효율적인 실시간 렌더링을 위한 삼각형 축약 기법으로 간략화된 점진적 메시

성명건, 박경환
동아대학교 컴퓨터공학과

Triangle Collapse Simplified Progressive Mesh for Effective Realtime Rendering

Myung-gun Sung, Kyung-hwan Park
Dept. of Computer Engineering, Dong-A University
E-mail : personar@donga.ac.kr,
khpark@daunet.donga.ac.kr

요 약

LOD중에서 점진적 메시는 실시간으로 세부 수준을 변경할 경우 높은 수준에서 변을 제거하는 방식으로 세부 수준을 낮추고, 한 정점을 분리하는 방식으로 세부 수준을 높이고 있다. C-LOD는 현재의 거의 모든 3D 엔진에서 지원되고 있으며 현재까지 많은 발전을 거듭해온 점진적 메시는 이제 렌더링 효과에서 벗어나 전체적인 성능의 효율에 초점을 두어 메모리나 CPU비용 등을 줄일 수 있는 방법을 모색하고 있다. 본 논문에서는 전체적인 성능을 높이는 방법으로 메시 간략화의 방법 중 삼각형 축약과 점진적 메시를 동시에 수행하는 방법으로 세부 수준 전환에 드는 비용을 최소화하면서 매우 빠른 간략화를 보여줄 수 있는 효과적인 방법을 제시한다.

1. 서론

점진적 메시(Progressive Mesh, PM)는 3D 메시 간략화(Mesh Simplification) 기술로 원래 모델의 형상과 특징을 그대로 유지해 나가면서 전체 메시의 수를 줄여나감으로써 3D의 세부 수준(Level of Detail)을 변경하는 방법 중의 하나이다. 3D 모델에서 제거하여도 전체 형상을 나타내는데 지장이 없다고 판단되는 메시를 제거해 나가면서 전체 렌더링 처리수를 줄이는 것으로 보통 렌더링 성능과 품질을 모두 만족시키기 위한 수단으로 쓰이고 있으며, 카메라로부터 원거리에 위치하고 있는 모델이라면 낮은 세부 수준으로, 가까이에 위치한다면 높은 세부 수준으로 렌더링 하는 것을 기본적인 방법으로 제시하고 있다. 점진적 메시가 처음 등장했을 때는 많은 관련 전문가들에게 흥미로운 연구 주제로써 뿐만 아니라 새로운 기술로써도 많은 관심의 대상이 되었으며, 많은 발전을 거듭하여 현재는 대부분의 3D API로 개발된 점진적 메시가 다수 존재하고 있다. 3D API 자체에도 포함되는 실정인데, 특히 뷰 독립적인 점진적인 메싱(View-Independent Progressive Meshing)으로의 기본적인 구현은 마이크로소프트의 DirectX8 Direct3D 라이브러리에서부터 채택되었다.

입력으로 들어온 원형 메시에 대해 변 제거(또는 에지 붕괴, Edge Collapse)와 정점 분할(Vertex Split)을 통해서 세부 수준을 변경하는 이 방식은 간단하면서도 매우 강력한 기법으로 널리 알려져 있다. 현재까지도 많은 구현방법들이 제시되어 있고, 이를 좀더 효율적으로 구현하기 위한 여러 방법이 존재하고 있다.

매우 빠른 속도로 진행하며 카메라의 근거리 시점으로부터 멀어지는 다수의 물체를 렌더링 한다고 가정한다면 이에 매우 빈번한 세부 수준 전환이 필요하다. 여기에 기존의 점진적 메시를 통해 간략화를 수행한다면 아주 짧은 시간에 많은 렌더링을 처리해야 한다는 부담이 있다.

이를 해결할 수 있는 방법으로 본 논문에서 제시하는 방법은 기존의 점진적 메시에서의 변 제거 이전에 삼각형 축약 방법으로 메시 간략화를 한 번 더 수행하는 것이다. 이를 통해 매우 빠른 속도로 진행하는 복잡한 물체의 간략화에 드는 비용을 줄임을 기대할 수 있다. 여기에서는 일반적인 점진적 메시에서의 정

점 분할을 통한 세부 수준 증가 부분은 제외시키고 변 제거를 통한 세부 수준 감소를 통한 간략화 부분에만 초점을 둔다.

2. 관련연구

본 절에서는 LOD의 기본 개념과 점진적 메시 그리고 삼각형 축약 기법에 대하여 간단히 살펴본다.

2.1. LOD

3차원 렌더링을 수행할 때 객체들과 캐릭터 등 기하학적 모델들을 화면상에 표현할 때, 같은 객체를 나타내는 다양한 수의 모델을 사용하여 카메라와 물체 사이의 거리에 따라 좀 더 세밀하게 표현하여야 할 때는 상대적으로 많은 수의 다각형과 더 큰 텍스처를 사용한 해상도 높은 모델로, 덜 세밀하게 표현할 때는 적은 수의 다각형과 작은 텍스처를 사용하는 방법이 전통적인 세부 수준(Discrete Level Of Detail)의 기본 개념이다.[8][11]

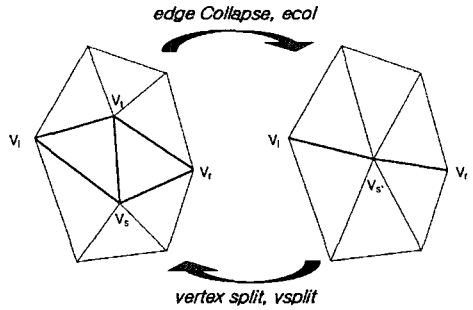
[그림 1] Level of Detail

다양한 세부 수준을 가진 여러 개의 모델들을 만들어 둔 상태에서 카메라와 객체 사이의 거리에 기반을 두어 렌더링을 수행할 모델을 선택한다. [그림 1]을 보면 렌더링을 수행할 세부 수준을 세 가지로 나누고 카메라와의 거리 역시 단계를 나눈 다음 물체와의 거리가 가장 가까운 1단계라고 한다면, 어떤 임의의 문턱값(threshold)을 적용하여 나온 값을 통하여 고수준의 모델 (a)를 사용한다. 또는 어느 정도 떨어져 있는 거리의 2단계라면 중간수준의 모델 (b)를, 그리고 카메라와의 거리가 가장 먼 3단계라면 저수준의 모델(c)를 사용한다.[2] 이는 전통적인 세부수준(Discrete LOD) 방식으로 현재까지 3D 그래픽을 적용하고 있는 많은 곳에서 사용되고 있다.

전통적인 LOD에서의 단점인 카메라가 이동하면서 세부 수준을 변경할 경우, 각 단계 간의 경계 값으로 근처에서의 잦은 단계 모델의 전환으로 발생하는 튀는 현상(Popping Effect)을 막기 위해 확대율의 계산에 의한 화면 크기의 비율에 따른 LOD 선택 방법이나 이력 문턱값 적용을 통한 전환의 빈도를 줄이는 등으로 불안정한 변화의 문제를 해결하고는 있지만 모델이 여러 단계에 따라 모델이 여럿 존재하는 이유로 데이터 저장에 많은 메모리 소요를 한다는 점은 여전히 전통적인 LOD 방식의 단점으로 지적되고 있다. 하지만 전통적 LOD는 그 구현의 간편성으로 인해 많은 분야에서 적용되고 있다.[6][7]

3. 점진적 메시

1996년 Hoppe가 SIGGRAPH 보고서에 처음으로 기술한 점진적 메시는 메시 단순화에 가장 큰 공헌을 한 알고리즘으로 알려져 있다. [그림 2]는 점진적 메시 절차를 나타내는 그림으로 널리 알려져 있다.



[그림 2] 세부 수준 증가와 감소

3.1. 점진적 메시의 개요

좀더 낮은 세부 수준으로의 간략화 부분을 보면 [그림 2]와 같이 변 제거(Edge Collapse), $ecol(V_t, V_s)$를 통해 정점 V_t와 이웃한 V_s중 하나의 정점 V_s로 단일화 시킨다. 이를 통해 임의의 원형 메시 $\hat{M} = M^n$에서 가장 거친(Coarser) 메시 M^0로 간략화된다.

$$(\hat{M} = M^n) \xrightarrow{ecol_{n-1}} \cdots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0$$

반대로, 정점 분할(Vertex Split), $vsplit(s, l, r, t, A)$를 통해 정점 V_s에서 새 정점 V_t를 다시 생성하게 된다. 이로써 M^0에서 임의의 원형 메시 $\hat{M}$으로 세밀화된다.[1]

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \cdots \xrightarrow{vsplit_{n-1}} (M^n = \hat{M})$$

위의 변 제거를 사용하여 반복적으로 변을 제거할 때마다 메시에서 두 개의 삼각형이 사라진다. 스택에 간략화를 수행하고 제거된 정점의 정보를 저장하면서 간략화된 새로운 모델을 가지고 계속 간략화를 수행한다. 이런 식으로 결국 제거할 변이 모두 없어질 때까지 간략화를 계속해 나간다. 이 시점에서 가장 낮은

해상도의 메시지와 제거된 각 변을 나타내는 정보들이 저장되어 있는 스택을 가지게 된다.

점진적 메시가 수행되는 동안 변 제거 집합, 정점 분할 집합 그리고 적용 모델 세 가지의 데이터 영역을 가진다. 정점 분할을 적용하기 위해서는 집합에서 하나를 가져와 메시에 필요한 작업을 수행하고 변 제거를 위해 정점 분할 정보를 구성한 후, 새로 만들어진 정점 분할을 정점 분할 스택에 저장한다.

3.2. 변 선택 함수

점진적 메시에서 변 제거를 수행할 경우 변을 공유하는 제거할 정점 중 어느 것을 선택할 것인지 결정하는 여러 가지 방법 중 보통 대부분이 두 점 중 하나의 정점을 선택하는 방법을 사용하고 있다. 이때 제거한 후 이동할 정점을 선택하는 방법으로 이차원 오차 행렬 알고리즘을 사용한다. Garland와 Heckbert가 1997년에 제시한 이 알고리즘은 대단히 빠를 뿐 아니라 결과도 만족할 만하다.

정점 v와 새로운 정점 v에서 v를 v으로 대체함에 따라 전체 모델에 얼마나 오차가 생길지를 판단하는 것으로, 모든 정점들이 정점을 공유하는 삼각형들이 가지는 평면(planes)의 교차점이라고 한다면, 새 정점이 각 평면에서부터 떨어진 거리로 오차를 정의한다.

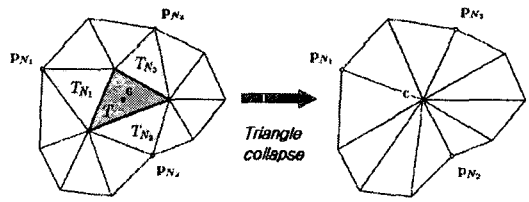
다음의 식을 통해서 오차값을 구한다.

$$\Delta(v) = v^T\left(\sum_{p \in planes(v)} K_p\right)v$$

점진적 메시에서는 이외에도 Melax의 알고리즘[9]이 간단하고 빠른 비용의 함수로 많이 사용되고 있었으나 현재는 대부분 이 이차원 오차 행렬을 사용하고 있으며 여기에서도 이를 사용한다.

4. 삼각형 축약

삼각형 축약(Triangle Collapse)은 점진적 메시 방법과 유사한 메시 간략화 기법이다. 1997년 Gieng에 의해 처음 제시된 삼각형 축약 기법이 점진적 메시와의 차이점은, 점진적 메시는 제거하고자 하는 변을 찾아 이를 통한 메시로부터의 삼각형 제거를 수행하는 것이고, 삼각형 축약 기법은 임의의 삼각형을 선택 후 이 위에 생성시킨 하나의 정점을 통한 삼각형을 제거하는 방식이다. 점진적 메시로 한 번에 축약되는 삼각형의 수가 두 개라면, 삼각형 축약은 중앙으로 선택된 삼각형과 이 삼각형과 변을 이웃하는 나머지 세 삼각형이다. 이를 통해 한번에 축소되는 삼각형의 수는 전부 네 개가 되어서 점진적 메시로 축약되는 삼각형 수의 2배가 된다.

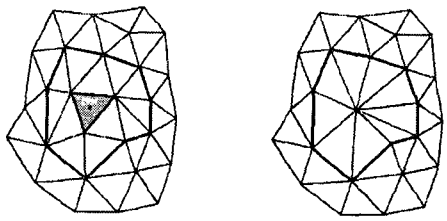


[그림 3] 삼각형 축약 기법

이중 가장 중요한 알고리즘으로 Gieng이 제안한 삼각형 축약 알고리즘의 경우 삼각형 축약 시 [그림 3]과 같이 축약하기로 정해진 삼각형 T와 그 이웃하는 삼각형 T_{N_1}, T_{N_2}, T_{N_3}을 축약할 경우, 중심에 위치한 T 상에 존재하는 새로운 정점 c로 단일화 되는 방법을 제시한다. 이때 삼각형 T의 세 정점의 좌표값이 정점 c의 좌표값과 같아지는 것이다.[4][5]

4.1. 스텐실

삼각형 T를 둘러싸고 있는 삼각형 축약에 필요한 삼각형들의 경계를 잡는 처리를 스텐실(Stencils of Triangle)이라 한다. 스텐실은 [그림 4]에서와 같이 굵은 선으로 표시되어, 이에 속한 삼각형 중 삼각형 T와 변(edge)을 인접한 삼각형들을 없앤다.



[그림 4] Stencils of Triangle

스텐실을 선택하는 방법으로 가장 간단하며 쉬운 방법은 축약될 중심의 삼각형 T의 정점을 같이 공유하는 삼각형들의 공유되지 않는 나머지 두 정점간의 변을 모두 연결하여 만들어진 경계를 선택하는 것이다. 이렇게 정해진 스텐실을 다음의 삼각형 축약 메서드로 축약한다.

4.2. 삼각형 축약 메서드

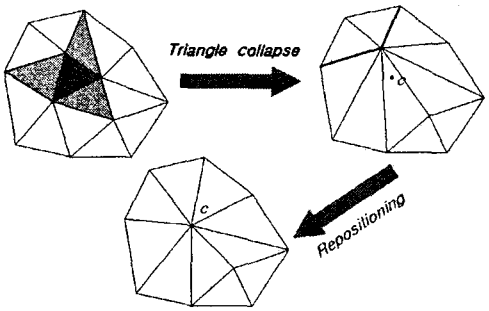
삼각형 축약시 가장 중요한 값은 [그림 3]의 정점 c이다. 이 값을 통해 삼각형 T의 정점들이 정점 c로

합쳐지는 것이다. 정점 c를 정하는 방법은 삼각형 T의 중점을 구하면 된다.

$(\hat{M} = M^n)^{triscol_{(n-1)}} \rightarrow \ldots^{triscol_1} \rightarrow M^1 {}^{triscol_0} \rightarrow M^0$

삼각형 축약 기법 역시 위의 식으로 점진적 메시와 유사하게 표현할 수 있다.

이후 삼각형 T의 정점들을 정점 c로 합하는 방법은 다음과 같이 반복한다.



[그림 5] 삼각형 축약의 전체적인 방법

1) 메시의 각 삼각형들의 무게(Weight) - 변경되어도 전체적인 메시 형태에 지장을 주지 않는 근사값을 무게라고 한다[5] - 를 구하고 이를 우선순위 큐(Priority Queue)에 오름차순으로 저장한다.

2) 가장 무게가 가벼운 삼각형을 T로 정하고, 이 위에 존재하는 정점 c를 구한다.

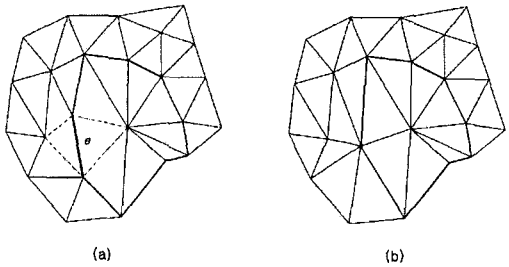
3) 이를 통한 스텐실을 구하고, 정점 c로 간략화하고 새로운 메시를 생성한다.

5. 알고리즘의 개발 및 구현

현재 본 논문에서 제시하는 알고리즘은 아주 빠른 화면전환에 필요한 메시 간략화를 위해 위에서 설명한 삼각형 축약 알고리즘을 통해 한번 간략화를 수행한 다음 점진적 메시의 처리를 통해 삼각형 축약에 의해 간략화된 메시를 한 번 더 간략화시키는 것이다.

점진적 메시를 구현하기 위한 몇 가지 방법이 현재까지 개발되어 있다. 본 논문에서는 표준 점진적 메시(Vanilla PM) 위에 삼각형 축약을 구현하기로 한다. 기존 존재하는 점진적 메시에는 표준 점진적 메시 이외에도 스킵 스트립 점진적 메시(Skip Strips PM) 구현법, 슬라이딩 윈도우 점진적 메시(Sliding Window PM) 구현법, 혼합모드 점진적 메시 구현법 등이 있으나, 이들은 대부분 삼각형 수를 줄이는데 필요한 색인 목록(Index List, Buffer) 으로의 접근 방식의 개선을 통해서 변 제거, 정점 분할에 대한 전체 성능의 효율성을 따지는 것이기 때문에 본 논문에서 제시하는 효율성의 방향과는 다르다고 할 수 있겠다.

알고리즘이 수행되는 순서를 설명하면 다음과 같다. 다음 세부 수준에서 제거될 삼각형들을 선택한 후, 이를 메시로부터 제거하고, 다음으로 제거할 변을 선택, 메시에서 제거하고, 이후 렌더러에서 쓰일 변 제거 데이터를 만들어 내는 과정을 반복한다.



[그림 6] 삼각형 축약 후 점진적 메시 수행

[그림 6]의 (a)는 [그림 4]의 메시를 먼저 삼각형 축약하여 간략화시킨 상태이다. 여기에서 정해진 변 e의 두 정점 중 한 정점을 점진적 메시의 변 선택 함수에 의해 결정된 하나의 정점으로 합하여 삼각형 축약으로 간략화된 메시를 다시 점진적 메시로 간략화시키는 것이 한 번의 수행 절차이다.

점진적 메시를 수행한 후 삼각형 축약기법을 수행한다면, 점진적 메시를 위해 필요한 변 선택 함수와 이후 삼각형 축약을 위한 무게를 구하는 알고리즘 두 번의 처리가 필요하게 된다. 하지만, 삼각형 축약에서의 무게를 계산하고 스텐실을 구하고 나면 그 위에서만 변 선택 함수를 돌리면 되기 때문에 처리양이 훨씬 줄기 때문에 효율이 좋다.

한 번의 수행으로 여섯 개의 삼각형이 줄어들기 때문에 간단히 생각해도 기존의 점진적 메시 방법보다 효율적이라는 것을 알 수 있다.

제안 알고리즘을 통한 구현은 IBM 컴퓨터를 사용하며, Windows NT계열 운영체제와 Visual C++ 6.0을 사용하여 프로그래밍하였다. 3D API로는 Direct3D 9.0 SDK를 사용해 개발한다. 메시 파일로는 x파일을 사용한다. x파일은 3D Max와 같은 3D 응용 애플리케이션에서 만들어진 메시 정보 파일에서 익스포트할 수 있으며, Direct3D 상에서 가장 편리하게 사용할 수 있는 파일이다.

5.2. 제안 알고리즘

제안하는 알고리즘의 전체 수행 순서는 다음과 같다.

// 삼각형 축약 부분(Triangle Collapse part)

1. 전체 메시에서 축약할 삼각형 T를 선택한다. T의 무게를 구해 큐에 저장한 후 그 중 가장 가벼운 무게의 삼각형을 지정, T 정점 정보를 정점 구조체에 저장한다. 정점 정보에는 직교 법선 벡터, 삼각형 자체의 법선 벡터 정보도 저장한다.

2. 정해진 삼각형 T와 인접한 삼각형을 선택한다. 인덱스 버퍼 상에서 T와 정점을 공유하는 세 삼각형의 바깥쪽 변들을 - 삼각형 T와 공유되지 않는 두 정점 간의 변 - 스텐실 정점 버퍼와 변 정보 버퍼에 저장한다.

3. 1에서 저장한 정점 구조체 내의 T의 정점 정보에서 삼각형 T의 중점 c를 구하여 정점 정보로 저장한다.

4. 삼각형 T위의 정점중 하나의 정점으로 나머지 값들을 옮긴다. 이후 합쳐진 하나의 삼각형 위의 정점을 앞에서 구한 정점 c로 옮김으로 재배치를 시킨다.

// 점진적 메시 부분(Progressive Mesh part)

5. 2에서 저장한 스텐실 정점 버퍼와 변 저장 버퍼에 저장된 값 중에서 이차원 오차 행렬을 사용하여 가장 오차값이 적은 변의 정점을 선택한다.

6. 축약할 정점의 정보를 5에서 선택한 정점으로 옮긴다.

7. 변에 대한 비용을 재계산하여 성공하게 되면 전체 알고리즘을 끝내고 새로운 모델을 다시 그린다.

8. 위의 모든 단계를 더 이상 간략화할 수 없는 삼각형의 수가 될 때까지 반복한다.

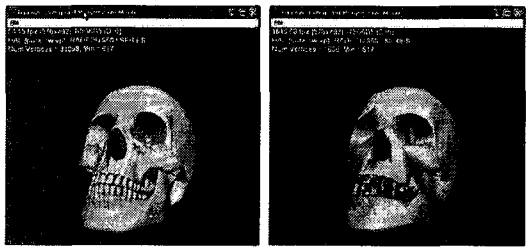
5.3. 구현 및 결과

구현상에 사용한 점진적 메시의 부분은 Svarovsky의 점진적 메시 소스를 사용하여 Direct3D 플랫폼에 맞도록 편집하고, 이와 연결하여 삼각형 축약 부분을 구현한다. 구현 후 메시 파일을 불러서 테스트한 결과는 [표 1]과 같다.

각 축약에서 한번에 여섯 개의 삼각형이 제거되는 것을 확인할 수 있었고, 총 간략화에도 [표 3]에서와 같이 30%이상의 효율을 보였다. 구현 결과 화면은 [그림 7]과 같다.

[표 1] 구현 결과

Models	Triangles	Step of PMTC	Step of PM
Skull	60339	10079	29255
Ship	8686	1469	4264
Teapot	2256	386	1157



[그림 7] 결과 화면

6. 결론

본 논문에서는 효율적인 렌더링을 위하여 점진적 메시 이전에 삼각형 축약 기법을 통해서 간략화를 한 번 더 수행하는 방식을 제안하였다. 점진적 메시만을 통해 한번에 두개의 삼각형을 제거하는 것 보다 전반적으로 효율적이라는 것을 확인할 수 있었다. 현재까지 구현된 것은 축약에 대한 방법만이기 때문에 반대의 경우에 대해서는 언급하지 않았다. 또한 보편적인 경우에 대해서만 구현되었기 때문에 아주 불규칙한 메시에 대해서는 문제를 해결할 수 있는 데 좀더 연구가 따라야 할 것으로 보인다.

[참고문헌]

[1] Hugues Hoppe, Progressive Meshes, Siggraph 1996 Proceedings, pp. 99-108, August 1996.

[2] Hugues Hoppe, View-dependent refinement of Progressive Meshes, Siggraph 1997 Proceedings, pp. 99-108, August 1997.

[3] Hugues Hoppe, Efficient implementation of progressive meshes, Computer & Graphics, vol. 22(1), pp. 27-36, 1998.

[4] Gieng, T.S.; Hamann, B.; Joy, K.I.; Schussman, G.L.; Trotts, I.J.; Constructing Hierarchies for Triangle Meshes, Visualization and Computer Graphics, IEEE Transactions on , Vol 4(2), pp 145-161, 1998.

[5] Gieng, T.S.; Hamann, B.; Joy, K.I.; Schussman,

G.L.; Trotts, I.J.;, Smooth Hierarchical Surface Triangulations, Visualizaion '97., Proceedings, pp379-386, 1997.

[6] Mark Deloura, Game PROGRAMMING Gems, 정보문화사, pp496-503, 2001.

[7] Mark Deloura, Game PROGRAMMING Gems 2, 정보문화사, pp459-476, 2002.

[8] 성명건, 이석희, 황성진, 박경환, 3D 온라인 게임을 위한 지형생성기, 한국멀티미디어학회, pp520-525, 2002.

[9] Melax, Stan, A Simple, Fast, and Effective polygon Reduction Algorithm, Game Developer magazine, pp44-49, 1998

[10] Peter Walsh, Advanced 3D Game Programming using DirectX 8.x, 영진닷컴, pp544-552, 2002.

[11] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery and Werner Stuetzle, Multiresolution Analysis of Arbitrary Meshes, SIGGRAPH '95 Proc., pp. 173-182, Aug. 1995

[12] David H. Eberly, 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics, Morgan Kaufmann Publishers, 2000