점 렌더링에서 GPU를 이용한 법선 벡터 계산

조광현° 권구주 신병석
인하대학교 컴퓨터 • 정보공학과
{g2031373°, g2012154}@inhavision.inha.ac.kr, bsshin@inha.ac.kr

Normal Calculation Using GPU in Point Rendering

Kwang-Hyun Cho° , Koo-Joo Kwon , Byeong-Seok Shin
Dept. of Computer Science & Infomation Engineering, Inha University

요 약

3차원 공간상에 분포되어 있는 점들로부터 기하정보를 재구성하여 렌더링 할 때 법선 벡터가 필요 하다. 이 점들은 서로간의 연결성 정보가 없고 법선 벡터가 없기 때문에 음영 효과를 표현 할 수 없다. 본 논문에서는 점들의 연결성 정보를 추정하여 법선 벡터를 구하는 과정에서 GPU를 사용하는 방법을 제안한다. GPU로 법선 벡터를 계산할 경우 CPU의 부하가 줄게 되고 계산 시간도 감소된다. 이 방법을 점 렌더링에 적용하여 법선 벡터를 신속하게 계산하고 가시성 검사와 음영처리를 효과적으로 할 수 있도록 한다.

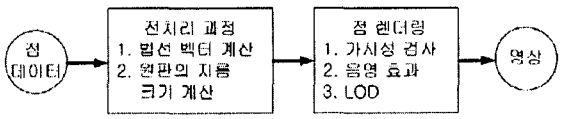
1. 서 론

3차원 공간상에 분포된 점들을 기하모델로 재구성 하는데 크게 두 가지 방법이 사용된다. 첫 번째로 삼각형 재구성 방법을 통해 3차원 모델을 만드는 방법이다. 이 방법은 점들을 삼각형화 하여 그물망(mesh)으로 만든다[1]. 렌더링 파이프라인에 최적화 된 기본 도형인 삼각형을 이용하기 때문에 객체를 효율적으로 렌더링 할 수 있다. 이 방법은 법선 벡터의 계산이 용이하여 음영효과를 정확히 표현할 수 있는 장점이 있다. 두 번째로 그물망 모델을 생성하지 않고 원하는 렌더링 속도를 낼 수 있는 점 렌더링 방법(Point Rendering)이 있다[2, 3]. 이 방법은 점 데이터를 최소 단위 기본 도형인 원판, 사각형, 타원, 구로 변형 한 후 이것을 관측면에 투영하여 렌더링 한다. 점 렌더링에서 점들의 개수가 많아지면 렌더링 속도가 감소되므로 이 문제점을 해결하기 위해 여러 방법들이 소개 되었다. 그 중에 하나는 그래픽 하드웨어를 사용하여 렌더링 속도를 높이는 방법이다[4, 5].

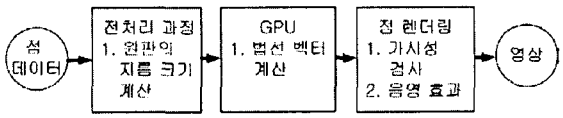
점 렌더링 방법에서 점 데이터에는 법선 벡터가 없기 때문에 음영 효과를 표현할 수 없고, 가시성 검사도 할 수 없다. 음영 효과와 가시성 검사를 위해 점들 간의 연결성(topology)을 찾아 법선 벡터를 구해야 한다. 점 렌더링의 과정은 점 데이터로부터 위치 정보를 얻고 법선 벡터와 원판의 지름의 길이를 계산하는 전처리 과정이 있다. 이 과정 후에 렌더링을 하기 위해 가시성 검사와 음영 효과, LOD(Level of Detail) 과정을 수행한다[그림 1].

본 논문에서는 점 데이터에서 점들의 연결성을 추정한 후 GPU(Graphic Processing Unit)를 이용하여 법선 벡터를 계산하는 방법을 제안한다. GPU는 병렬처리 구조를 가지고 있으므로 속도가 빠르고 CPU의 부하를 감소시켜준다. 이 방법을 점 렌더링에 적용하여 가시성 검사와 음영처리를 효과적으로 할 수 있도록 한다.

논문의 2절에서는 연결성 정보가 없는 점 데이터에서 법선 벡터를 계산하는 과정과 이를 GPU로 계산하는 방법을 자세히 설명한다. 3절에서는 본 논문에서 제안한 방법의 실험 결과를 보이고, 4절에서 결론을 맺는다.



(a) 기존 점 렌더링 방법



(b) 논문에서 제안한 방법

그림 1. 점 렌더링 방법의 흐름도

2. 법선 벡터 계산

3차원 공간상에 분포된 점들로부터 법선 벡터를 구하는 과정은 다음과 같다. 먼저 모든 점들을 대상으로 거리를 계산하여 가장 인접한 두 점을 선택한다. 또한 점 데이터들 가운데 잡음(noise) 점을 제거하기 위해 주변 점들과의 거리가 임계값(e) 보다 큰 점은 제거한다. 다음으로 인접한 두 점을 이용하여 GPU 상에서 법선 벡터를 구한다.

2.1. 인접점 계산

점 데이터는 위치 정보(좌표)만을 가지고 있기 때문에 점들 간의 연결성 정보를 알 수 없다. 점들의 연결성 정보를 추정하기 위해 인접한 두 점을 찾는다.

한 점으로부터 나머지 모든 점들과의 거리 계산하여 인접한 두 점을 구한다. 점 데이터를 얻어오는 장치의 오류로 인하여 점 데이터 안에 잡음 점이 있을 수 있다. 잡음 점을 제거하기 위해서 임계값(e)을 적용하여 다른 점들과의 거리가 이 보다 큰 점들은 인접점 계산에서 제외시킨다. [그림 2]는 이 과정을 나타낸 의사코드이다.

```
ComputeTopology( Point Array Pt,
                 Point Array Result )
{
   Array dist1, dist2;
   Point Array tempPt1, tempPt2;
   for( all points i in Pt)
      for ( all the remaing points j in Pt)
         dist = EuclidianDistance(Pt(i), Pt(j))
         if (dist < e )
            if(dist < min_dist1)
               min_dist1 = dist1;
               tempPt1 = Pt[j];
            if(dist > min_dist1 &&
              dist < min_dist2)
               min_dist2 = dist;
               tempPt2 = Pt[j];
         result[i] = tempPt1, tempPt2;
      next j
    next i
}
```

그림 2. 인접한 두 점을 계산하기 위한 의사 코드

2.2. GPU를 이용한 법선 벡터 계산

여기서는 GPU를 이용하여 법선 벡터를 계산하는 방법을 소개한다. GPU를 이용한 벡터 연산은 CPU의 벡터 연산보다 빠르고 CPU의 부하를 줄일 수 있다. 또 GPU에서는 프로그래밍이 가능하여 벡터 연산과 조명 연산을 할 수 있다. GPU를 이용하여 벡터 연산을 사용하기 위해서는 어셈블리어와 DirectX 9.0에서 제공하는 HLSL(High Level Shader Language)라는 언어를 사용하여야 한다. 쉐이더는 정점 쉐이더와 픽셀 쉐이더로 나누어진다.

본 논문에서는 어셈블리 언어로 작성하고 정점 쉐이더 버전 2.0을 이용하여 법선 벡터를 계산하는 방법을 설명한다. 정점 쉐이더를 사용하기 위해서는 쉐이더 버전을 정의 하고 입력 값에 대한 정의가 필요하다. [그림 3]은 정점 쉐이더 프로그램의 내용이다. 여기서, 입력 값으로 세 점의 위치 값을 받는다. 법선 벡터를 계산할 점 v0와 v0에 인접한 두 점 v1, v2 이다.

두 점 v1와 v2로 벡터 연산을 하여 v0에서 v1으로 향하는 벡터와 v0에서 v2로 향하는 벡터를 계산한다. 다음으로 이 두 벡터를 정점 쉐이더에서 지원하는 *crs(dest, src0, src1)* 함수를 이용하여 외적(cross product) 계산을 하고 단위 벡터로 만들기 위해 정규화 과정을 거친다. 이 결과 값으로 나온 법선 벡터는 r3 레지스터에 저장된다.

```
vs_2_0                ;쉐이더 버전
dcl_position v0       ;법선 벡터를 계산할 점의 좌표
dcl_position v1       ;인접한 두 점 v1 , v2
dcl_position v2

add r1, v1, -v0       ;벡터 연산
add r2, v2, -v0

crs r3, r1, r2        ;외적 연산

mov oPos, v0          ;출력 위치 값

dp3 r3.w, r3, r3      ;법선 벡터 정규화
rsq r3.w, r3.w
mul r3, r3, r3.w
```

그림 3. 법선 벡터 계산을 위한 정점 쉐이더

3. 실험결과

실험은 펜티엄 IV 2.0GHz CPU와 768MB Main Memory, NVIDIA GeForce FX 5600을 장착한 컴퓨터에서 수행하였다. GPU를 이용한 정점 쉐이더는 DirectX 9.0에서 지원하는 쉐이더 2.0 어셈블리어를 사용하였다.

실험을 위해서 약 2만개(Data A), 약 5만개(Data B)의 점으로 구성된 데이터를 사용하였다. 실험은 동일한 환경에서 CPU와 GPU를 사용하여 각 점 데이터의 법선 벡터 계산 시간을 측정하였다. [표 1]에서 보는 것과 같이 Data A로 실험 했을 때 GPU를 사용했을 경우가 CPU일 때보다 36%정도 빨랐다. 점의 개수가 증가될수록 CPU보다 GPU를 사용했을 때 효과가 큰 것을 알 수 있다.

결과 영상에서의 차이를 확인하기 위해 약 56만개의 등잔 모형과 약 46만개의 용 모형을 대상으로 영상을 생성하였다. [그림 4]는 본 논문에서 제안한 방법으로 생성한 영상이다. 점 렌더링 방법으로 렌더링 하였고 그림에서 보는 것 같이 법선 벡터가 잘 적용되어 음영 효과가 잘 표현된 것을 알 수 있다.

(단위 : 초)

	CPU	GPU
Data A	0.59	0.38
Data B	2.13	0.57

표. 1 CPU와 GPU의 법선 벡터 계산 시간 비교

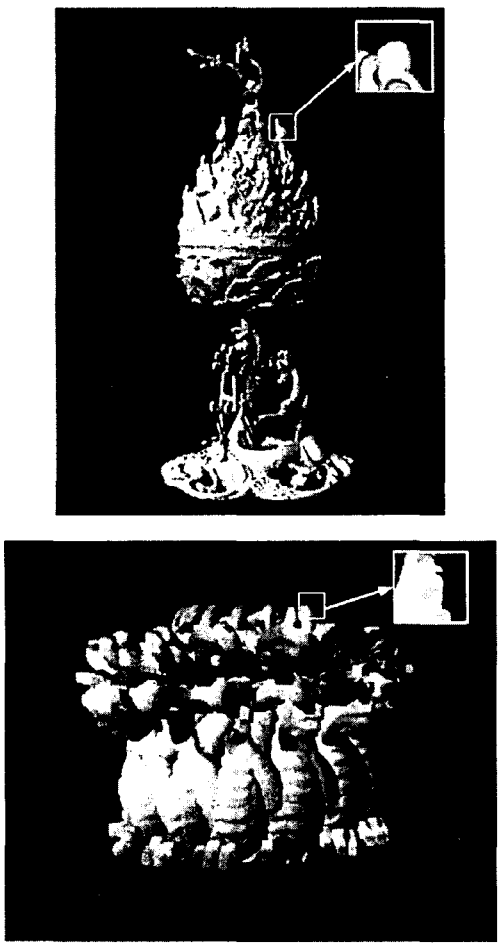


그림 4. 등잔 모형(위)과 용 모형(아래)을 점 렌더링으로 생성한 영상

4. 결론 및 향후 과제

본 논문에서는 3차원 공간에 분포된 점들에서 법선 벡터를 구하기 위해 GPU를 이용하는 방법을 제안하였다. 각 점과 인접한 두 점들로 부터 정점 쉐이더 언어를 이용하여 법선 벡터를 계산 하였다. 이 법선 벡터를 점 렌더링에 적용하여 영상을 생성하였다. 본 논문의 방법으로 법선 벡터를 계산하는 시간을 단축시킬 수 있었고 결과 영상을 통하여 계산 값을 확인 할 수 있었다. 또한 CPU의 부하를 줄여 CPU에서의 다른 작업을 할 수 있었다.

향후 연구할 과제는 GPU에 최적화된 연산을 위한 점들의 자료구조를 설계하는 방법과 점 렌더링에서 빠른 전처리 과정을 위한 자료구조를 설계하는 방법에 대한 연구가 필요하다.

참고 문헌

[1] D. Cohen-Steiner, E. Colin de Verdiere, and M. Yvinec, *Conforming Delaunay triangulations in 3D.* Proc. 18 th Annu. Sympos. Comput. Geom. , pp. 199-208, 2002

[2] M. Levoy and T. Whitted, The use of points as a display primitive. Computer Science Department, *University of North Carolina at Chapel Hill*, In TR 85-022, 1985

[3] S. Rusinkiewicz and M. Levoy, QSplat: A Multiresolution Point Rendering System for Large Meshes, In Computer Graphics, SIGGRAPH 2000 Proceedings, pp. 343-352. Los Angeles, CA, July 2000

[4] L. Coconu and H. Hega, Hardware-accelerated point based rendering of complex scenes. In proceeding of Eurographics Workshop on Rendering pp. 41-51, 2002

[5] M. Botsch and L. Kobbelt, High-Quality Point-Based Rendering on Modern GPUs. In Proc. 11th Pacific Conference on Computer Graphics and Applications (PG'03) October 08-10, Canmore, Canada. pp. 335-343 , 2003