대용량 폴리곤 데이터 편집을 위한 자료구조

권 대현, 김 해동, 오 광만

한국전자통신연구원 가상현실연구개발센터

A Data Structure for Editing Very Large Polygon Data Set

Daniel D. Kwon, Hae-Dong Kim, Kwang-Man Oh

Virtual Reality R&D Center

Electronics and Telecommunications Research Institute(ETRI)

{dkwon, hdkim, okman}@etri.re.kr

요 약

이 논문에서는 대용량 3 차원 데이터를 효율적 편집과 실시간 랜더링하는 방법과 자료구조를 제안한다. 3 차원 스캐닝 데이터로부터 만들어지는 대용량 폴리곤 데이터를 사용자 인터페이스를 이용하여 실시간에 편집하기 위한 효율적인 자료구조를 제안하고, 이를 이용한 폴리곤 연산자의 구현 방법 설명하고 있다. 또한, 여기서 제안한 자료구조가 기존 edge 기반 자료구조와의 성능비교를 통해 대용량 메쉬 데이터 편집시스템에 적합한 자료구조임을 보이겠다.

1. 서 론

3 차원 스캐닝에 의한 모델 생성방법이 전통적인 모델링의 대안으로 부상하면서 3 차원 스캐너 후처리 소프트웨어에 대한 관심이 높아지고 있다. 그러나, 3 차원 스캐너를 통해 얻어진 폴리곤 데이터의 크기가 기존 모델링작업을 통해 얻어진 데이터에 비해 워낙 크고, 여러 종류의 오류 거리정보(error data points)를 포함하고 있어 3 차원 스캔 데이터를 그대로 사용할 수는 없고, 후처리 과정을 통해 사용가능한 크기의 오류 거리정보 없는 데이터로 변환하게 된다. 이 과정에서 대용량 데이터를 사용자 인터페이스를 이용하여 편집하고, 편집된 결과를 실시간에 렌더링하는 방법을 필요로 한다.

기존의 3 차원 스캐닝 후처리 소프트웨어에서는 후처리 과정을 3 차원 그랙픽 모델러나 CAD 프로그램의 연장선으로 보고 대용량 데이터를 winged edge[2]기반의 half-edge[2]나 quad-edge[3]기반의 자료구조를 사용하여 폴리곤 데이터를 편집하여 왔다. 그러나, 3 차원 스캐닝 데이터는 기존의 3 차원 모델링 시스템이 수용하는 폴리곤 수보다 수십에서 수백배 많은 데이터가 입력으로 주어 지므로 효율적인 자료구조의 설계가 필수적이며, 이 데이터를 빠른 시간내에 랜더링하기 위한 원시데이터(primitive data) 접근방법을 제공해야 한다.

여기서 제안하는 자료구조는 기존의 edge 기반 자료구조 대신 no-edge 기반 자료구조 표현 방법인 LTL(Lawson's Triangle List)[1]를 사용하고 있다. 이 자료구조는 edge 기반 자료구조에 비해 저장 및 폴리곤 삽입/삭제시 갱신하야 할 정보가 적어지므로 후처리 소프트웨어의 수행속도를 높일 수 있을뿐 아니라, Edge 기반 자료구조처럼 constant time 에 인접 폴리곤의 정보를 얻을 수 있다. 또한, edge 기반의 자료구조보다 효율적으로 랜더링 원시 데이터[(rendering primitives)[4] 결점(vertex)이나 삼각면(triangle)에 접근할 수 있으므로 랜더링 속도를 향상시킬 수 있다.

본 논문에서 LTL 구조, edge 자료구조와 의 비교 및 no-edge 자료구조에서의 폴리곤 연산에 대해 설명하고, edge 기반 자료구조로 완성된 시스템과 성능비교를 통해 이 논문에서 제안된 자료구조로의 효율성에 대해 밝히겠다.

2. 제안된 자료구조

이 장에서는 기존 edge 기반 자료구조와 no-edge

기반 자료구조에 대해 설명하고, 이 논문에서 제안된 no-edge 구조인 LTL 구조 이용한 폴리곤 연산자의 구현에 대해 설명하겠다.

2.1Edge 기반 자료구조

edge 기반 자료구조는 물체를 구성하는 폴리곤의 edge 를 중심으로 편집 및 렌더링이 수행된다. 예를 들어, 3 개의 결점(vertex)가 3 개의 edge 를 이루고 이 3 개의 edge 들이 한 개의 삼각형을 구성할 때, edge 기반 자료구조는 그림 1 과 같이 왼쪽부터 결점배열, edge 배열, 삼각형이 저장된다.

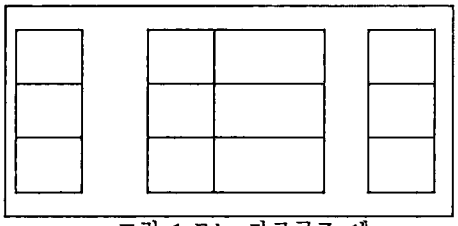

그림 1. Edge 자료구조 예

여기에 더하여 edge 의 방향성을 나타내는 정보되고, edge 구조의 종류에 따라 인접 삼각형의 정보가 추가된다.

2.2 No-edge 기반 자료구조

No-Edge 기반 자료구조는 edge 에 관한 정보를 없애고, 단순히 결점배열과 삼각형를 이루는 3 결점으로 표현한다. 그림 1 에 표현된 삼각면을 no-edge 구조로 표현하면 그림 2 와 같이 표현된다.

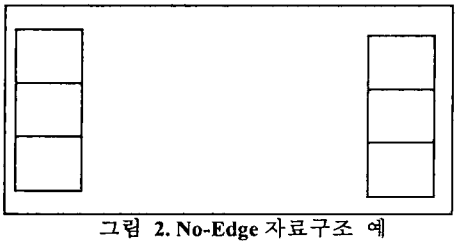

그림 2. No-Edge 자료구조 예

또한, 인접 삼각형의 정보가 더해져 no-edge 기반 자료구조를 이룬다.

2.3 LTL 자료구조

LTL 구조는 2.2 에서 설명한 no-edge 자료구조의 하나로 삼각면을 이루는 세 결점(vertex)와 삼각형의 세 edge 와 인접한 삼각형의 정보를 가지고 있다.

그림 1.3 에서 처럼 16 개의 결점과 이들로 이루어진 18 개의 삼각형 메쉬가 있을 때 굵은 검정색으로 표시된 삼각형의 LTL 구조는 다음과 같이 표시된다.

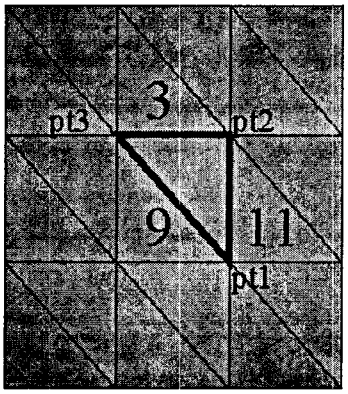

그림 3. 삼각형데이터의 예

삼각형 결점 = { pt1, pt2, pt3}
인접삼각형의 인덱스 = { 3, 9, 11 }

No-edge 자료구조는 edge 자료 구조와 달리 삼각메쉬들의 방향성, 즉 clockwise(CW) 또는 counter clockwise(CCW),을 묵시적으로 선언 하므로 edge 자료구조처럼 각 삼각면의 edge 마다 방향성을 선언할 필료가 없다. 특히, 3 차원 스캐너의 데이터의 경우 CW 나 CCW 메쉬가 혼재된 경우가 없으므로 no-edge 자료구조가 매우 유용하다. 또한, edge 구조 처럼 랜더링 원시데이터(primitive)인 점과 삼각형에 접근하기 위해 edge 들을 거칠 필요가 없으므로 3 차원 스캐너처럼 대용량 메쉬 데이터를 렌더링하는 경우 훨씬 효율적으로 다룰 수 있다.

또한, 대용량 데이터를 편집하기 위한 메모리 요구량 역시 LTL 구조가 edge 기반 자료구조보다 효율적임을 알 수 있다. 그림 3 의 삼각형 메쉬를 저장하기 위해 필요한 31 개의 edge 와 edge 방향성에 대한 정보 필요 없고, 및 한 edge 를 기준으로 인접한 두개의 삼각면에 대한 정보가 1 개로 줄일 수 있으므로 3 차원 스캐너처럼 대용량 메쉬 데이터의 실시간 편집이 요구되는 시스템에 적합하다.

3. 폴리곤 연산자 구현

Edge 자료구조를 이용한 삼각형 메쉬편집에 쓰이는 기본 연산자가 no-edge 구조에서도 잘 작동할 수 있음을 보이겠다. 이 장에서는 모든 폴리곤 연산 중 기본이되는 연산자인 삽입/삭제 및 loop 연산자 구현

알고리즘에 대해 설명하겠다.

3.1 폴리곤 삽입

이미 존재하는 폴리곤 메쉬의 포인트를 이용하여 하나의 폴리곤을 더하는 연산은 다음 같이 정의할 수 있다.

```
void addTriangle( Vertex pt[3]) {
   triangle tri = CCW( pt[0], pt[1], pt[2]);
   for ( i = 0; i < 3; i++)
      loop[i] = Mesh.loop (pt[i]);

   for ( i = 0; i < 3; i++) {
   find = FALSE;
   for ( j = 0; j < loop[i].size(); j++) {
       triangle t = loop[i].getTriangle(j);
       if ( t.includeEdge( pt[j], pt[(j +1)% 3]) ) {
           tri.adj[i] = t;
           find = TRUE;
      }
   if ( !find ) tri.adj[i] = null;
   }
}
```

3.2 폴리곤 삭제

폴리곤 메쉬에서 하나의 폴리곤을 빼는 연산은 다음 같이 정의할 수 있다.

```
void deleteTriangle( TriangleID id) {
   triangle tri = Mesh.getTriangle(id);

   for ( i = 0; i < 3; i++) {
       if ( tri.adj[i] == null ) continue;
       triangle t = Mesh.getTriangle(tri.adj[i]);
       for ( j = 0; j < 3; j++)
         if ( t.adj[j] == t ) t.adj[j] = null;
      }
   }
   Mesh.deleteTriangle(id);
}
```

3.3 폴리곤 루프

폴리곤 메쉬에서 한 점을 포함하는 삼각형의 루프는 연산은 다음 같이 정의할 수 있다.

```
LOOP loop( VertexID id) {
   triangle tri = Mesh.containVertex(id);
   int index = t.getIndex(id);

//search in clockwise direction
   t = tri;
   while ( t.containsVertex(id) {
   loop.add(t);
   t = t.adj[(index+2) % 3];
   index = t.getIndex(id);
  }

//search in counter-clockwise direction
   t = tri;
   while ( t.containsVertex(id) {
   loop.add(t);
   t = t.adj[(index+1) % 3];
   index = t.getIndex(id);
   }

 return loop;
}
```

3.4 기타 구현시 고려사항

3 차원 스캐닝 데이터의 특징은 후처리 과정에서 대용량 메쉬 데이터 편집 후 90%이상의 스캔 데이터가 소멸하는 경향이 있다. 이 같은 경향은 3 차원 스캐너의 데이터가 상당히 조밀하다는 사실에 기인한다. 이 같은 데이터의 특성 때문에 연산이 가해질 때 마다 지워지는 폴리곤의 숫자가 지속적으로 증가한다. 구현시 폴리곤에 할당된 메모리를 지우는 연산은 매우 시간이 오래 걸리는 연산이므로 폴리곤을 지울 때, 실제 할당된 메모리공간은 지우지 말고 표시만 해두었다가, 특정한 개수 이상의 폴리곤이 지워질 때 실제 메모리공간을 지우는 것이 효과적이다. 또한, 대용량 데이터를 다루는 시스템이므로 체계적인 메모리 관리 정책이 필요하다.

4. 성능비교

Edge 자료구조를 사용한 시스템과의 비교를 위해 edge 구조 중 일반적으로 사용되는 winged edge 구조와 비교하였다. 이 실험에서는 n * m 의 격자형 데이터를 그림 3 과 같은 형태의 폴리곤 집합으로 저장하였을 때, 두 자료구조의 메모리 효율성, 연산자 효율성 및 랜더링 효율성을 계산하였다.

4.1 메모리 요구량

n x m 의 격자형 거리정보 데이터(range data)를 저

장하는데 필요한 공간은 다음과 같다.

결점(Vertex)저장공간 S(v) = n * m
폴리곤 저장공간 S(p) = (n-1) * (m -1)*2
Edge 저장공간 S(e) = n*(n-1)+m*(n-1)+(m-1)*(n-1)
= 3mn – 2n – 2m + 1
인덱스저장공간 I(n) = the number of elements in S(n)

따라서, winged edge 의 경우 결점, edge 및 폴리곤을 저장하기위한 공간과 Edge 의 양쪽 인접 삼각형 정보와 폴리곤 각 Edge 의 방향성에 관한 정보가 추가되어야 하므로,

메모리 요구량
M(h) = S(v) + S(p) +S(e) + 2I(S(e)) + 3I(S(p))

LTL 구조의 경우 결점 및 폴리곤을 저장하기위한 공간과 각 삼각면의 Edge 마다의 인접 삼각면 정보가 필요하므로,

메모리 요구량 M(l) = S(v) + S(p) + 3I(S(p))
= S(v) + 4S(p)

따라서, LTL 구조가 half-edge 구조보다 3S(e) 만큼의 메모리 효율성을 가짐을 알 수 있다.

4.2 연산자 시간복잡도

연산자의 효율성을 계산하기 위하여 3 장에서 제시한 원시 연산자(primitive operation)의 시간복잡도(time complexity)와 갱신정보량을 비교하였다. Half-edge 구조의 연산자들의 시간복잡도와 LTL 구조의 시간 복잡도는 모두 선형시간(linear time)이다. 그러나, LTL 구조의 경우 정의된 메쉬방향의 반대방향 탐색의 경우 2 번의 접근이 필요하므로, 두 자료구조의 시간복잡도는 다음과 같이 정의된다.

Half-Edge 구조의 연산 시간복잡도 CT(h) = c
LTL 구조의 연산 시간복잡도 CT(l) = 2c,
c : 인접삼각형을 방문하는데 걸리는 시간

또한, 두 자료구조의 데이터 갱신 시간효율은 각 자료구조가 가지고 있는 인접 정보의 양에 비례하므로.

Half-Edge 구조의 갱신시간
U(h) = k * (2I(S(e)) + 3I(S(p)))
LTL 구조의 갱신시간 U(l) = k * 3I(S(p))
k : 한 개의 데이터를 갱신하는 시간

위에서 보는 것처럼 Half-Edge 와 LTL 구조의 연산자 효율성은 연산 시간복잡도에서 Half-Edge 가 우수하며, 갱신시간 효율면에서는 LTL 구조가 우수한 것으로 나타난다. 그러나, m 과 n 이 커짐에 따라 I(S(e))값의 증가분이 더욱 커지므로 메모리 갱신속도가 시스템 성능에 커다란 영향을 미치게 된다. 따라서, 대용량 메쉬 데이터의 경우 LTL 의 성능이 우수하게 나타난다.

4.3 랜더링 효율성

렌더링 효율성을 렌더링 요소(Primitive)인 결점(Vertex)과 삼각면의 접근시간으로 나타낼 수 있다. 두 구조 모두 결점 접근시간은 같고, Half-Edge 의 경우 삼각면에 접근하기위해 각 Edge 마다 두번의 결점 접근 그리고 방향성 정보에 접근해야 한다. 따라서,

Half-Edge 구조의 결점 접근시간 RV(h) = c
LTL 구조의 결점 접근시간 RV(l) = c

Half-Edge 구조의 삼각면 접근시간 RT(h) = 7c
LTL 구조의 결점 접근시간 RT(l) = 3c
c 는 렌더링 원시데이터 접근시간

따라서, LTL 구조의 접근시간이 2 배이상 빠르다는 것을 알 수 있다.

5. 결론

이 논문에서 LTL 구조가 대용량 폴리곤 데이터 편집시스템을 유용하게 사용될 수 있음을 보았다. 앞에서 언급한 것처럼 LTL 구조는 메모리 요구량, 연산자 시간복잡도, 랜더링 효율성면에서 Edge 구조의 편집 시스템보다 우수하다는 것을 알 수 있다.

LTL 구조를 사용하는 메쉬 편집시스템은 대용량 메쉬 데이터의 대화형 편집시스템에 적합하다. 특히 3 차원 스캐너 후처리 작업은 일반 3 차원 그래픽 모델러처럼 대용량의 메쉬 데이터가 존재하고, 프로그램이 Edge 를 직접 다루지 않고, 메쉬 데이터 전체가 삼각면으로 이루어지므로 매우 적합한 자료구조이다.

참고로 LTL 구조 사용한 3 차원 후처리 소프트웨어는 한국전자통신연구원 기술이전을 통해 상용화되었음을 밝힌다.

참고문헌

[1] C. L. Lawson. "C1 Surface Interpolation for Scattered Data on a Sphere" Rocky Mt. Journal of Mathtatics, Vol 14, No 1. 1984
[2] Edelsbrunner H., Algorithms in Combinatorial Geometry, Springer-Verlag, Heidelberg, Germany, 1987.
[3] Guibas L., Stolfi J., ``Primitives for the manipulation of General Subdivisions and Computation of Voronoi Diagrams", ACM Transactions on Graphics, Vol. 4, No. 2, April 1985, pp. 74-123.
[4] OpenGL Programming Guide 2nd Edition, Addison-Wesley Press, 1997