DS MAX 7.0 을 사용하였다.

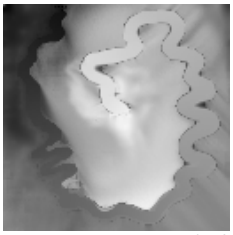


(그림 4) Terrain 3D View

하이트맵을 읽어와 지형을 테스트할 수 있다. TerrainView 에서 raw 파일을 읽어와 하이트 맵(0~255)을 생성한다. 하이트 맵에 하이트 스케일과 픽셀당 너비를 곱해서 기본 지형을 만든 후 라이트를 계산 하고 텍스처를 입히는 효과를 준다. 카메라의 위치는 지형의 위치보다 조금 높여서 지형 위를 걷는 효과를 준다. (그림 4)에서 보면 오른쪽 Top View2D 지형 window 를 생성해서 기존 Raw 파일의 모습을 보여준다. Top View2D 에서 현재 카메라 위치를 계산해 빨간색의 네모박스를 뿌려준다.

2.3 Heightmap 구현

Heightmap 은 각각의 항목이 지형 격자내의 특정 버텍스의 높이와 대응되는 배열이다. 즉, 각 요소가 지형 격자내의 각 버텍스와 일대일 대응관계를 가지는 행렬이라 할 수 있다. 파일로 보관할 때 하이트맵의 각 요소에 1 바이트의 메모리를 할당하는데 0 부터 255 까지의 높이를 가진다. 3D 상에서 이용할 때 높이 단위가 255 보다 클 수 있는데, 데이터를 읽어 들이는 단계에서 원하는 배율로 값을 조정해서 늘리면 된다. 낮은 고도를 어두운 값으로, 높은 고도를 밝은 값으로 표현하는 GrayScale 맵을 사용한다. (그림 5)는 GrayScale 로 표현된 HeightMap 을 보여준다.



(그림 5) GrayScale 로 표현된 HeightMap

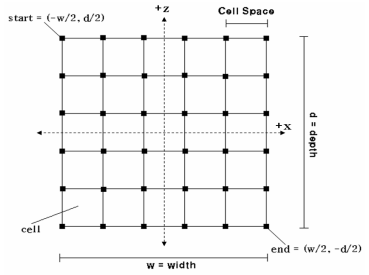
HeightMap 작성을 마친 후에 8Bit-Raw 파일로 저장한다. Raw 파일은 이미지의 각 픽셀의 음영을 헤더 없이 이진으로 순서대로 저장하며 이미지를 읽어 들이는 과정을 간단하게 만든다. (그림 6)은 HeightMap 파일을 읽어와 변수에 저장하는 프로그램이다.

```
_heightMap = (BYTE*)malloc( sizeof(BYTE) * width * height );

FILE* fp;
fp = fopen(fileName, "rb");

for(register int i=0; i<width*height; i++)
    fread(&_heightMap[i], sizeof(BYTE), 1, fp);

fclose(fp);
```

(그림 6) HeightMap 파일 변수 저장 방법

폴리곤 격자(grid)의 특성과 격자 선의 점들로 이루어진 버텍스를 (그림 7)에서 보여준다. Heightmap

은 이 방법으로 X 와 Z 좌표를 얻고, Y 좌표는 읽어 들인 Heightmap 데이터 구조체 내에서 얻을 수 있다. (그림 8)은 지형 버텍스를 생성하는 프로그램이다.



(그림 7) 버텍스 계산하기

```
int startX = -_width / 2; int startZ      = _depth / 2;
int endX   = _width / 2; int endZ         = -_depth / 2;
float uCoordIncrementSize = 1.0f / (float)_cellsPerRows;
float vCoordIncrementSize = 1.0f / (float)_cellsPerCols;
int i = 0;

for(int z = startZ; z >= endZ; z -= _cellspacing) {
    int j=0;
    for(int x = startX; x <= endX; x += _cellspacing) {
            int index = i * _rows + j;
          cv[index] = TerrainVertex(
                    (float)x,
                    (float)_heightMap[index] * _heightScale,
                    (float)z,
                    (float)j * uCoordIncrementSize,
                    (float)i * vCoordIncrementSize);
    j++;  }  i++; }
```

(그림 8) 지형 버텍스 생성 프로그램

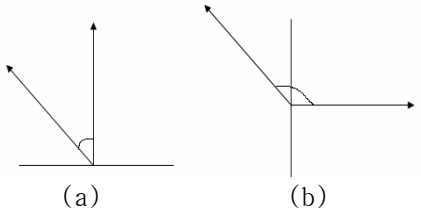
D3DXCreateTexture()라는 함수를 이용해 빈 텍스처를 만들고, 눈을 표현하기 위해 각각의 텍셀에 흰색으로 컬러를 입힌다. 텍셀을 비추는 태양 빛의 각도에 따라 텍셀의 밝기를 조정한다. 지형은 정적이며 미리 계산을 하기 때문에 실시간으로 빛 계산을 할 필요가 없어서 훨씬 빠른 속도를 낼 수 있다. 또한 버텍스 법선을 저장하지 않아도 되기 때문에 메모리가 절약된다.

```
_tex->LockRect( 0, &lockedRect, 0, 0);

DWORD* imageData = (DWORD*)lockedRect.pBits;

for(register int i=0; i<=texHeight; i++) {
    for(register int j=0; j<texWidth; j++) {
          D3DXCOLOR c = D3DCOLOR_XRGB(255, 255, 255);
          imageData[i  *  lockedRect.Pitch  /  4  +  j]  =
(D3DCOLOR)c;
    }
 }
_tex->UnlockRect(0);
```

(그림 9) 텍스처에 색을 넣는 프로그램

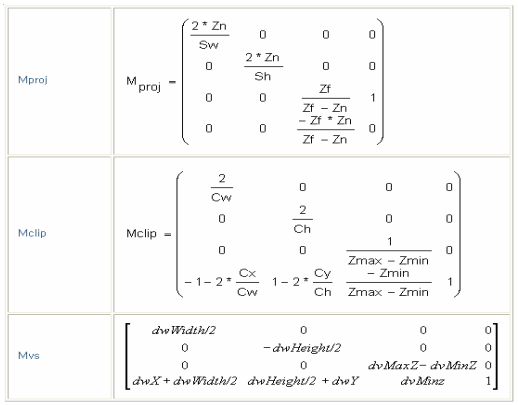
(그림 9)는 텍스처에 색을 넣는 프로그램으로 Bits 는 바이트로 주어지며 DWORD 가 4 바이트로 이루어져 있기 때문에 Bits 를 4 로 나누었다.

지형 내의 각각의 사각형에 대해 빛 벡터 L 과 사각형 법선 N 간의 각도를 계산하여 표면이 받는 빛의 양을 조절한다. 각도가 커질수록 사각형 면이 점차 광원에서 멀어져서 빛을 덜 받는다는 것을 보여주고 각도가 작아지면 사각형 면이 점차 광원으로 향하면서 더욱 많은 빛을 받는다. 빛 벡터가 법선과의 각도에서 90 도를 넘어서면 표면은 더 이상 빛을 받지 못한다. (그림 10)에서 (a)는 90 도 이하의 각도를 보여주며 (b)는 90 도를 넘는 각도를 보여준다. 빛 벡터와 표면 법선과의 각도 관계를 이용해 표면이 받는 빛의 양을 결정하는 [0, 1]범위의 음영 스칼라를 만들 수 있다. 이 음영 스칼라에서 큰 각도는 0 에 가까운 스칼라로 표현되며, 음영 스칼라와 컬러를 곱하면 0 에 가까운 값이 만들어져 어두운 컬러가 된다. 반면에 작은 각도는 1 에 가까운 스칼라로 표현되며 컬러와 곱하면 1 에 가까운 밝은 값이 만들어진다.



(그림 10) 각도에 따른 빛의 양

카메라 좌표계의 좌표가 화면좌표로 변환되는 과정은 Mproj(projection)에서 Mclip(clipping)으로 그 다음 Mvs(Viewport scaling)에서 projection divide 의 과정으로 변환된다. (그림 11 은) 변환 Matrix 식을 보여준다.



(그림 11) Matrix 변환

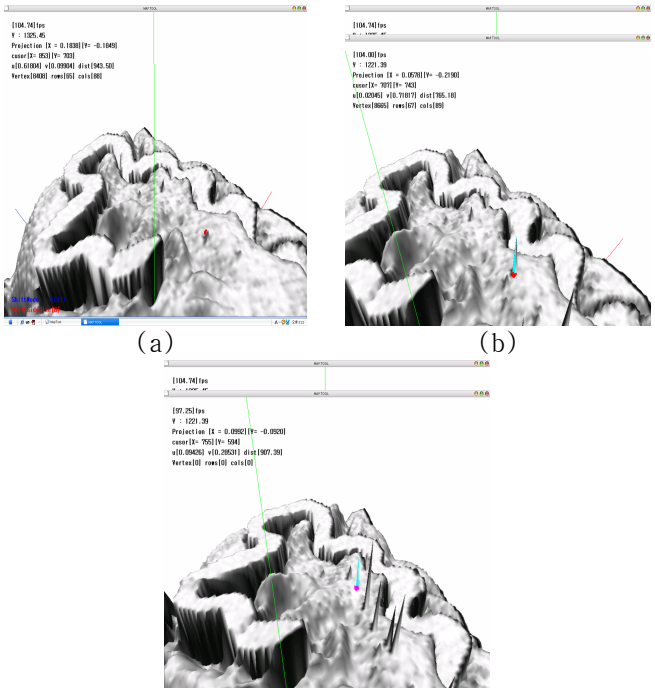
projection 행렬(MDXproj)은 Mproj * Mclip 을 나타내며 위의 행렬을 모두 곱하면 ViewMatrix 를 구할 수 있다. Inverse 를 취하면 월드 공간에서의 좌표가 나온다. 이 방법을 이용해서 마우스 좌표에 해당하는 3D 점을 구할 수 있고 이 후 카메라 좌표와 계산된 점을 잇는 반직선을 만든다. (그림 12)에서 계산한 반직선이 삼각형에 교차하는지를 검사해서 pick 을 수행한다. 이 중에서 카메라에 가장 가까운 삼각형을 골라낸다.

```
  GetCursorPos( &ptCursor );
  ScreenToClient( g_hwnd, &ptCursor );
  D3DXMATRIXA16 matProj;

  pDevice->GetTransform( D3DTS_PROJECTION, &matProj );

  pjPos.x =  ( ( ( 2.0f * (ptCursor.x-clientRect.x )) /
clientRect.width  ) - 1 ) / matProj._11;
  pjPos.y = -( ( ( 2.0f * (ptCursor.y-clientRect.y )) /
clientRect.height ) - 1 ) / matProj._22;
  pjPos.z =  1.0f;

    // Get the inverse view matrix
    D3DXVECTOR3 vPickRayDir;
    D3DXVECTOR3 vPickRayOrig;
    D3DXMATRIXA16 matView, m;
    pDevice->GetTransform( D3DTS_VIEW, &matView );
    D3DXMatrixInverse( &m, NULL, &matView );

    vPickRayDir.x    =  pjPos.x*m._11  +  pjPos.y*m._21  +
pjPos.z*m._31;
    vPickRayDir.y    =  pjPos.x*m._12  +  pjPos.y*m._22  +
pjPos.z*m._32;
    vPickRayDir.z    =  pjPos.x*m._13  +  pjPos.y*m._23  +
pjPos.z*m._33;

    vPickRayOrig.x = m._41;
    vPickRayOrig.y = m._42;
    vPickRayOrig.z = m._43;

    D3DXVec3Normalize(&vPickRayDir, &vPickRayDir);

    terrain->InterSectTriangle(&vPickRayOrig, &vPickRayDir);
```

(그림 12) Picking 프로그램

D3DXIntersectTri()함수에서 마지막 인자를 얻어오면 폴리곤과 반직선의 교점이 카메라에서 얼마나 떨어져 있는지 알 수 있기 때문에 가장 가까운 삼각형을 추출할 수 있다. 3D 마우스 좌표는 광선의 위치, 광선의 방향, 반직선의 교점으로 구할 수 있다.

(그림 13)은 구현된 Height Map Tool 을 이용하여 지형을 구성해 본 화면이다. 3D 상의 지형의 버텍스와 마우스 점이 맞닿은 곳을 잡아 당겼을 때 높은 지형이 생성된 것을 확인할 수 있다.



(a) (b) (c)

(그림 13)구현된 Height Map

3. 결론 및 향후 연구 과제

본 구현을 통하여 스노우보드 게임에서 사용되는 지형을 간단하고 쉽게 조작 할 수 있었고, 3D 상에서 지형의 높낮이를 구분하기 수월하였다. 또한 카메라를 1 인칭 모드로 변환하여 실행했을 때 실제 지형이 게임에 표현되는 방법을 확인 할 수 있었다. 하지만 몰입과 체감이 적용된 스노우보드 게임에 사용되기에는 아직 부족하다. 향후 가상현실 장비를 이용한 체감형 스노우보드 게임에서 사용될 수 있도록 지형에 대한 물리적 기술요소가 포함되도록 연구가 진행되어야 한다.

4. 참고문헌

[1] DirectX9 를 이용한 전략 게임 프로그래밍, 정보문화사, Todd Barron 저 , 최현호 역
[2] IT EXPORT 3D 게임 프로그래밍, 한빛미디어, 김용준 저
[3] 3ds MAX 7.x 리얼리티, 디지털북스, 김초석저
[4] www.winapi.co.kr
[5] www.gamza.net
[6] msdn.microsoft.com/library