영역트리를 이용한 효율적인 세밀정도제어

황주영, 이종현, 김경호, 임상석, 박규호
한국과학기술원 전기 및 전자공학과 컴퓨터공학연구실
Email: jyhwang@computer.org

An Efficient Level-of-detail Control of Meshes by Region Tree

Joo Young Hwang, Jong Hyun Lee, Kyung Ho Kim, Sang Seok Lim, Kyu Ho Park
Computer Engineering Laboratory, Electrical Engineering Department
Korea Advanced Institute of Science and Technology

요 약

기존의 LOD 제어방법들은 렌더링속도를 성공적으로 증가시켜왔으나 오버헤드가 크다는 단점을 갖고있다. 이러한 오버헤드는 각 vertex 마다 view-frustum clipping, back-face culling, 스크린 공간 기하학적 오차계산과 같은 view-dependent refinement criteria 를 측정하고, 메쉬의 LOD 를 바꾸기 위해서 edge collapse/vertex split 를 수행하기 때문이다.

제안하는 방법은 메쉬를 여러 개의 region 들로 나누고 vertex 가 아닌 region 에 대해 view-dependent refinement criteria 를 측정하므로 오버헤드가 훨씬 작다. 또한 각 region 들의 LOD 가 바뀔 때 미리 만들어 둔 LOD 버전들중에서 하나를 선택하기만 하면 되므로, edge collapse/vertex split 을 수행하는 오버헤드는 없다. 실험적으로 제안하는 LOD 제어방법은 기존의 방법들보다 작은 메모리를 사용하고 LOD 제어 오버헤드도 적으며, LOD 제어를 하지 않은 경우보다 2 배 – 5 배의 렌더링 속도향상을 얻었다.

1. 서 론

3 차원 그래픽에서 사용되는 메쉬는 그 복잡도가 빠르게 증가하여 렌더링 가속기를 사용하여 실시간에 렌더링할 수 없다. *Level-of-detail*(LOD)제어방법은 메쉬의 세밀정도를 제어하므로써 렌더링속도를 향상시키는 방법이다. view-dependent LOD 제어방식은 시점에 따라서 시점과 가까운 부분은 자세한 LOD 를 사용하고, 시점에서 먼 부분은 LOD 를 떨어뜨리는 방식이다.

최근에 제안된 *view-dependent* LOD제어 방법들은 대부분 edge collapse(ecol)/vertex split(vsplit)을 사용하여 메쉬의 LOD를 변화시킨다. 그러한 ecol/vsplit을 찾기 위해서 vertex마다 시점과의 관계를 조사하여야 하며, 이를 위해서 vertex마다 view-frustum *culling*, back-face *culling*, 스크린 공간 기하학적 오차산출이 요구된다. 이러한 과정은 상당한 오버헤드를 갖는다.

본 논문에서는 오버헤드가 적은 LOD제어방법을 제안한다. 제안하는 방법은 메쉬를 여러 개의 patch 또는 영역(region)들로 나누고, 각 *region*의 여러 개 LOD 버전들을 미리 만들어둔다. LOD 제어는 vertex가 아닌 region에 대해 clipping, back face *culling*을 수행하며, 각 *region*들마다 렌더링할 LOD 버전을 결정한다.

제안하는 방법은 실험적으로 LOD 제어를 하지 않았을 때와 비교하였을 때 시점에 따라서 2 – 5배 정도 렌더링속도를 향상시켰다. 또한 메모리의 요구량과 오버헤드가 기존의 방법보다 훨씬 작았다.

논문의 구성은 다음과 같다. 2절에서는 기존의 *view-dependent* LOD 제어방법들을 알아보고 문제를 정의한 다음 제안하는 *LOD*제어방법의 개요를 제시한다. 3절과 4절에서는 LOD 제어방법을 자세히 기술하며, 5절에서 실험결과를 제시하고, 6절에서 결론을 내린다.

2. 기존 연구 동향

LOD제어가 적용된 후의 메쉬에 존재하는 vertex나 face는 LOD제어를 하지 않은 메쉬의 vertex,face의 subset이고, 각각을 active vertex, active face라고 부른다.

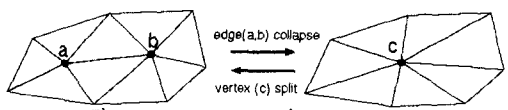


그림 1. Edge collapse(ecol)과 vertex split(vsplit)

Xia[1]는 메쉬를 그림 1과 같이 edge collapse(ecol) 연산을 사용하여 단순화시키고, collapse되는 edge의 두 끝점은 edge 가collapse된 다음에 남은 점의 자식으로 정하므로써 vertex간의 계층구조를 정의하였다. 이러한 점들간의 계층구조를 merge tree로 만드는 데, 메쉬가 주어지면 가능한 한 많은 ecol을 생성시키므로써 merge tree가 O(logN)의 깊이를 갖도록 한다. LOD제어는 매 프레임마다 merge tree로부터 active vertex 를 추출하고 이를 이용하여 active face를 찾아서 메쉬를 렌더링한다..

View-dependent refinement of progressive meshes(VDPM) [2]에서는 Progressive meshes(PM)[3]를 view-dependent LOD제어에 사용한다. [1]과 마찬가지로 vertex 계층구조를 구성하며, 이로부터 active vertex를 추출한다. 그러나 [1]과 달리 PM으로부터 vertex 계층구조를 구성하기 때문에 기하적으로 최적화되어 있어서 동일한 스크린 공간 오차한계에 대해서 [1]보다 적은 폴리곤 개수를 갖는 메쉬를 생성한다.

기존 방식들[1][2]은 매 프레임마다 active vertex를 찾기 위해서 vertex 계층구조를 traverse하면서 각 vertex들이 refine되어야 하는 경우 vsplit을 수행하고, simplify되어야 하는 경우 ecol 을 수행한다. Vertex 가 refine되어야 하는 지를 결정하기 위해서 view-frustum culling, back-face culling, 스크린 공간 기하학적 오차와 같은 view-dependent refinement criteria를 사용한다. 이러한 view-dependent refinement criteria를 계산하는 것과 ecol/vsplit을 수행하는 것은 큰 오버헤드이다. 따라서 본 논문에서는 기존방법들보다 오버헤드가 적은 새로운 LOD제어방법을 제안한다.

제안하는 방법은 다음과 같다. 메쉬가 주어지면 여러 개의 region 들로 나눈다음, 각 region들마다 LOD 버전들을 생성한다. 그런 다음 region들을 병합하여 더 큰 region을 생성하고, 새로 생긴 region 의 LOD 버전을 생성한다. 이때 병합된 region들은 병합으로 새로 생긴 region의 자식이 된다. 이러한 region의 부모-자식관계로부터

region hierachy를 구성한다. Region hierachy의 구성은 off-line으로 preprocessing되며 결과는 디스크에 저장한다.

Interactive Visualization에서는 디스크로부터 region hierarchy를 로드하고, 사용자의 시점변화에 따라서 메쉬의 LOD를 제어한다. 이 때 region hierarchy에서 렌더링할 region들을 추출하게 되는 데, 이것을 active region이라고 한다. 이 때 active region의 LOD version중에서 사용자가 준 스크린 공간 오차한계를 넘지 않는 버전이 선택되어진다. 제안하는 LOD 제어방식의 특징은 다음과 같다.

view-dependent refinement criteria를 region level에서 수행한다. 이로써 vertex마다 하는 경우보다 훨씬 적은 오버헤드를 갖는다.

region의 LOD가 변화해야 하는 경우 미리 만들어둔 LOD 버전들 중에서 하나를 선택하여 렌더링한다. 기존의 방법들과 같이 ecol/vsplit이 쓰지 않으므로 빠르다. 또한 기존방법들이 사용하는 *vertex hierarchy*는 많은 메모리를 요구한다. 제안된 방식은 *region* hierarchy이므로 메모리가 감소한다.

3. Interactive Visualization

이 절에서는 논문의 핵심이 되는 LOD제어방법을 설명한다. LOD 제어는 off-line으로 미리 만들어진 region tree에서 active region을 추출해내고, 각 active region들마다 스크린 공간 오차가 스크린 공간 오차한계를 넘지않는 LOD버전을 선택하고 렌더링하는 것이다.

Active region 을 찾는 것은 temporal coherency를 사용하는 데, 이전 프레임에서의 active region을 담고 있는 active region list(ARL)를 traverse하면서 다음과 같은 view-depentdent refinement criteria를 적용한다.

1. frustum culling
2. back face culling
3. 스크린 공간 기하학적 오차
4. local illumination

region은 그것의 bounding sphere가 view frustum외부에 있으면 frustum culled된 것이다. region이 back face culled되려면 region내의 모든 vertex에서의 법선벡터가 시점과 back facing이어야 한다. 이것을 테스트하기 위해서 법선벡터 cone[4]방법을 이용하고, [6]에서와 같이 테스트한다.

region의 광원과의 관계를 사용하여 LOD를 제어하는 방법은 다음과 같다. Local illumination은 ambient term, diffuse term, specular term으로 이루어진다. Diffuse term을 고려하기 위해서 region이 광원과 back facing이면 diffuse term은 거의 없다. 따라서 이러한 region은 가장 단순한 LOD버전을 사용한다. 또한 specular term을 고려하기 위해서 region이 광원과 시점사이의 halfway vector와 front facing이면 highlight된 것이므로 세밀한 LOD버전을 사용한다.

ARL은 up-switch/down-switch에 의해서 갱신되어진다. Region에서의 up-switch는 ARL의 후미가 region의 부모에 더해지고 region의 모든 형제들이 ARL에서 제거되어진다. 한 region이 부모를 가지고 있고 그것의 모든 형제들이 active이며 부모가 down-switch하지 않는다면 그 region에서 up-switchable 이다. region에서의 down-switch는 ARL에서 그 region을 제거하고 그것의 모든 자식들을 ARL의 후미에 더하는 것이다.

$$\tau = \frac{k}{\|\vec{R}-\vec{E}\|}, where\ k = \frac{w}{2\tan(\frac{\phi}{2})} \qquad (1)$$

$$e_{scr} = \tau \cdot e_{obj}$$

ARL의 region R이 frustum culled되거나 back face culled 되어 시점에서 전혀 보이지 않는 경우 R은 렌더링 하지 않고, R에서 upswitchable한 경우 up-switch한다. 만약 R이 시점에서 보이는 경우 R의 LOD 버전들 중에서 스크린 공간 기하학적 오차가스크린 공간 오차한계를 넘지 않는 버전을 선택하여 렌더링한다. LOD버전들의 스크린 공간 기하학적 오차 e_{scr} 를 구하기 위해서 LOD 버전들의 기하학적 오차 e_{obj} 는 수식 (1)에서와 같이 view-parameter τ로 곱해진다. 수식 (1)에서 $\vec{R}$ 은 region의 bounding sphere의 중심이고, $\vec{E}$ 는 시점의 위치이며, w는 viewport 의 너비, ϕ는 camera의 field of view각도이다. 특히 R이 highlight된 경우에는 τ를 weighting factor로 곱해주어 보다 세밀한 LOD버전을 선택한다.

한편 R의 LOD 버전들 중에서 렌더링할 LOD버전을 찾을 수 없는 경우는 다음의 두 가지가 있다. 첫째, R의 모든 LOD 버전들의 e_{scr}이 스크린 공간 오차한계보다 작은 경우에는 R이 필요이상으로 세밀하므로 더욱 단순화시키기 위해서 up-switch해야 한다. 둘째, R의 모든 LOD 버전들의 e_{scr}이 스크린 공간 오차한계보다 큰 경우 R을 더 세밀하게 하기 위해서 down-switch해야 한다.

$$\varepsilon = \frac{\left| \frac{k}{\|\vec{R}-\vec{E}\|} - \frac{k}{\|\vec{v}-\vec{E}\|} \right|}{\frac{k}{\|\vec{v}-\vec{E}\|}} \qquad (2)$$

수식 (1)의 τ는 region내에 있는 모든 vertex들의 view parameter를 대표하므로, vertex v에 따라 수식 (2)에서와 같이 오차 ε를 갖게 된다. ε은 (R의 크기/R의 시점까지 거리)에 비례하므로 ε를 충분히 작은 값을 유지하려면 region의 크기는 메쉬가 시점에 가까울 때 작고, 메쉬가 시점에서 멀 때 커야 한다. 한편 τ는 메쉬가 가까운 경우 크고, 메쉬가 멀어지면 작아지므로 메쉬가 가까운 경우 작은 region들은 보다 세밀한 LOD버전을 갖고 있는 것이 바람직하고, 메쉬가 멀어진 경우의 큰 region들은 보다 단순한 LOD버전을 갖는 것이 좋다. 만약 작은 region이 세밀한 LOD버전부터 아주 단순한 버전까지 갖고 있는 경우 메쉬가 멀어지더라도 작은 region들이 계속 active region으로 남아있게 되고, 따라서 작은 region들간의 경계부분은 단순화되지 못하고 남아있게 된다. 이러한 문제점을 해결하기 위해서는 region의 LOD버전을 만들 때 LOD버전의 오차가 특정 상한값을 넘지 않도록 제어한다. LOD버전 오차의 상한값은 region의 크기에 비례하도록 조정한다.

4. Region tree의 생성

이 절에서는 LOD제어를 위해서 필요한 region tree를 생성하는 방법을 설명한다. 메쉬에서 다양한 크기의 region을 만들고, region의 LOD버전들을 만드는 것이 핵심이다.

다양한 크기의 region들은 계층적으로 형성되며 region tree로 저장된다. 즉 leaf region들은 아주 작은 크기이고, 윗 레벨로 갈수록 크기가 커진다. Region tree는 다음과 같이 bottom-up으로 형성된다. 메쉬가 주어지면 leaf region들을 생성하고, 각 region들을 단순화시키면서 LOD 버전을 만든다. 그런 다음 leaf region들을 merge하여 새로운 수준의 region을 만든다음 마찬가지로 단순화를 시킨다. 이러한 과정을 반복하여 region tree를 만든다.

메쉬의 leaf region을 만들기 위해서 region growing방법을 사용한다. 즉, 메쉬에 있는 face중에서 임의로 하나를 선택하고 이 face의 이웃한 face들을 region에 포함시키면서 region을 키워나간다. 이 때 평평한 region은 back face culling시에 유리하고, 작은 region은 frustum culling시에 효과적이기 때문에 평평하고 작은 region을 생성하는 것은 필수적이다. 평평하고 작은 region을 생성하기 위하여 region과 가장 가까이 있고, 비슷한 법선벡터을 갖는 face순서대로 region에 포함시켜 가면서 region의 곡률과 크기에 대한 제약조건이 위반되면 region growing을 멈춘다. 여러 크기의 region을 얻기 위하여 leaf region에 곡률과 크기에 대한 강한 제약을 주고 상위 레벨의region으로 갈수록 region에 대한 제약 조건은 완화한다. 그러므로 region tree의 leaf region은 매우 작고 거의 평평하게 되고 상위 레벨의 regions은 더 커지고 더 굽어진다. Region growing시에 region의 크기와 곡률을 추적하기 위해서 그림 2와 같이 minimum bounding sphere와 법선벡터cone방법이 사용되었다.

vertex merge 연산을 region에 적용하여 단순화 시키므로써 여러 LOD 버전들을 생성한다. Vertex merge연산 *vmerge*(vs, vt) 은 source vertex vs 를 target vertex vt 로 결합한다. region의 LOD버전을 만들기 위해서 가능한 최대한 많은 vmerge를 생성한다. 또한 LOD버전을 부드럽게 바꾸기 위해서, 그림 5에 보인 바와 같이 LOD 버전이 A에서 B로 바뀌는 시간동안 merge되는 vertex들의 위치는 source vertex위치와 target vertex위치사이에서 선형보간한다. 그렇지 않으면 LOD버전이 바뀔 때 popping현상이 나타난다. 이 때

같이 merge되는 vertex들이 인접하는 경우 LOD버전이 변화하는 동안 crack이 발생할 수 있다. 이러한 crack을 발생하지 않는 vmerge의 그룹을 CVM으로 명명하고, 다음과 같이 정의한다.

$$CVM = \{vmerge(vs, vt) \mid$$

$$\forall (vmerge_i(v_{s_i}, v_{t_i}), vmerge_j(v_{s_j}, v_{t_j}))$$

$$where \quad i \neq j, star(v_{s_i}) \cap star(v_{s_j}) \neq \phi\}$$

$star(v)$는 vertex에 인접한 faces이다. 다음과 같이 가장 세밀한 버전 R0 에 연속적으로 CVM을 적용하므로써 여러 LOD 버전들을 만든다.

$$R_0 \xrightarrow{CVM_0} R_1 \xrightarrow{CVM_1} \cdots \xrightarrow{CVM_{n-1}} R_n$$

CVM은 아래와 같이 계산한다. *region*의 경계점들을 merge하는 경우, 인접한 다른 region과의 crack이 발생한다. 따라서 Region boundary를 제외한 region내에 존재하는 점들만을 merge시키되 region의 오차를 최소한으로 하는 vertex들을 먼저 merge시켜간다. 3절에서 설명한 region의 LOD버전들이 가질 수 있는 오차의 상한선을 넘지 않는 vmerge를 찾을 수 없으면 LOD버전의 생성이 완료된다.

3절에서 설명한 region의 LOD 버전의 오차 eobj는 region의 모든 점의 오차 중에서 가장 큰 값으로 정한다. 각 점의 오차는 [5]에서 기술되어진 quadric error metric을 사용한다. 이것은 비록 vertex의 오차가 정확하지 않더라도 오차가 conservative하게 측정되고, 빠르면서도 비교적 좋은 결과를 생성할 수 있다.

모든 LOD 버전은 버전번호를 갖는다. 버전번호는 3절에서 설명한 바와 같이, PL을 사용하여 region의 vertex들의 위치를 찾는 데에 필요하다. 만약 region이 leaf region이라면 그때 가장 세밀한 버전의 버전 번호는 0이다. 그리고 만약 부모 region이 몇 개의 자식 region에 의해 생성된다면 부모 *region*의 가장 세밀한 버전의 버전 번호는 자식 region의 버전 번호의 최대치가 된다. 그리고 버전번호는 LOD 버전이 생성될 때마다 1씩 증가시킨다.

5. 실험결과

제안한 알고리듬을 R10000 CPU와 Maximum Impact graphics system을 갖는 SGI Indigo2에서 C와 OpenGL을 사용하여 구현하고 실험하였다. 실험에 사용한 메쉬는 다음과 같다.

1.bunny : 69,451 개의 면과 34,834 개의 점으로 구성되었고, stanford 대학 computer graphics 연구실로부터 제공되었다.

2.grand canyon : height field data로부터 생성한 TIN(triangular irregular networks) 메쉬이다. 69,824 faces와 35,223 vertices를 갖는다. Region tree의 생성은 bunny의 경우 32초, grand canyon의 경우 44초가 소요되었다. Interactive visualization에서는 스크린 공간 오차한계는 1 pixel이고, viewport는 600x600이다.

5.1 렌더링 속도

그림 6은 bunny model을 bunny머리 위에서 내려다 보았을 때의 모습이다(VDPM과의 비교를 위해서 VDPM과 같은 시점으로 설정). VDPM과 비교해 보면 메쉬의 폴리곤 개수는 VDPM의 경우 10,528 개, 제안된 방식의 경우 10,541개로 거의 같음을 알수 있다. VDPM의 경우는 150MHz R4400 CPU를 갖는 SGI Indigo2 Extreme에서 실험하였고, LOD적용전에 1.9 frames per second(fps)이고, LOD적용후 6.7 fps를 얻었다. 이 때 LOD 제어에 든 오버헤드는 20ms정도이다. 제안된 방식의 경우 LOD 적용 전 5fps이고, 제안된 LOD제어방식을 적용 후 17fps로 향상되었다. 이 때 LOD제어 오버헤드는 7ms 로서 VDPM의 오버헤드보다 훨씬 작은 값이다.

특히 VDPM의 경우는 시점의 변화가 빠르게 일어나거나 연속적이지 않은 경우 ecol/vsplit의 개수가 많아지고, 따라서 오버헤드가 크게 증가하여 렌더링 속도가 급격히 떨어질 수 있으나, 제안된 방법은 그러한 현상이 일어나지 않는다.

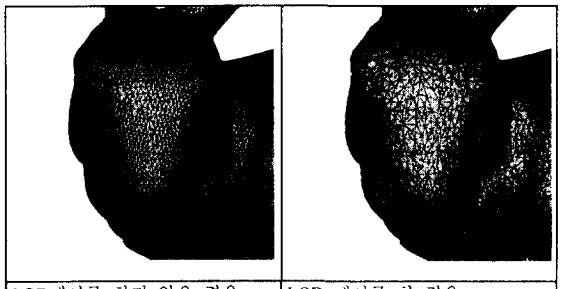
5.2 메모리 사용량

Interactive Visualization 에서의 메모리 사용량은 bunny, grand canyon 의 경우 약 4MB 가 사용되었다. 이것은 VDPM[2]이 bunny model 을 위해 7.6MB 를 사용하는 것에 비교하면 훨씬 적은 양이다.

6. 결론 및 추후과제

본 논문에서는 3 차원 그래픽스에서 사용되는 메쉬를 빠르게 렌더링할 수 있는 LOD 제어방법을 제안하였다. 제안된 방법은 기존 LOD 제어방식의 단점이었던 큰 오버헤드와 메모리의 사용량을 줄이면서 성공적으로 렌더링 속도향상을 얻을 수 있었다.

그리고 앞으로 인간의 시각시스템의 특성을 이용하여 더 많은 view-dependent refinement criteria를 사용하므로써, 이미지의 질은 떨어뜨리지 않으면서 렌더링속도는 높이는 일이 필요하다.



LOD제어를 하지 않은 경우 5 frames per sec	LOD 제어를 한 경우 스크린공간오차 = 1 pixel 17 frames per sec

그림 6. Bunny 에 LOD 를 적용한 예

참고문헌

[1] Julie C. Xia, Amitabh Varshney , " Adaptive real-time level-of-detail based rendering for polugonal models" , IEEE trans.on Visualization and Computer Graphics, 3(2): 171-183, April-June 1997

[2] Hughes Hoppe, " View-dependent Refinement of Progressive Meshes" , Computer Graphics(SIGGRAPH ' 97 Proceedings), 31:189-198,August 1997

[3] Hughes Hoppe, " Progressive Meshes " , Computer Graphics (SIGGRAPH ' 96 Proceedings),: 99-108 August 1996

[4] L. Shirman and S.Abi-Ezzi. " The cone of normals technique for fast processing of curved patches" , Computer Graphics Forum. Proceedings of Eurographics ' 93, pages 261-272, 1993

[5] Michael Garland and Paul S. Heckbert, " Surface simplification using *quadric error metrics* " , Computer Graphics, Proceedings of SIGGRAPH ' 97, pages 209-216, ACM, 1997

[6] Carl Erikson and David Luebke, " View-dependent simplification of arbitrary polygonal environments" , Computer Graphics proceedings of SIGGRAPH 97, 31:199-208, August 1997