한국게임학회 논문지
제7권 제4호 2007년 12월

Walk-through를 지원하는 GPU 기반 지형렌더링

박선용[0],오경수*, 조성현**
{숭실대학교 미디어학과}[0,*], 홍익대학교 게임학부**
{aknyong[0], oks*}@ssu.ac.kr, scho@hongik.ac.kr**

GPU-only Terrain Rendering for Walk-through

Sun-Yong Park[0], Kyoung-Su Oh*, Sung Hyun Cho**
{Dept. of Media, Soong-Sil University}[0,*], School of Game, Hong-Ik University**

요 약

이 논문에서는 다양한 종류의 응용프로그램에 바로 적용할 수 있는 GPU 레이캐스팅 기법에 기반 하는 효율적인 실시간 지형 렌더링 방법을 제안한다. 여기에서 제안되는 방법은 별도의 메시 구조 없이 이미지(높이맵) 만으로 지형을 표현하는 것이 가능하며, 공중과 지상에서의 활동이 자유로워 가상현실은 물론 게임에 바로 사용할 수 있다. 메시에 기반 하지 않으므로 별도의 LOD 조절이 필요하지 않으며, 높이맵과 컬러맵의 해상도에 따라 기하표현의 정밀도와 화질이 결정된다. 더욱이 GPU-only 기법은 CPU가 더 일반적인 작업에 집중할 수 있도록 함으로써 시스템의 전반적인 성능을 향상시킨다. 지금까지 높이맵을 사용한 많은 지형렌더링 관련 연구는 대부분이 상당부분 CPU 의존하거나 그 응용범위를 비행 시뮬레이션 등에 제한하여 왔다. 우리는 기존의 변위매핑 기법을 개선하여 지형렌더링에 적용함으로써 기존 폴리곤기반 기법에서 나타나던 크랙이나 팝핑 등의 문제점들을 근본적으로 배제하였다. 이 논문이 기여하는 바는 지표면 탐색(walk-through)시의 임의 시선에 대한 효율적 처리와 높이장(height field)의 곡면 재구성을 통한 화질의 획기적인 개선이다. 우리는 재구성된 곡면과 시선의 교차점을 계산하는 효율적인 방법을 제시한다. 우리가 제안하고 있는 알고리듬은 100% GPU에서 구현되었으며, 256×226 ~ 4096×4096까지의 다양한 해상도에서 실험한 결과 초당 수십 ~ 수백 프레임을 얻었다.

ABSTRACT

In this paper, we introduce an efficient GPU-based real-time rendering technique applicable to every kind of game. Our method, without an extra geometry, can represent terrain just with a height map. It makes it possible to freely go around in the air or on the surface, so we can directly apply it to any computer games as well as a virtual reality. Since our method is not based on any geometrical structure, it doesn't need special LOD policy and the precision of geometrical representation and visual quality absolutely depend on the resolution of height map and color map. Moreover, GPU-only technique allows the general CPU to be dedicated to more general work, and as a result, enhances the overall performance of the computer. To date, there have been many researches related to the terrain representation, but most of them rely on CPU or confined its applications to flight simulation. Improving existing displacement mapping techniques and applying it to our terrain rendering, we completely ruled out the problems, such as cracking, poping etc. which cause in polygon-based techniques. The most important contributions are to efficiently deal with arbitrary LOS(Line Of Sight) and dramatically improve visual quality during walk-through by reconstructing a height field with curved patches. We suggest a simple and useful method for calculating ray-patch intersections. We implemented all these on GPU 100%, and got tens to hundreds of framerates with height maps a variety of resolutions(256x256 to 4096x4096).

Keyword : Real-time rendering, Ray-casting, Walk-through, Curved patching

이 논문은 2007년도 정부재원(문화관광부)으로 한국게임산업진흥원의 지원을 받아 연구되었음.
본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.

1. 서 론

컴퓨터 그래픽스 분야에서 지형의 중요성은 여러 곳에서 확인된다. 군사, 과학적인 측면에서의 GIS로부터 Virtual Reality, Computer Game에 이르기까지 그 응용분야는 매우 다양하다. 그러나 이러한 다양성에 비추어 그를 충족할 만한 방법론은 많지 않다.

지금까지 컴퓨터 그래픽스 분야에서의 지형표현에 관한 연구를 살펴보면, 첫째가 지형의 기하구조 자체를 모델링하는 방법이다. 폴리곤 메시를 이용한 지형표현은 높이맵 외에 추가적으로 메시 구조를 필요로 하며, LOD 조절은 크랙, 팝핑 등의 문제와 함께 CPU에 많은 부담을 가져온다. 반면 여기에서 제안하는 방법은 이미지를 기반으로 하여 별도의 기하구조를 가지지 않으며, 높이장(height field) 탐색시 컬러를 동시에 처리하여 실행속도의 향상을 가져온다. 둘째로 GPU 의존도(dependency)를 높이기 위한 연구이다. 여기서 제안하는 기법은 완전히 GPU에서 구현되었다. 지형표현을 100% GPU에서 처리하여 CPU를 게임을 위한 AI, 물리기반 연산, 자원관리 등 더 일반적이고 상황 의존적인 작업에 전념할 수 있도록 함으로써 컴퓨터의 전반적인 수행능력을 제고하였다. 마지막으로 많은 연구들이 공중에서 바라본 지형의 표현과 대량의 데이터의 처리에 그 중점을 두고 있다. 하지만 카메라 또는 눈의 위치를 공중으로 제한하여 지표면에서의 자유로운 활동이 힘들다는 단점이 있다.

이 논문에서 사용된 기법은 변위매핑에 기본을 둔다. 변위 매핑은 높이장 정보를 이용하여 법선매핑의 조명조작 효과는 물론 대량의 폴리곤으로 모델링 된 지형 이상의 3D 효과를 얻을 수 있다. 최근 기법 중 parallax occlusion mapping[1]이나 relief mapping[2] 등은 복잡한 표면에 대한 실시간 렌더링이 가능하지만 경우에 따라 심각한 시각적 화질상 결함을 보인다. 이를 해결한 것이 Oh et al[3]의 Pyramidal Displacement Mapping이다. 이 방법은 변위 맵을 최대값에 의해 쿼드트리로 만들고 개선된 탐색 알고리듬을 통해 [1, 2]에서 발생하는 화질상 문제를 해결 하였다. 기존의 변위 매핑은 하향시선을 기본으로 하고 있기 때문에 공중에서 바라보는 지형(fly-through)에는 바로 적용되지만 높이장 내부로 들어갈 경우(walk-through) 상향시선에 대한 처리가 필요하다. 특히, 높이장 탐색시 텍셀이 픽셀보다 커지는 경우 각 요소들이 사각기둥 형태로 보이는 문제가 있다. 우리는 인접한 4개의 높이장 요소로 쌍선형 패치를 구성하여 높이장 전체를 곡면화 함으로써 상당부분 화질을 개선하였다.

우리 논문이 지형렌더링 관련분야에 기여하는 바는 다음 세 가지로 요약된다.

1. 계층적 변위 맵을 개선하여 지형렌더링에 적용함으로써 높이장을 효율적으로 탐색하였으며,
2. Fly-through와 walk-through를 동시에 구현하여 시점과 시선의 자유도를 확보하였고,
3. 높이장을 곡면화 함으로써 지표면 근접시의 화질을 개선하였다.

이 논문의 2절에서는 관련연구를 소개하고, 3절은 곡면패치 없이 높이맵으로만 렌더링 된 지형에 대해 논의 하며, 4절에서는 3장에서 논의된 결과를 바탕으로 높이장의 곡면처리와 그 과정에서 나타난 문제점 및 해결책에 대해 기술한다. 5절에서는 렌더링 된 최종결과를, 마지막으로 6절에서는 결론과 추후 연구방향에 대해 언급한다.

2. 관련연구

폴리곤 기반 지형표현에서의 주 관심은 LOD 적용에 따른 기하구조 재편(re-construction)과 그에 따른 텍스처 리매핑(re-mapping)이다. 작업의 상당 부분이 CPU 상에서 이루어지므로 많은 부담으로 작용한다.

기하-텍스처 기반기법의 문제점을 개선하기 위해 M. H .Cross et al.[4]은 적응적 쿼드트리 메시(adaptive quad-tree Mesh)를, Pajarola[5]는 제한 쿼드트리 삼각형 분할(restricted quad-tree triangulation) 기법을 제안 하였으며, Duchaineau et al.[6]은 메시 재구성을 위한 ROAM(Real-time Optimally-Adapting Meshes) 기법을 고안하였다.

Pomeranz[7]는 공유 모서리를 제한하는 GPU 기반의 알고리듬인 RUSTiC을 제시하였으며, L. M. Hwa et al.[8]은 4-8메시를 활용, 텍스처와 높이장(height field) 양쪽에 다이아몬드 계층구조를 구축하였다. 이 알고리듬은 GPU 메모리를 캐시로 사용하여 효율적인 out-of-core 렌더링을 가능하게 한다.

지형표현은 시점의 위치와 활용목적에 따라 관점을 달리 할 수 있다. Daniel Cohen-Or et al.[9]은 레이캐스팅 기법을 이용한 CPU 기반의 실시간 fly-through 알고리듬을 소개하였다. 효율성을 위해 높이맵으로부터 생성된 복셀 맵을 이용하였으며, 비행 시뮬레이션과 구분하여 visual fly-through라는 용어를 정의하였다. 광활한 지역을 fly-through 하기 위한 방법으로 clip-mapping이 있다. Frank Losasso et al.[10]은 texture clipmap[11]을 응용한 geometry clipmap을 제안하였다. Geometry clipmap은 지형데이터를 내포된 정사각형 격자 형태로 캐싱하며 캐싱된 격자들은 버텍스 버퍼로 저장되어지고 시점의 이동에 따라 다시 채워진다. 이 알고리듬 역시 공중에서 바라본 지형을 렌더링 할 뿐 시점이 내부로 들어가지는 않는다. Asirvatham과 Hoppe[12]는 이를 개선하여 거의 모든 렌더링 과정을 GPU에서 해결하였다.

이 논문에서 우리는 높이맵으로부터 재구성된 계층 구조맵을 사용한다. Cohen et al.[13]은 CPU 기반 계층구조를 사용한 가속화된 높이장 렌더링 알고리듬을 제시하였다. 이 방법은 다른 CPU 기반의 알고리듬[13, 14, 15]에 비해 빠르지만, 역시 비실시간이며, 부드러운 화질 표현에 문제가 있다.

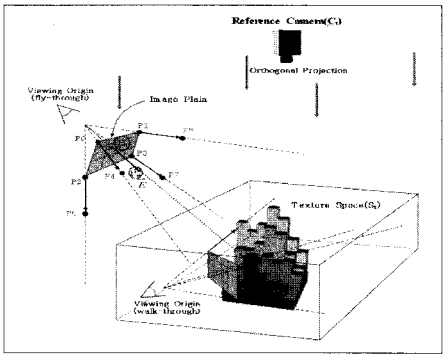
몇몇 GPU기반 렌더링기법이 상호작용 가능한 속도로 높이장을 표현한다. Parallax occlusion mapping[1]은 선형보간 된 시차변위를 이용하며, Relief mapping[2]은 이진탐색을 사용한다. 하지만 각각 변화가 많은 지형과 지표각에서 문제를 가지고 있다. Dynamic parallax occlusion mapping[16]은 시선의 방향과 높이장의 변화에 따라 샘플링을 달리 함으로써 이러한 문제를 완화시켰지만 완전한 해결책으로 제시되지는 못하였다.

Oh et al[3]의 Pyramidal Displacement Mapping은 밉맵을 활용, 변위 맵을 쿼드트리 구조로 재구성하고 탐색방법을 개선하여 이 문제를 완전히 해결 하였다.

3. 변위맵을 이용한 GPU-only 지형 렌더

우리가 제안하는 기법은 계층적 변위매핑 기법(PDM)에 기반 하기 때문에 PDM의 장점을 그대로 유지면서, 이미지를 기반으로 하여 별도의 기하구조나 LOD 정책을 필요로 하지 않는다.

PDM은 밉맵의 활용범위를 본래목적 이 외의 영역으로 확장하였으며, 다른 GPU기반 기법들[1 ,2]과 같은 픽셀 단위 조명기법을 사용하여 좋은 화질을 보장한다. PDM은 밉맵과 쿼드트리에 의한 계층형 자료구조를 이용하며, 모든 연산이 GPU 상에서 이루어지므로 매우 빠르다. 우리는 PDM의 시점 및 시선의 방향에 대한 개선을 통해 임의시점에서의 지형탐색이 가능하도록 하였다.



[그림 1] 광선 설정

3.2 광선의 설정

픽셀단위 ray casting을 위해서는 픽셀쉐이더를 이용해 픽셀 당 하나의 광선을 할당해야 한다.

우선 높이장 가상공간 St를 정의하기 위해 참조카메라 Cr을 시점으로 하는 변환행렬 VPT_{camera}(View-Projection-Texture transform matrix)를 설정한다.

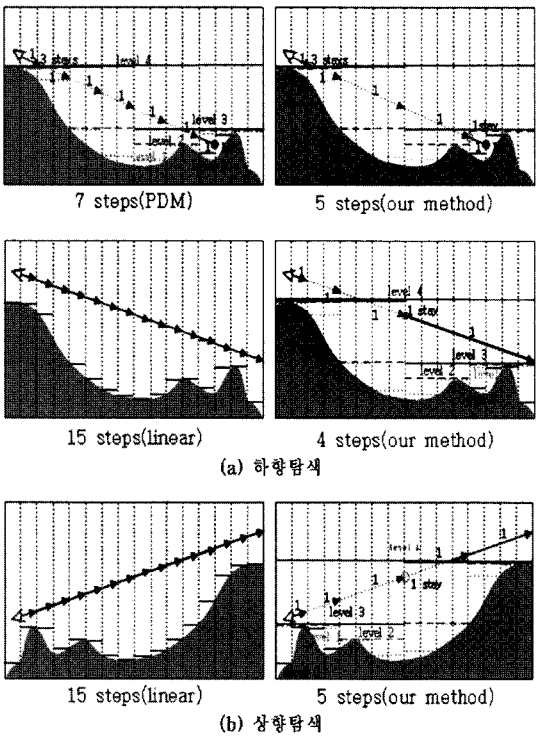
뷰공간에서 시점과 시선방향은 1) world 공간으로의 역변환 2) 행렬 VPT_{camera}를 통해 높이장 공간(S_t)으로 변환될 수 있다.

우리가 제안하는 방법은 기하구조를 가지지 않으므로 쉐이더를 이용하기 위해서 이미지 평면에 상응하는 가상의 참조평면을 설정한다[그림 1]. 정점쉐이더에서는 1) 뷰 원점과 평면의 각 꼭지점을 연결한 $OP_0 \sim OP_3 \times 2$ 만큼의 위치에 점 $P_4 \sim P_7$를 설정하고, 2) 8개의 점 $P_0 \sim P_7$을 월드공간으로 역변환 한 뒤, 3) 행렬 VPT_{camera}를 적용해 공간 St로 변환한다. 픽셀 쉐이더에서는 1) 보간된 좌표들을 w로 나눠 원근보정을 한 뒤 2) 위치간 뺄셈(P_e-P_s)과 3) 정규화를 통해 시선의 방향의 단위 벡터(E)를 얻는다.

3.3 높이장 탐색

Relief mapping[1, 2, 16]에서 사용되어진 대부분의 알고리듬은 아래로 향하는 광선만 고려하였다. 위에서 내려다 보기 때문이다. 하지만 높이장 내부로 들어가는 walk-through를 지원하는 시스템에서는 시선이 위쪽을 향하는 경우가 많기 때문에 오히려 더 중요하다.

우리는 기존의 탐색 알고리듬을 개선하여 임의 시선을 효율적으로 처리하였으며, 쿼드트리구조를 활용하여 효율성을 제고하였다.



[그림 2] 최하위 레벨에서의 상향탐색(a)과 하향탐색(b)

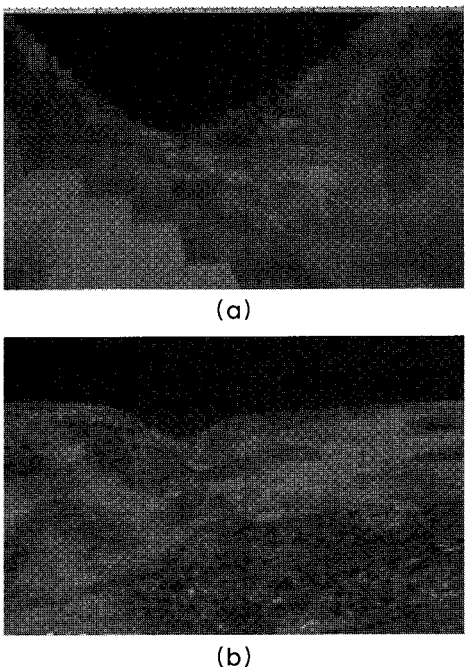
3.3.1 하향탐색

하향하는 방향에서의 탐색은 PDM의 경우와 기본적으로 같다. 다만 높이장과 만나지 않을 경우 광선을 강제로 전진시키면서 트리의 레벨을 높여 탐색횟수를 선형탐색의 약 1/3~1/2로 줄였다[그림 2](a). 광선이 지형 가까이 지나가는 경우에도, 기존의 방법(PDM) 보다 근소하지만 개선효과를 보였다([그림 2](a) 위).

3.3.2 상향탐색

위에서 언급한 바와 같이 walk-through를 지원하는 지형렌더링에서는, 상향탐색이 더 많은 경우를 차지한다. 가장 간단한 방법은 선형탐색으로 구현이 쉬운 반면 맵의 해상도에 비례해 비용이 증가한다.

시점이 공중에 있을 때보다 지상에 있을 때 탐색횟수 증가를 보이기 때문에 이를 최소화 할 수 있는 방법이 요구된다. GPU에서는 CPU 기반 알고리듬의 장점인 공간적 또는 광선간의 연관성(coherence)을 이용할 수 없다는데 한계가 있다. 우리는 쿼드트리의 어느 레벨에서라도 높이장 요소와 만나지 않으면 레벨을 상위로 조정하여 한 번에 전진하는 거리를 증가시켜 탐색횟수를 최소화 하였다. 여러 경우에 대해 효율성을 평가하였으며, 레벨을 일률적으로 한 단계 증가시키는 경우 가장 좋은 결과를 나타냈다. [그림 2]는 광선이 허공을 지나갈 때 선형탐색, 기존 PDM, 그리고 이 논문에서 제안하는 알고리듬을 비교하였다. 선형탐색에 비해 약 1/3로 줄어듦을 알 수 있다.



[그림 3] 변위매핑을 이용한 지형렌더링 : 근거리시점(a)과 원거리시점(b)

3.4 렌더링 결과

[그림 3]은 곡면을 사용하지 않았을 때의 렌더링 결과를 보여준다. [그림 3](b)에서와 같이 원거리에서 큰 문제점을 발견할 수 없으나, 지표 근처로 이동하면서 텍셀의 크기가 픽셀의 크기보다 커져 지형표면이 사각기둥의 집합으로 보

인다. [그림 3](a). 우리는 쌍선형 패치를 이용해 이 문제를 개선하였다(4절). 여러 가지 곡면 모델 중 연산의 효율성을 고려하여 쌍선형 패치를 적용하였으며, 몇몇 경우에 있어서는 패치 경계에서 미세한 시각적 문제가 발생하였다.

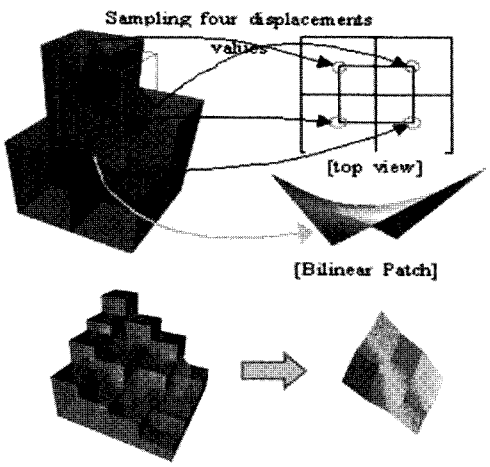
여기서 제안하는 알고리듬의 효율성은 폴리곤 기반 기법과 달리 LOD 조작이 아닌 탐색횟수의 최소화에 있다. 알고리듬의 시간 복잡도는 평균 $O(\log_2 n)$, 최악의 경우 선형탐색과 $O(n)$을 나타낼 수 있다.

4. 쌍선형 패치를 이용한 렌더링

3.4절에서 우리는 높이맵을 그대로 지형으로 렌더링 할 경우의 심각한 화질상의 문제를 확인하였다. 이번 절에서는 그러한 문제점을 개선하기 위해 곡면패치를 활용하는 방안에 대해 논의한다. 베지어, 스플라인, B-스플라인 등 다른 곡면모델의 적용이 가능하지만 실행 속도를 고려, 쌍선형 패치를 사용하였다.

4.1 광선-패치 교차검사

쌍선형 패치가 구성되려면 먼저 광선이 높이장 요소와 만나야 한다. 일단 충돌이 일어나면 부딪힌 요소와 인접한 네 개 요소의 중심위치를 연결해 패치를 구성하는데 , 네 지점에서 깊이 값을 읽어 3차원 좌표를 결정하고 그 좌표로 쌍선형 패치를 만든다. [그림 4]는 이러한 과정과 이렇게 만들어진 쌍선형 패치를 보여준다.



[그림 4] 쌍선형 패치의 구성

다음으로 레이-패치 교점은 다음의 방정식을 풀어 비교적 간단하게 구해질 수 있다.

$$
\begin{aligned}
&(1-u)\{P_{00}\cdot(1-v)+P_{01}\cdot v\}+u\{P_{10}\cdot(1-v)+P_{11}\cdot v\}\\
&=(1-u)(1-v)\cdot P_{00}+v(1-u)\cdot P_{01}+u(1-v)\cdot P_{10}+uv\cdot P_{11}\\
&=(u,v,(1-u)(1-v)z_{00}+v(1-u)z_{01}+u(1-v)z_{10}+uvz_{00})\\
&=(u,v,w)=P_{Inter\ sec\ tion}\quad\cdots\cdots(1)
\end{aligned}
$$

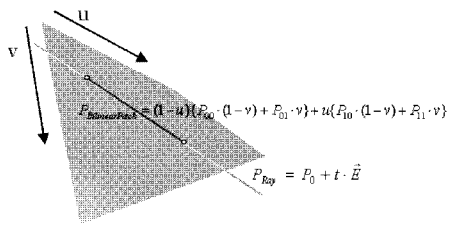
$$
\begin{aligned}
&t=(w-P_{0z})/E_z\\
&=\{uv(z_{11}-z_{10}-z_{01}+z_{00})+u(z_{10}-z_{00})+v(z_{01}-z_{00})+z_{00}-P_{0z}\}/E_z\\
&=(uv\cdot A+u\cdot B+v\cdot C+z_{00}-P_{0z})/E_z\quad\cdots\cdots(2)
\end{aligned}
$$

$$
\begin{aligned}
&P_{Inter\ sec\ tion}=(u,v,w)=P_0+t\cdot\vec{E}\\
&\begin{cases}u=t\cdot E_x+P_{0x}\\ v=t\cdot E_y+P_{0y}\end{cases}\quad\cdots\cdots(3)
\end{aligned}
$$

$$
\begin{aligned}
&(E_x\cdot E_y\cdot A)t^2+(E_x\cdot P_y+P_x\cdot E_y)\cdot A+E_x\cdot B+E_y\cdot C-E_x\cdot E_z)t\\
&+(P_x\cdot P_y\cdot A+P_x\cdot B+P_y\cdot C+D-P_{0z})\\
&=at^2+bt+c=0\quad\cdots\cdots(4)
\end{aligned}
$$

$$
\Leftrightarrow t=\frac{-b\pm\sqrt{b^2-4ac}}{2a}\quad\cdots\cdots(5)
$$

$$
D=b^2-4ac\quad\cdots\cdots(6)
$$

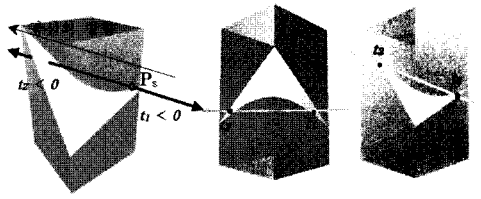


[그림 5] 광선-쌍선형 패치 교차검사

그러나 여기서 구해진 근(5)가 모두 유효하지는 않다. 식(4)의 근은 쌍선형 패치가 아닌 쌍선형 곡면과의 교차를 의미한다. 패치 내부에서 교차가 일어난 경우만을 유효한 근으로 선택해야 한다. 식 (4)의 계수를 통해 일차적으로 해의 종류와 유효한 해를 구분한다. [그림 6]의 파란색(밑줄) 부분이 실수해를 나타내며, 패치내부의 통과여부는 추가적으로 경계검사를 통해 확인되어야 한다. 이를 확인하는데 과정이 [그림 8]에 설명되어 있다.

```
If(a==0){
  if(b!=0) one intersection with a coplain; ..... (1)
  else if(b==0){
       if(c!=0) no solution(pass over the patch);
       else trivial solution; .....(2)
  }
else one double or two solutions; .....(3)
```

[그림 6] 계수를 통한 해의 구분



[그림 7] 광선-패치 교차의 일반적 형태

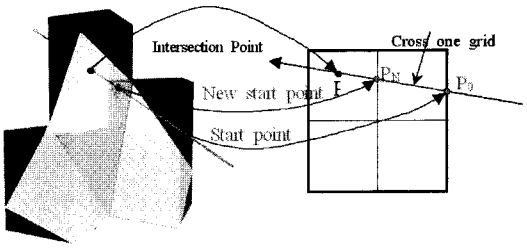
```
If(t1<0){
  t2 is a candidate for the right intersection ·····(1)
Else
  t2 is the right intersection ·····(2)

if(t2<0){
  if(t2 is out of the patch) ·····(3)
      Make the ray move forward one more grid
       until it meets with another height field element;
      Find a intersection at the position;
  else
      Select t2 as the right intersection ·····(4)
}
```

[그림 8] 패치 경계검사

[그림 7](a)는 바운딩 볼륨 내부에 포함되어 있는 패치와 광선의 시작점(P_s)을 나타낸다. 바운딩 볼륨 생성과정은 4.3절에서 자세히 설명되어 질 것이다. 전진하는 광선은 일반적으로 [그림 7](b)의 두 형태로 패치와 만나는데, [그림 5] 식(4)의 근 t_1, t_2는($t_2 \geq t_1$) 시작점 P_s로부터 얼마만큼 전진해야 곡면과 만나는 지를 나타내므로 광선이 바운딩 볼륨 내 패치와 만나기 위해서는 일단 t_1 또는 $t_2 \geq 0$가 되어야 한다. 음수([그림 7](a))라면 바운딩 볼륨 뒤쪽에서 곡면과 만나므로 근에서 제외시키고, 패치의 경계검사를 통해 t_2가 패치와 만나는지 확인한다.

만약 t_1이 양수라면 [그림 7](b)의 좌측 경우로 정확히 교점을 나타낸다. 반대로 t_1이 음수라면 [그림 7](b)의 우측 경우가 되어 t_2를 후보 값으로 하고 경계검사를 통해 패치와 만나는 지 확인해야 한다. 만약 두 근 모두가 패치와 교차하지 않는다면 광선을 더 전진시켜 다음에 구성되는 패치와의 광선-패치 교차 검사를 통해 픽셀 값을 결정한다[그림 9].

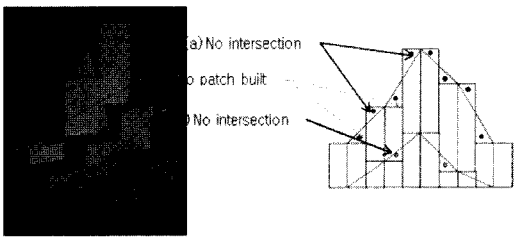


[그림 9] 패치 경계검사

4.2 광선-패치 교차검사의 문제점

모든 문제가 해결된 것으로 보이지만 픽셀 쉐이더를 이용할 때 주의해야 할 점이 있다. 이것은 프래그먼트 프로그래밍의 전통적인 문제점으로 픽셀쉐이더는 각 픽셀의 값만을 결정할 수 있다. 별도의 자료구조를 사용하지 않는다면 주변패치와 관련된 어떠한 정보도 이용할 수 없다. 그러므로 4.1절의 방법에 따라 GPU 상에서 광선-패치 교점을 구할 경우 크게 다음 두가지 경우에 의해 문제가 발생한다.

1) 높이장과 만나고 패치와 만나지 않는 경우([그림 10](a),(b))
2) 패치와 만나고 높이장과 만나지 않는 경우([그림 10](c))



[그림 10] 광선-패치 교점을 구할 수 없는 경우

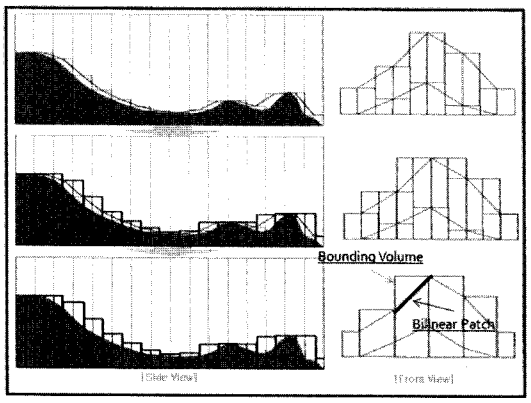
[그림 10]의 (a),(b)는 case 1), (c)는 case 2)를 나타낸다. 광선(a), (b)는 각각 높이장 요소와 만나므로 쌍선형 패치를 구성하고 [그림 5]의 식(4)를 풀어 해를 구할 수 있지만, 모든 교차가 패치 외부에서 일어나기 때문에 광선을 전진시키거나([그림 10](a)) 후진시켜([그림 10](b)) 다른 인접패치와의 교차여부를 확인해야 한다.

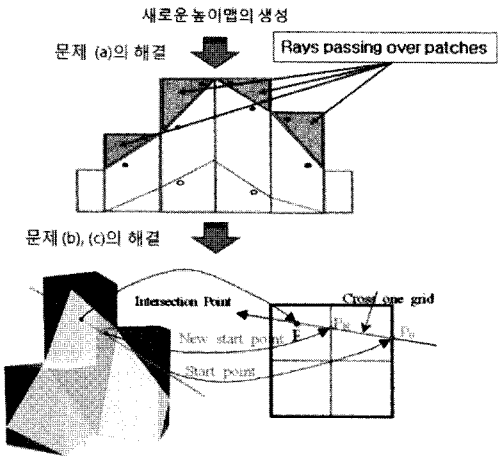
[그림 9](c)는 높이장 요소와 교차가 일어나지 않으므로 처음부터 패치가 구성되지 않으며 결과적으로 교차점을 구할 수 없다. 이 경우 실루엣에 재기(jaggy) 현상이 나타나거나 지형 중간에 hole이 생기게 된다.

4.3 4.2에서 제시된 문제의 해결

우리는 위에 제시된 문제를 해결하기 위해 인접한 4개 높이의 최대값을 하나의 높이장 요소로 하는 새로운 높이맵(lid map)을 정의하고[그림 11], 이 맵을 leaf로 하는 새로운 쿼드트리를 만들어 기존의 트리를 대체한다, 이 후 광선은 이전과 동일한 방법으로 새로운 계층 맵을 탐색한다. 차이는 광선이 높이장과 만난 이 후에 있다. 광선이 만난 위치를 원본 맵으로 옮겨 패치와의 교차점을 구한다(4.1절 참조).

새로 구성된 맵은 원본 맵의 덮개(lid) 역할을 하면서 각 패치의 바운딩 볼륨을 구성한다. 결과적으로 4.2절에서 제시된 문제점들 중 [그림 10](b), (c)가 해결되며, (a)는 4.1절에서 언급된 바와 같이 광선을 강제로 전진시켜 광선 방향 전방에 있는 다른 패치와의 교차여부를 확인한다. 실험결과 수차례 전진 후 교점을 찾을 수 있었다[그림 9].





[그림 11] 덮개맵(Lid map)의 구성/ 문제해결과정

5. 실험 및 결과

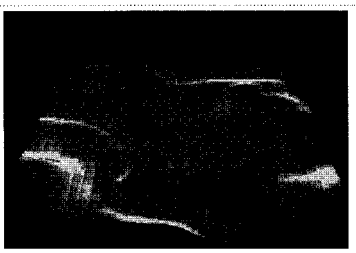
이 논문에서 제안하는 방법은 DirectX 9.0를 이용하였으며, 시스템은 ATI Radeon 2900 그래픽 카드와 2.33GHz Intel CPU를 사용하였다. 실험은 256×256부터 4096×4096까지 5가지 해상도의 높이 맵을 사용하였다. [표 1]은 해상도별 쿼드트리 구성을 위한 전 처리에 소요되는 시간과 실제 렌더링 시간(framerates)를 나타낸다.

[그림 12]는 시점별 탐색횟수(광선의 전진횟수)를 색깔별로 나타낸 것으로 시점이 지형의 최대고도보다 높은 경우([그림12] (a)) 16회 이내(blue, sky blue)에서 90% 이상의 탐색이 완료됨을 알 수 있다. [그림12](b), (c)는 지상에서 바라본 장면으로 일부 5% 이하의 일부지역에서 80회 이상(붉은색)의 탐색횟수를 나타내며, 적지만 실행속도에 상당한 영향을 주기 때문에 탐색방법의 추가적인 개선이 요구된다.

[그림 13]은 임의지형의 높이 맵을 사용하여 렌더링 한 결과로 세부지형과 실루엣이 실제와 같이 표현되고 있음을 알 수 있다.

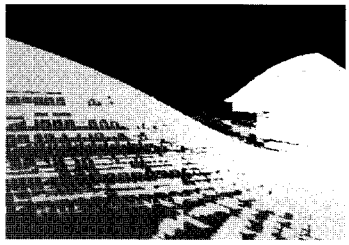
해상도	전처리 (sec)	실행속도(fps)	
		fly-through	walk-through
256×256	0.7	105~115	55~98
512×512	2.0	69~99	54~85
1024×1024	4.5	53~75	40~82
2048×2048	11	47~56	37~80
4096×4096	45	40~44	32~67

[표 1] 해상도별 전처리/ 실행속도

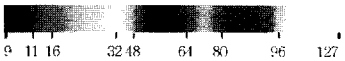


(a) fly-through

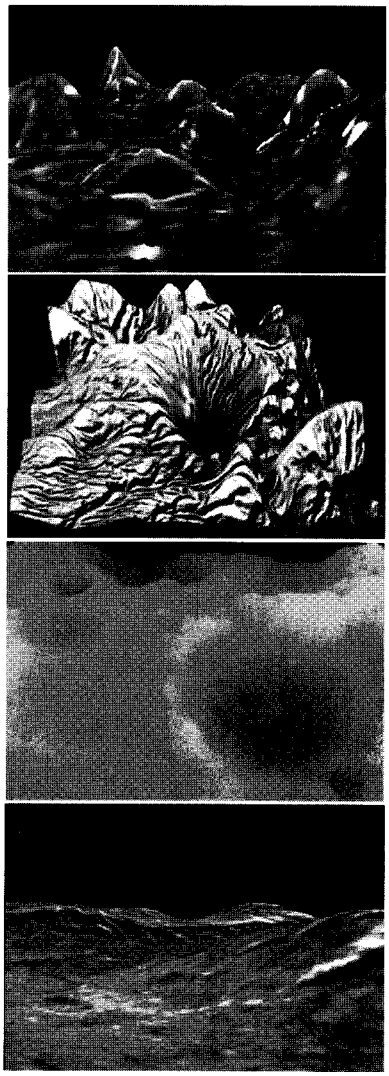
(b) walk-through(1)



(c) walk-through(2)



[그림 12] 시점에 따른 탐색횟수



[그림 13] 렌더링 결과

6. 결론

우리는 이 논문에서 GPU만을 이용한 지형렌더링 방법을 소개하였다. 이 논문에서 제안하는 방법은 특정한 기하구조에 대한 기반 없이 높이맵 만으로 사실적인 지형의 표현을 가능하게 한다. 시점이 지상에 머물 경우 쿼드트리의 효율적인 상향 탐색을 통해 시선의 자유도를 확보하였으며, 각 높이장 요소를 곡면패치로 처리하여 지면 부근에서의 화질을 개선하였다. 높이맵에 대한 우리 접근방법은 항공기에서 지상을 바라본 광경(fly-through)은 물론, 지상에서 땅을 딛고(walk-through) 걸어 다닐 수 있는 효율적인 방법을 제공함으로써 비행 시뮬레이션을 포함 지상게임/ 가상현실 등 폭넓은 분야에 활용이 가능하다. 다만 광범한 지형 데이터의 끊임없는 표현을 위해 tiling 또는 geometry clipmap 등의 방법을 사용하여 표현 범위를 확장시킬 수 있는 추가적인 연구가 요구된다.

참고 문헌

[1] Z. Brawley and N. Tatarchuk, "Parallax Occlusion Mapping: Self-Shadowing, ,Perspective-Correct Bump Mappping Using Reverse Height Map Tracing", In ShaderX3: Advanced Rendering with DirecX and OpenGL, pp135-154, 2004.

[2] F. Policarpo, M. M. Oliveira and J. Comba, "Real-Time Relief Mapping on Arbitrary Polygonal Surfaces", In ACM SIGGraph Symposium on Interactive 3D Graphics and Games Proc. pp359-368, 2005.

[3] Oh, K. Ki H., Lee C., "Pyramidal Displacement Mapping: A GPU based Artifacts-Free Ray Tracing through an Image Pyramid", In Proc. of the ACM Symposium on Virtual Reality Software and Technology, pp75-82, 2006.

[4] M. H. Gross, R. Gatti, and O. Staadt, "fast multiresolution surface meshing", in Proc. IEEE Visualization, pp135-142, 1995

[5] R. Pajarola. "Large scale terrain visualization using the restricted quad-tree triangulation", In Proc. IEEE Visualization, pp19-26, 1998.

[6] M. A. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes", In Proc. IEEE Visualization '97, pp81-88, 1997

[7] A. A. Pomeranz, "ROAM using surface trianle clusters(RUSTiC)", Master' s thesis, Center for Image Processing and Integrated Computing, University of California, Davis, 2000.

[8] L. M. Hwa, M. Duchaineau, and K. Joy, "Adaptive 4-8 texture hierarchies", In Proc. Visualization, pp219-226, 2004.

[9] D. Cohen-Or, E. Rich, "A Real-Time Photo- Realistic Visual Flythrough", In Proc. IEEE Visualization '96, Vol. 2, No 3, 1996.

[10] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids", In Proc. IEEE Visualization '98, pp35-42, 1998.

[11] C. C. Tanner, C. J. Migdal, and M. T. Jones, "The clipmap: A virtual mipmap", In Proc. ACM SIGGraph '98, pp151-158, 1998.

[12] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids", In Proc. ACM SIGGraph '04, pp109-118, 2004.

[13] D. Cohen and Shaked, "Photo-Realistic Imaging of Digital Terrain", Computer Graphics Forum, pp363-374, 1993.

[14] J. R. Wright, and J. C. L. Hsieh, "A Voxel-Based, forward Projection Algorithm for Rendering Surface and Volumetric Data", In Proc. IEEE Visualization '92, pp340-348, 1992.

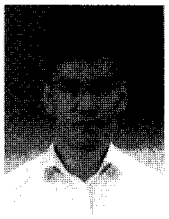
[15] C. Lee and Y. G. Shin, "An Efficient Ray Tracing Method for Terrain Rendering", In Proc. Pacific Graphics, pp180-193, 1995.

[16] M. Natalya, "Dynamic parallax occlusion mapping with approximate soft shadows", In Proc. the 2006 symposium on Interactive 3D graphics and games, pp 63-69, 2006.

[17] H. Qu, F. Qiu, N. Zhang, A. Kaufman, M. Wan, "Ray tracing height fields", Computer Graphics Intern- ational, 2003

[18] S. Ramsey, K. Potter, C. Hansen, "Ray Bilinear patch Intersections", Journal of Graphics Tools, Vol 9, No 3, 2004.

[19] T.J. Purcell, I. Buck, W.R. Mark, and P. Hanrahan. "Ray tracing on programmable graphics hardware", Proc. ACM SIGGraph '02, pp703-712, 2002.



박선용(Park Sun Yong)

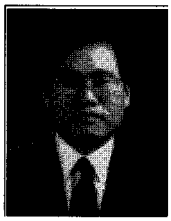
1995. 2 동국대학교 컴퓨터공학과(학사)
2006. 2 연세대학교 경영정보학과(석사)
2006. 3 ~ 현재 숭실대학교 미디어학과(박사과정)

관심분야 : Real-time Rendering, Augmented Reality, Visualization

오경수(Oh Kyoung Su)

1994년 서울대학교 계산통계학과(학사)
1996년 서울대학교 전산과학과(석사)
2001년 서울대학교 전기 컴퓨터 공학부(박사)
2001 ~ 2002(주) 조이멘트 개발팀장
2003 ~ 현재 숭실대학교 미디어학부 조교수

관심분야 : Real-Time Computer Graphics, Game



조성현(Cho Sung Hyun)

1978년 서울대학교 계산통계학과 (이학사)
1980년 서울대학교 계산통계학과 (이학석사)
1995년 UCLA 컴퓨터과학과 (이학박사)
1996년 ~ 현재 홍익대학교 게임학부 교수

관심분야: 게임프로그래밍, 게임 관련 법 및 정책, 분산시스템