한국 CAD/CAM 학회 논문집
제8권 제2호 2003년 6월 pp. 65-74

학술논문

Z-Buffer와 간략화된 모델을 이용한 효율적인 가려지는 물체 제거 기법(Occlusion Culling)에 관한 연구

정성준*, 이규열**, 최항순**, 성우제**, 조두연***

A Study on the Efficient Occlusion Culling Using Z-Buffer and Simplified Model

Jung, S. J.*, Lee, K. Y.**, Choi, H. S.**, Sung, W. J.** and Cho, D. Y.***

ABSTRACT

For virtual reality, virtual manufacturing system, or simulation based design, we need to visualize very large and complex 3D models which are comprising of very large number of polygons. To overcome the limited hardware performance and to attain smooth realtime visualization, there have been many researches about algorithms which reduce the number of polygons to be processed by graphics hardware. One of these algorithms, occlusion culling is a method of rejecting the objects which are not visible because they are occluded by other objects, and then passing only the visible objects to graphics hardware.Existing occlusion culling algorithms have some shortcomings such as the required long preprocessing time, the limitation of occluder shape, or the need for special hardware implementation. In this study, an efficient occlusion culling algorithm is proposed. The proposed algorithm reads and analyzes Z-buffer of graphics hardware using Microsoft DirectX, and then determines each object's visibility. This proposed algorithm can speed up visualization by reading Z-buffer using DirectX which can access hardware directly compared to OpenGL, by reading only the region to which each object is projected instead of reading the whole Z-Buffer, and the proposed algorithm can perform more exact visibility test by using simplified model instead of using bounding box. For evaluation, the proposed algorithm was applied to very large polygonal models. And smooth realtime visualization was attained.

Key words : Occlusion Culling, Visibility, Z-Buffer, Simplified model

1. 서 론

1.1 연구 배경 및 필요성

최근 많은 연구가 진행되고 있는 가상현실 구축, 가상 생산 시스템, simulation based design와 같은 분야에서는, 규모가 크고 복잡하여 매우 많은 수의 다각형(Polygon)으로 이루어진 3차원 모델을 실시간으로 가시화해야 한다. '실시간 가시화'를 구현하려면 FPS (Frames Per Second: 1초 동안 화면이 다시 그려지는 횟수)가 일정 정도 이상이 되어야 하는데, 영화가 24 FPS이고 TV가 30 FPS이므로 이 정도 FPS이면 매우 부드러운 화면을 볼 수 있고, 60 FPS 정도가 사람 눈이 차이를 느낄 수 있는 최고치라고 한다. 그리고 KRISO의 선박운항 시뮬레이터는 10 FPS 정도로 작동하는데 이 정도만 되어도 어느 정도 부드러운 화면을 볼 수 있다. 그러나 모든 그래픽 하드웨어는 단위 시간 동안 처리할 수 있는 다각형의 수에 한계가 있으므로, 다각형이 많아지면 속도가 저하되어 실시간 가시화가 불가능한 문제점이 발생한다.

이러한 문제를 해결하기 위해 Visibility Culling, LOD(Level of Detail), Texturing 등 그래픽 하드웨어가 처리해야 할 다각형의 수를 줄여주는 기법이 많이 연구되어 왔다. Visibility Culling은 실제로 화면에 보이지 않을 물체를 미리 제거하여 그래픽 하드웨어가 처리해야할 다각형의 수를 줄여주는 기법이고, LOD는 카메라에서의 거리에 따라서 여러 단계의 간략화된 모델을 가시화 하도록 하는 기법이다. Texturing은

*쌍용정보통신 KCTC프로젝트팀
**서울대학교 조선해양공학과 및 해양시스템공학연구소
***서울대학교 조선해양공학과 대학원
- 논문투고일: 2002. 02. 18
- 심사완료일: 2003. 02. 03

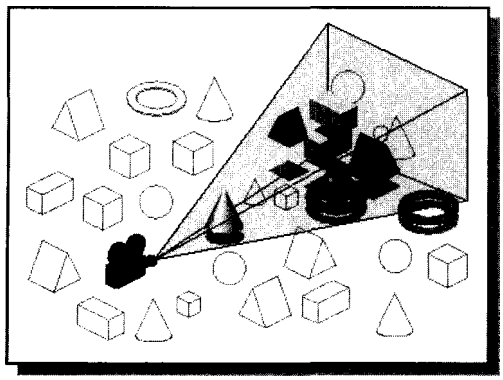


Fig. 1. Concept of View Frustum Culling and Occlusion Culling.

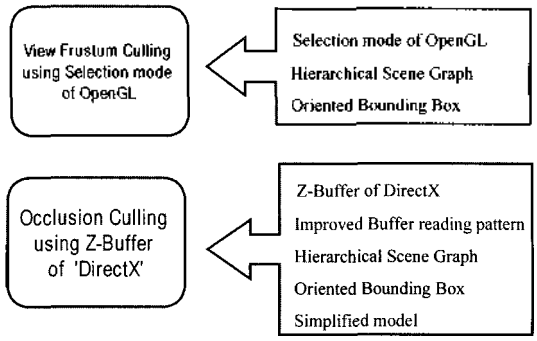


Fig. 2. Outline of the proposed View Frustum & Occlusion Culling.

건물의 창문이나 문의 질감을 표현하기 위해 매우 많은 수의 다각형을 이용하는 대신 2차원 이미지를 물체의 표면에 매핑시켜서 매우 적은 수의 다각형으로 현실감 높은 이미지를 얻어내는 기법이다. 그 중 visibility culling에는 View Frustum Culling과 Occlusion Culling이 있으며 많은 연구가 수행되어 왔다[1-3].

Visibility Culling 기법들 중의 하나인 View Frustum Culling은 Fig. 1에서 카메라의 시야에 들어오지 않는 물체들을 미리 제거해 버리고 시야에 들어오는 물체들만 그래픽 하드웨어가 처리하도록 하는 방법이다. 또 Occlusion Culling은 Fig. 1에서 카메라의 시야에 들어오는 물체들 중에서도 앞의 물체에 가려져서 보이지 않는 물체들은 미리 제거해 버리고 실제로 보이는 물체만 그래픽 하드웨어로 보내어 가시화 함으로써 그래픽 하드웨어가 처리해야할 다각형의 수를 줄여주는 기법이다.

1.2 연구 범위 및 목적

본 연구에서는 Z-Buffer와 간략화 된 모델을 이용한 Occlusion Culling을 연구하고 구현하였다. 이 Occlusion Culling 기법은 먼저 Microsoft에서 제공하는 그래픽 라이브러리인 DirectX를 이용하여, 각 물체의 Bounding Box를 이루는 삼각형들(직육면체의 한 면당 2개씩 총 12개의 삼각형)을 그래픽 하드웨어로 보내어 Bounding Box가 Z-Buffer에 그려지도록 한 후 Z-Buffer의 내용이 변했는지를 검사하여 물체의 가시화 여부를 결정한다. Bounding Box를 이용하는 이유는 Occlusion Culling을 위한 계산량을 줄이기 위함이다. 그리고 가시화 해야 할 물체들만 그래픽 하드웨어로 보내어 가시화 하도록 한다. 이를 위하여 Scene을 이루는 물체들을 계층적으로 트리 구조에 저장하고, 각 물체의 Bounding Box를 자동적으로 생성하도록 하였다.

본 연구에서는 DirectX를 이용하여 Z-Buffer를 읽고 검사하도록 함으로써 OpenGL을 이용한 기존의 연구보다 빠른 Occlusion Culling을 수행할 수 있었고, 각 물체의 Bounding Box를 뷰평면에 투영하고 그 물체로 인해 Z-Buffer의 내용이 변할 수 있는 x, y 좌표 범위를 구하여 그 투영된 영역만 읽고 검사함으로써 많은 검사 속도 향상을 얻을 수 있었다.

또한 물체의 Bounding Box 대신 물체의 형상을 유지하면서 삼각형의 개수를 줄인 간략화 된 모델을 검사에 이용하여 훨씬 더 정확한 가시화 여부 검사를 수행할 수 있었다.

구현된 Occlusion Culling 기법을 많은 수의 다각형으로 이루어진 모델들에 적용하여 테스트 해봄으로써 적용 가능성과 그 효율성을 보였다.

1.3 관련 연구 현황

지금까지 여러 가지 Occlusion Culling 방법들이 제안되었고 [3]과 [4]에서 매우 자세한 관련연구 현황 조사를 볼 수 있다. 그 중에는 많은 수의 각 시점에서 보이는 물체들을 미리 계산해서 리스트로 저장해 놓고 그 물체만을 가시화 하는 PVS(Potential Visible Set) 기법[5], 계층적인 Z-Buffer를 만들어 한 물체의 Bounding Box를 그릴 때 이 버퍼의 내용이 변하는지 검사하여 물체의 가시화 여부를 검사하는 기법[1], 그리고 각 물체의 그림자를 2차원 상에 투영하여 검사하는 기법[6,7] 등이 있다. 그러나 기존의 occlusion culling 기법은 많은 양의 preprocessing이 필요하거나, 건축물 내부 같은 특정 모델에만 적용 가능한 경우, occluder(다른 물체를 가리는 물체)의 형상에 제한

이 있는 경우, 특별한 하드웨어가 필요한 경우 등의 단점이 있었다. 최근에 이러한 문제점들을 해결하기 위해서 'OpenGL'의 Z-buffer를 이용한 기법이 연구되었지만 그래픽 하드웨어의 정보를 읽어오는데 많은 시간이 걸리는 단점이 있었다[7].

2. 그래픽 파이프 라인

3차원 물체를 2차원 화면에 가시화 하기 위한 그래픽 파이프라인은 간단히 설명해서 Fig. 3과 같이 구성되어 있다[8,9]. 먼저, 물체를 이루는 다각형(삼각형)들의 모든 꼭지점들을 3차원 실세계 좌표계에서 화면에 그려질 2차원 좌표계로 변환하는 것을 "Transform"이라고 부른다. 그리고 각 꼭지점의 색을 normal vector와 조명을 이용해서 계산하는 "Lighting" 단계를 거친 다음, 화면에 그려질 영역을 벗어나는 것들을 제거하는 "Clipping" 단계를 거친다. 마지막으로 2차원으로 변환된 삼각형의 꼭지점을 이용해 삼각형의 내부를 채워 넣는 단계인 "Rasterization"을 거치면 화면에 그림이 그려지게 된다.

2.1 Z-Buffering

Z-Buffering은 그래픽 파이프라인에서 여러개의 물체들이 겹쳐져 있을 때 서로에게 가려서 화면에는 보이지 않을 부분과 화면에 그려져야 할 부분을 가려내는데 사용하는 방법이다.

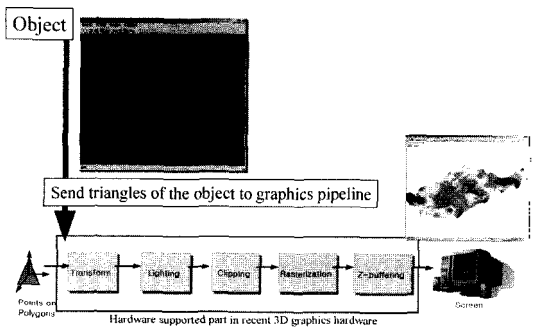


Fig. 3. 3D Graphics Pipeline.

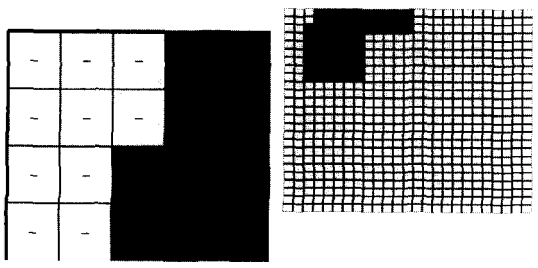


Fig. 4. Example of Z-Buffering.

Fig. 4에서처럼 두 개의 사각형을 화면에 그린다면 위 쪽의 긴 사각형의 내부는 5, 아래 쪽의 정사각형의 내부는 10이라는 각 점의 깊이를 저장해 둔 것이 Z-Buffer이다.

Z-Buffering은 화면에 각 점을 그릴 때마다 현재 Z-Buffer에 저장되어 있는 z 값과 지금 그릴 점의 z 값을 비교한다. Z-Buffer에 저장되어 있는 값이 더 크다면 깊이가 더 깊은 것이므로 새로 그릴 점이 이미 그려져 있는 점보다 위에 있는 것이다. 그러면 새로운 점을 덮어씌우고 Z-Buffer에는 새로운 z 값이 저장되게 된다.

이렇게 Z-Buffering은 각 다각형을 그릴 때마다, 각 점에 대해서 검사를 수행하는 기법이다. 최근에는 대부분의 그래픽 카드에서 기본으로 채택하고 있는 방법이며 하드웨어적으로 상당히 빠르게 수행된다.

3. Occlusion Culling

Occlusion Culling은 복잡한 연산을 많이 해야하는 그래픽 파이프라인의 작업을 줄이려는 방법이다.

Occlusion Culling을 하지 않을 때는, 전체 Scene에 있는 모든 물체를 이루는 모든 다각형이 Fig. 3의 그래픽 파이프라인의 Clipping 단계에서 시야에 들어오지 않는 다각형이 제거되지만, Occlusion Culling을 수행하면 각 물체의 Bounding Box만을 가지고 물체가 실제 화면에 그려지는지 검사하고 나서 화면에 그려지지 않는 물체들은 Fig. 3의 "Transform"부터 이후의 모든 과정을 거치지 않으므로 속도의 향상을 얻게 된다.

3.1 Scene Graph

전체 Scene을 구성하는 물체들은 계층적 구조를 갖

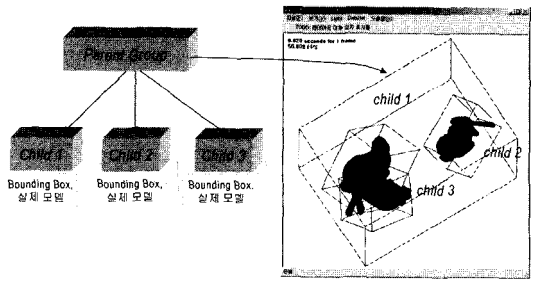


Fig. 5. Hierarchical Scene Graph.

고 각 그룹마다 Bounding Box를 가진다. 예를 들어 Fig. 5와 같이 Scene Graph가 구성되어 있을 때, 토끼 세 마리 전체를 감싸는 Parent Group의 Bounding Box가 시야에 전혀 들어오지 않는다면 그 Child Group들(child 1, 2, 3)은 검사해볼 필요도 없이 가시화 하지 않으면 된다. 그리고 Scene 전체의 Bounding Box가 시야에 조금이라도 들어온다면 그 Child Group의 Bounding Box들을 각각 검사해서 가시화 여부를 결정하게 된다.

이 Scene Graph를 이용하는 이점은 Occlusion Culling 자체가 상당히 많은 처리 시간을 필요로 하는 작업인데, Parent Group을 검사한 후 그 Child Group들을 검사하지 않을 수 있다면 상당한 성능 향상을 기대할 수 있다는 것이다.

3.2 Oriented Bounding Box(OBB)의 생성

각 물체의 Bounding Box는 가장 단순하게 x, y, z 좌표의 최대 최소값을 구해서 좌표축에 평행한 직육면체로 만들 수도 있다(Axis Aligned Bounding Box; AABB). 하지만 Fig. 6의 (a)에서처럼 AABB는 빈공간이 많이 생기기 쉽다. 따라서 Fig. 6(b)에서처럼 보다 물체에 잘 맞는 Bounding Box를 만들기 위해서 본 연구에서는 그 물체를 구성하는 점들의 분포를 조사하여 가장 공분산(covariance)이 큰 축을 기준으로 직육면체를 만든다(Oriented Bounding Box; OBB)[10,11]. OBB를 만드는 과정은 먼저 x, y, z 좌표의 공분산을 계산하여 Bounding Box의 축을 잡고 그 축을 기준으로 하는 새로운 좌표계로 모든 점들을 변환 시킨다. 그리고 그 좌표계에서 x, y, z 좌표의 최대, 최소를 구해서 Bounding Box의 경계를 구한 뒤, 그 경계들을 다시 원래의 좌표계로 역변환 하면 OBB를 얻을 수 있다.

Bounding Box를 만드는 시간이 오래 걸리기 때문에 본 연구에서는 scene data를 읽어올 때 Bounding Box를 미리 만들어 두고 계속 사용한다.

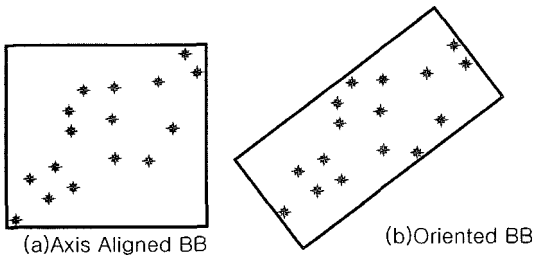


Fig. 6. OBB is more fit to model than AABB.

4. Z-Buffer를 이용한 Occlusion Culling

4.1 OpenGL의 Z-Buffer를 이용한 기존의 Occlusion Culling

최근에 OpenGL의 Z-Buffering을 이용해서 각 물체의 Bounding Box가 가려지는지의 여부를 검사하여 Occlusion Culling을 구현한 기법이 연구되었다[2].

이 기법에서는, Z-Buffer를 직접 읽어오는 것보다 더 빨리 읽어올 수 있는 Stencil Buffer를 이용하여 'Virtual Occlusion Buffer'를 만든다. 그리고 Fig. 7처럼 각 물체의 가시화여부를 판단하기 위해서 각 물체의 Bounding Box를 'Virtual Occlusion Buffer'에 그려 넣는다. 그러면 Z-Buffering에 의해서 그 물체의 Bounding Box가 앞의 물체에 가려지지 않을 때에만 'Virtual Occlusion Buffer'에 그려진다. 그 후 이 버퍼를 읽어서 버퍼의 내용이 변했는지를 검사한다. 버퍼의 내용이 변했으면 그 물체는 가시화 해야하는 것이고, 버퍼의 내용이 전혀 변하지 않았으면 그 물체는 앞의 물체에 가려서 화면에 보이지 않을 물체라고 판

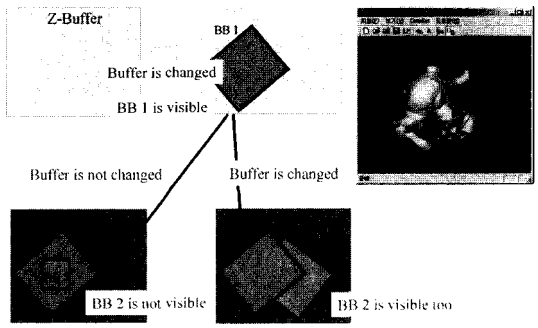


Fig. 7. Visibility test by checking if occlusion buffer is changed.

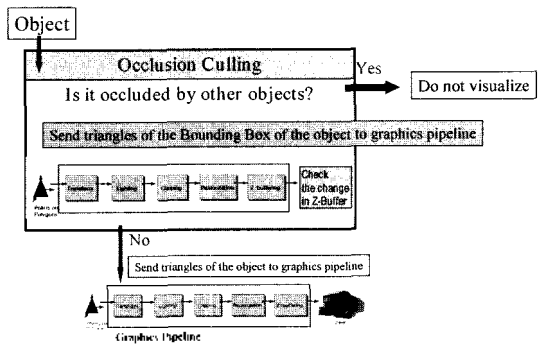


Fig. 8. Procedure of Occlusion Culling using Z-buffer.

단할 수 있다.

각 물체의 Bounding Box를 Occlusion buffer에 그려 넣는 작업은 그래픽 카드에서 수행된다. Fig. 8과 같이 Occlusion Culling을 수행하기 위해서 각 물체의 Bounding Box를 이루는 삼각형들이 그래픽 카드로 보내지고, Occlusion Buffer(Z-buffer)에 Box들이 그려진다. 그러면 이 버퍼의 내용을 읽어서 Occlusion Culling을 수행하고 가시화 해야할 물체를 이루는 삼각형들만 다시 그래픽 카드로 보내져서 화면에 가시화 된다.

그런데 이러한 Occlusion Culling 기법에서는 각 물체의 가시화 여부를 검사할 때마다 버퍼 전체를 읽어서 내용이 바뀌었는지를 검사해야하기 때문에 버퍼를 읽어오고 이전의 내용과 비교하는 작업에 상당히 많은 시간이 걸리게 된다.

따라서 OpenGL을 이용한 Occlusion Culling에서는 버퍼의 전체 내용을 모두 읽지 않고 일부분만 읽어서 비교하는 'span reading'을 구현하였다. 'span reading'을 통해서 상당한 속도 향상을 거둘 수 있었지만 아직 '실시간(real time)'이라 하기엔 부족한 결과를 얻었다.

4.2 Z-Buffer와 간략화 된 모델을 이용한 제안된 Occlusion Culling

Z-Buffer를 이용한 Occlusion Culling에서 버퍼를 읽어오는데 걸리는 시간이 가장 중요한 문제였으므로 본 연구에서는 하드웨어를 보다 더 직접 제어할 수 있는 DirectX를 이용하여 Z-Buffer를 읽고 분석하는 Occlusion Culling을 구현하였다. Microsoft에서 제공하는 그래픽 라이브러리인 DirectX는 이름에서 말하듯이 OpenGL보다 그래픽 하드웨어에 보다 더 직접적으로 접근할 수 있도록 해준다. 즉 버퍼를 읽어 올 때 OpenGL보다 더 빠른 속도를 낼 수 있는 그래픽 라이브러리이다.

OpenGL을 이용한 기법보다 DirectX를 이용하였을 때의 더 좋은 점은, 먼저 하드웨어에 보다 더 직접 접근하므로 버퍼를 읽어오는 속도가 더 빠르다는 점이다. 그리고 Stencil Buffer를 이용하지 않고 바로 Z-Buffer를 읽어오므로 Stencil Buffer에 쓰고 읽는 시간을 절약할 수 있다. 그리고 'span reading'을 할 때 버퍼의 내용을 읽어오는 패턴을 더 효율적으로 만들 수 있다.

본 연구에서는 또한 각 물체의 Bounding Box를 그래픽 카드가 Z-buffer에 그리도록 한 뒤 그 Bounding Box가 2차원 평면상에 투영된 x, y 좌표

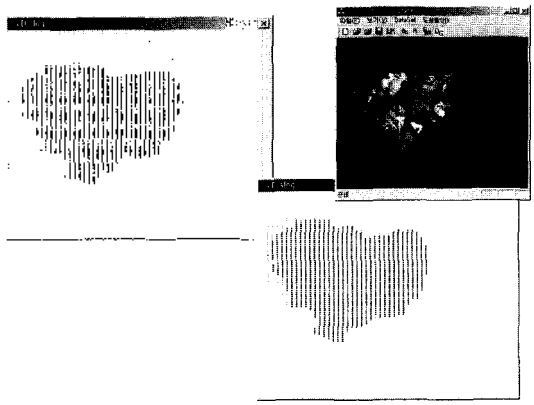


Fig. 9. Reading only the projected region of each bounding box.

범위를 계산하여 그 범위의 Z-buffer만을 읽고 검사하도록 하였다. 각 물체에 대하여 Z-buffer의 내용이 바뀔 수 있는 범위만 읽고 검사하도록 함으로써 많은 속도 향상을 얻을 수 있었다. Fig. 9에서 왼쪽 그림은 Z-buffer 전체를 읽어온 것을 알 수 있지만, 오른쪽 그림에서는 각 물체의 Bounding Box가 투영된 영역만을 읽고 검사한 것을 알 수 있다. 또한 Bounding Box 대신 간략화 된 모델을 가시화 여부 검사에 이용하여 훨씬 더 정확한 가시화 여부 검사를 수행할 수 있었다.

4.3 Z-Buffer를 읽어들이는 Pattern

Z-Buffer를 읽어오는 작업이 많은 시간을 요구하기 때문에 본 연구에서는 버퍼의 내용 중 일부만 읽어오도록 했다. 그런데 OpenGL에서는 버퍼에서 읽어오는 횟수가 많아지면 속도가 현저히 느려지므로 Fig. 10(a)처럼 가능한 한 한번에 많은 수의 점들을 읽어와야 했다. 따라서 Fig. 10(a)와 같은 패턴이 만들어진다. 하지만 이 그림에서 진한 점들만 읽어오고 검사하므로 빈 공간에 그려지는 물체는 가시화 여부를 제대로 검사할 수 없다.

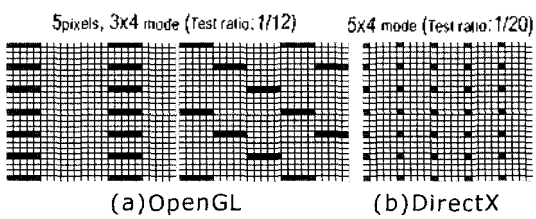


Fig. 10. Buffer reading pattern.

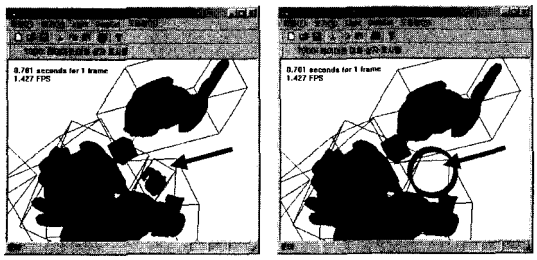


Fig. 11. Object culled inaccurately by using Bounding Box.

DirectX에서는 Fig. 10(b)처럼 한 번에 점을 하나씩 읽어와도 같은 비율의 점을 읽어왔을 때 OpenGL 보다 속도가 더 빨랐다. Fig. 10(b)와 같은 경우 검사할 수 있는 점들의 그물이 더 촘촘하게 되어서 검사의 성능을 향상시킬 수 있다.

Fig. 10(b)에서 '5×4 mode'라는 것은 가로로 점을 4개씩 건너뛰고 세로로 3개씩 건너뛰면서 검사한다는 의미이다. '5×4 mode'에서는 20개의 점 중에서 하나의 점을 읽고 검사하게 된다. Fig. 10(a)의 '3×4 mode'란 5개의 점을 읽고 가로로 10개를 건너뛰고 세로로는 3줄을 건너뛴다는 뜻이다.

4.4 간략화 된 모델의 이용

Occlusion Culling에 Bounding Box를 이용하는 이유는 Culling을 위한 계산의 양을 줄여주기 위함이다. 만약 실제 모델의 모든 삼각형을 이용해서 검사를 한다면 Culling을 하지 않고 그냥 그래픽 카드가 모두 처리하도록 하는 것이 훨씬 더 빠른 결과를 얻을 수 있을 것이다. 그러나 Bounding Box를 이용한 가시화 여부 검사는 Oriented Bounding Box를 이용하더라도 Fig. 11과 같이 실제 물체에는 가려지지 않지만 Bounding Box에는 가려져서 보이지 않는 것으로 잘못 제거되는 물체가 생기기 쉽다.

더 정확한 검사를 하기 위해서는 하나의 물체를 여러개의 Bounding Box로 쪼개서 더 많은 검사를 수행해야 한다. 하지만 더 많은 검사를 수행하려면 Occlusion Culling 자체에 필요한 시간이 더 증가하는 문제점이 있다.

본 연구에서는 여러개의 Bounding Box를 이용하지 않고 물체를 표현하는데 간략화된 모델을 이용함으로써 물체 하나당 한번의 검사로 매우 정확한 가시화 검사를 수행할 수 있었다. 여러 가지 간략화 기법에 대한 설명은 [12]에서 찾아볼 수 있는데 이 논문에서는 원본 물체의 특징적인 형상을 잘 유지시켜 주면서 간략화 된 모델의 삼각형의 개수를 조절할 수 있는

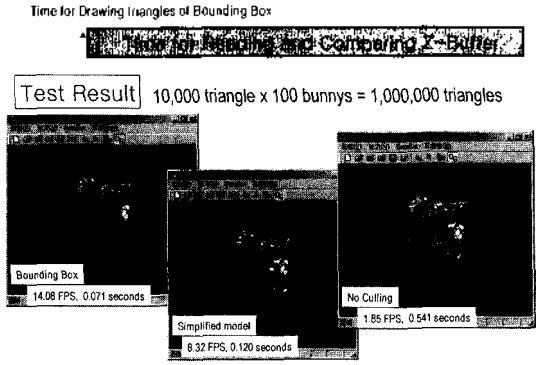


Fig. 12. Test result of using simplified model.

QSlim[18]이라는 공개된 프로그램을 이용하였다.

본 연구에서 구현된 Occulsion Culling 기법은 먼저 각 물체의 Bounding Box를 Z-buffer에 그린 후 그 내용을 읽고 검사하는데, Bounding Box를 그리는 작업은 그래픽 카드에서 매우 빠르게 이뤄지므로 Fig. 12의 그림처럼 전체 수행시간 중에 매우 작은 부분만을 차지한다. 따라서 간략화 된 모델을 사용하여 그려야할 삼각형의 수가 늘어나더라도 검사에 걸리는 시간은 크게 늘어나지 않았다. Fig. 12에서 Bounding Box를 이용한 방법과 간략화 된 모델을 이용한 방법 간에 어느 정도의 속도 차이가 나는데 이것은 검사 자체에 걸린 시간 보다는 간략화 된 모델을 이용하면 더 정확한 검사를 수행하여서 Culling으로 제거되는 물체가 줄어들기 때문에 나온 결과이다. Occlusion Culling 자체에만 걸린 시간을 측정하기 위해서 Culling으로 제거되는 물체도 모두 가시화 하도록 하고 테스트 했을 때는 속도의 차이가 측정되지 않을 정도로 작았다.

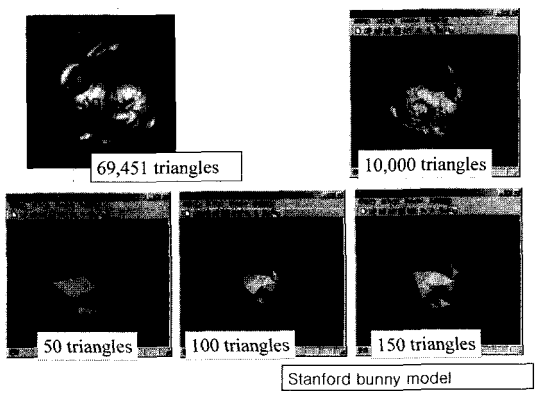


Fig. 13. Simplified models of different number of triangles.

Fig. 13에 삼각형의 개수가 다른 여러 가지 간략화 된 모델을 보였는데 본 연구에서는 테스트 결과 적절한 검사의 정확성을 보여준 100개의 삼각형으로 이루어진 간략화 된 모델을 이용했다.

5. 제안된 Occlusion Culling의 구현 예

본 연구에서 구현된 Occlusion Culling의 효율성을 테스트하기 위해서 Fig. 14와 같은 시스템을 구성하였고, 구현을 위하여 [13~15]를 참조하였다. DirectX를 이용하기 위해서는 EzGraph사의 그래픽 라이브러리인 EzGL을 이용하였다[16]. 먼저 OpenGL의 Selection mode를 이용한 View Frustum Culling[12]을 구현하고 테스트 해보았다. Fig. 15에서 보이듯이 테스트 결과 시야에 들어오는 물체의 수가 적을 때는 30 FPS 이

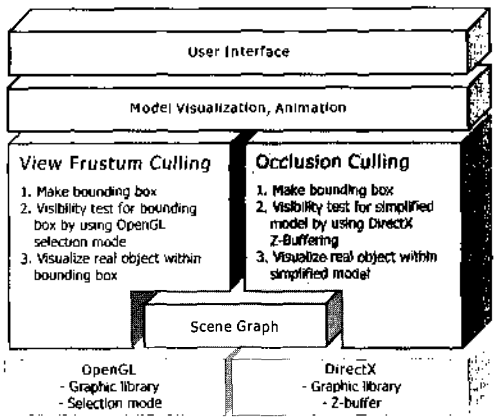


Fig. 14. Configuration of the proposed Visualization Test System.

No.	No. of Triangles in one model / No. of models	without Culling	with Culling	
			Only 1 bunny is visible	All bunnys are visible
1	5000 / 100	2.3 fps	More than 30fps	2~3 fps
2	30,000 / 50	0.7 fps	More than 30 fps	0.7 fps
3	200 / 1000	10 fps	20 fps	10 fps

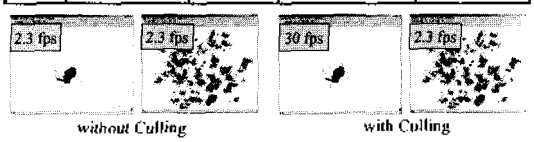
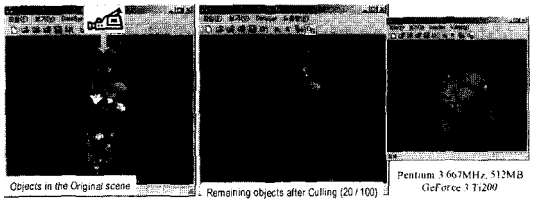


Fig. 15. Test result of View Frustum Culling using OpenGL's Selection mode.



	Reading Projected Region	Reading Entire Z-Buffer
No Culling	1.85 FPS, 0.541 seconds	1.85 FPS, 0.541 seconds
View Frustum Culling only	1.78 FPS, 0.561 seconds	1.78 FPS, 0.561 seconds
View Frustum & Occlusion Culling	(avg) 14.08 FPS, 0.071 seconds (best) 8.32 FPS, 0.120 seconds	2.02 FPS, 0.496 seconds

10,000 triangles per bunny(1 object) * 100 objects = 1,000,000 triangles.

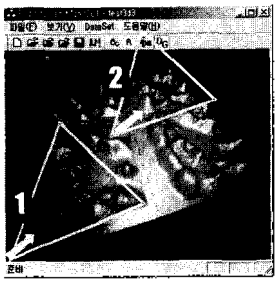
Fig. 16. Test result of General model using the proposed View Frustum & Occlusion Culling.

상의 빠른 결과를 얻었다.

Occlusion Culling의 효율성을 확인하기 위해서 Fig. 16과 같이 10,000개의 삼각형으로 이루어진 물체 100개로 이루어진 scene을 이용했다. 왼쪽부터 카메라의 위치와 scene 전체의 모습, Occlusion Culling으로 제거되고 남은 물체들, 실제 화면에 보이는 이미지를 나타내고 있다. 테스트는 GeForce3 Ti200 그래픽 카드를 장착한 펜티엄3 667 MHz에서 수행되었다. Culling을 하지 않거나 View Frustum Culling만을 수행하였을 때는 2 FPS가 안되는 속도를 보였으나 Occlusion Culling을 수행하였을 때는 Bounding Box를 이용했을 때는 평균 14.08 FPS, 간략화 된 모델을 이용하였을 때는 8.32 FPS의 결과를 얻었다. 각 물체를 검사할 때마다 버퍼 전체를 읽어오도록 했을 때는 Occulsion Culling을 수행하여도 2 FPS 정도의 속도 밖에 나오지 않았지만 각 Bounding Box가 투영된 영역만을 읽도록 하여서 많은 속도의 향상을 얻을 수 있었다.

또 하나의 예제로 약 200만개의 삼각형으로 이루어진 해저지형 모델에 적용하였다. 전체 모델을 256개의 블록으로 분할했고 각 블록은 Bounding Box와 삼각형 100여개로 간략화 한 모델을 가지고 있다. View Frustum Culling 및 Occlusion Culling은 이 블록 단위로 적용된다.

Fig. 17처럼 이 해저지형 위를 카메라가 임의로 돌아다닐 때, 해저지형 전체가 보일 때는 View Frustum Culling 및 Occlusion Culling에 걸리는 시간이 더 필요한데 Culling으로 제거되는 블록은 없으므로 Culling을 전혀 하지 않았을 때보다 더 느린 속도를 보여주었다. 하지만 256개의 블록에 대해서 검사를 수



No Culling	0.825 FPS. 1.212 seconds
All blocks are visible	0.811 FPS. 1.232 seconds
View 1	10.0 FPS. 0.10 seconds
View 2	3.846 FPS. 0.26 seconds

2,134,016 Triangles

Fig. 17. Test scene of Seabed model using the proposed view Frustum and Occlusion Culling.

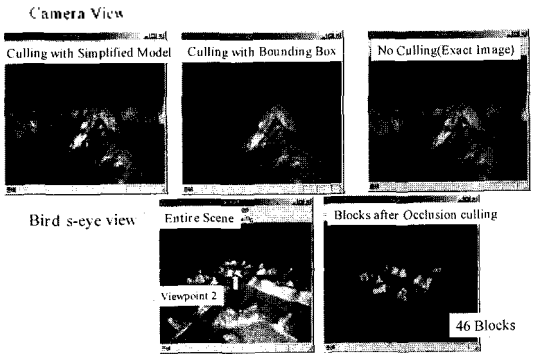


Fig. 18. Occlusion Culling using Simplified Model and Bounding Box.

행하는데 0.02초 정도만 더 소요되었으므로 검사가 매우 빠르게 수행됨을 확인할 수 있었다.

Fig. 17의 시점 1과 같이 카메라가 화살표 방향을 바라보고 있을 때는 10.0 FPS(Frame Per Second), 시점 2의 경우에는 3.8 FPS까지 속도 향상을 확인할 수 있었다. 이 때는 시야범위(삼각형 안) 밖의 블록들은 View Frustum Culling으로 이미 제거 되었고 시야범위 안의 블록 중에서도 일부는 Occlusion Culling으로 제거 되었다. 따라서 256개의 블록 중에서 시점 1에서는 9개, 시점 2에서는 Fig. 18에서 보이는 46개의 블록만이 그래픽 카드에서 처리되었기 때문에 View Frustum Culling 및 Occlusion Culling을 하지 않았을 때보다 더 빠른 속도를 낼 수 있었다.

Fig. 18에서 Bounding Box를 이용한 Occlusion Culling은 실제로는 보이지만 Bounding Box에는 가려지는 물체를 잘못 제거해서 컬링을 하지 않은 정확한 이미지와 많이 다른 결과를 나타냈지만, 간략화 된 모델을 이용한 Occlusion Culling은 거의 정확한 이미지를 만들어 냈다.

OpenGL의 Z-Buffer를 이용한 방법과의 비교를 위

Fig. 19. Test Scene of Bunny model to compare the Occlusion Culling by using OpenGL and DirectX.

Table 1. Comparison of the Occlusion Culling by using DirectX and OpenGL

물체개수	DirectX	OpenGL
9개	21.7 FPS	7.8 FPS
100개	2.32 FPS	0.71 FPS
1000개	0.26 FPS	0.07 FPS

해서 Fig. 19와 같이 삼각형 1000개로 이루어진 모델을 여러 개 복사해서 Scene을 구성하였다. 임의로 시점을 변경하면서 평균 속도를 측정하여 Table 1과 같은 결과를 얻었다.

테스트 결과 본 연구에서 구현한 DirectX의 Z-Buffer를 이용한 Occlusion Culling이 OpenGL의 Z-Buffer를 이용한 방법보다 3배 이상 빠른 결과를 보였다.

6. 결론 및 향후 연구 계획

본 연구에서는 가상현실 환경이나 simulation based design등에서 매우 많은 수의 삼각형으로 이루어진 모델을 실시간으로 가시화 하기 위한 기법 중 하나인 Occlusion Culling을 DirectX의 Z-Buffer와 간략화 된 모델을 이용해 구현하였다. 이 기법은 일반적인 3차원 가속 그래픽 카드에서 구현 가능하고, 적용 가능한 모델에 제한이 없다. 또한 DirectX가 OpenGL 보다 하드웨어에 더 직접적으로 접근할 수 있기 때문에 OpenGL을 이용한 방법보다 훨씬 더 빠르고, 더 촘촘한 패턴으로 Z-buffer를 읽고 검사함으로써 더 정확한 검사를 수행할 수 있다. 또한 Z-buffer의 전체 내용을 검사하지 않고 각 물체의 Bounding Box가 투영된 영역만 읽고 검사하도록 함으로써 많은 속도 향상을 얻

을 수 있다. 그리고 간략화 된 모델을 이용함으로써 Bounding Box와 검사의 횟수를 증가시키지 않고도 매우 정확한 Occlusion Culling을 수행할 수 있다.

본 연구에서 구현한 Z-Buffer와 간략화 된 모델을 이용한 Occlusion Culling은 매우 복잡하고 큰 모델의 가시화 성능에 도움을 줄 수 있음을 테스트를 통해 확인 하였다.

향후 연구 계획으로는 AUV 시뮬레이터의 해저 지형 가시화와 선박 내부 모델 가시화에 Occlusion Culling을 적용할 계획이다.

감사의 글

이 연구는 과학기술부가 지원하는 국가지정연구실인 서울대학교 공과대학 해양공학 연구실에서 수행하는 "수중체의 자율 지능제어 연구"의 지원과 정보통신부의 대학기초과제의 지원으로 수행되었습니다.

참고문헌

1. Greene, N., Kass, M. and Miller, G., "Hierarchical *Z*-Buffer Visibility," *Proceedings of ACM SIGGRAPH*, pp. 231-238, 1993.
2. Bartz, D., Meisner, M. and Huttner, T., "OpenGL-assisted Occlusion Culling for Large Polygonal Models," *Computers & Graphics*, Vol. 23, pp. 667-679, 1999.
3. Cohen-Or, D., Chrysanthou, Y., Silva, C. and Durand, F., A Survey of Visibility for Walkthrough Applications, *submitted to IEEE Transactions on Visualization and Computer Graphics.*
4. Saona-Vazquez, C., Navazo, I. and Brunet, P., "The Visibility Octree: a Data Structure for 3D Navigation," *Computers & Graphics*, Vol. 23, pp. 635-643, 1999.
5. Teller, S. and Sequin, C., Visibility Pre-processing for Interactive Walkthroughs. *Proceedings of ACM SIGGRAPH*, pp. 61-69, 1991.
6. Zhang, H., Manocha, D., Hudson, T. and Hoff, E., Visibility Culling Using Hierarchical Occlusion Maps. *Proceedings of ACM SIGGRAPH*, pp. 124-133, 1997.
7. Zhang, H., Effective Occlusion Culling for the Interactive Display of Arbitrary Models, Ph.D. thesis, Department of Computer Science, UNC-Chapel Hill, 1998.
8. Anand, V. B. 원저, 이현찬, 채수원, 최 영 공역, *컴퓨터 그래픽스 및 형상 모델링*, 시그마프레스, 1996.
9. Watt, A., "*3D Computer Graphics*," Addison-Wesley, 1993.
10. Gottschalk, S., Lin, M. and Manocha, D., "OBBTree: A Hierarchical Structure for Rapid Interference Detection," *SIGGRAPH 1996.*
11. David Kirk, *Graphics Gems III*, Morgan Kaufmann, 1992.
12. 강성찬, 이규열, 김태완, "다중해상도해석을 위한 Boundary를 가지는 비정규 Mesh의 Normal Mesh 화 방법," 한국 CAD/CAM 학회 논문집, 제6권, 제3호, pp. 184-192, 2001. 9.
13. Woo, M., Neider, J., Davis, T. and Shreiner, D., *OpenGL Programming Guide Third Edition*, Addison Wesley, 1999.
14. Watt, A., "*3D Games Real-time Rendering and Software Technology*," Addison-Wesley, 2001.
15. Kovach, P. J., *Inside Direct3D*, Microsoft, 2000.
16. EzGRAPH, http://www.ezgraph.co.kr.
17. 정성준, 이규열, 최항순, 성우제, 조두연, "Z-Buffer와 간략화된 모델을 이용한 효율적인 가려지는 물체 제거 기법(Occlusion Culling)에 관한 연구," *2002* 한국 CAD/CAM 학회 학술발표회.
18. Garland, M., Quadric-Based Polygonal Surface Simplification, Ph.D. thesis, Computer Science Department of Carnegie Mellon University May 9, 1999.

정 성 준

2000년 서울대학교 조선해양공학과 학사
2002년 서울대학교 조선해양공학과 석사
2002년~ 쌍용정보통신
관심분야: Computer Graphics, Virtual Reality, Simulation Based Design

최 황 순

1970년 서울대학교 공과대학 조선공학과 학사
1972년 서울대학교 공과대학 조선공학과 석사
1979년 독일 Munich 공과대학 Civil Engineering 박사
1979년~현재 서울대학교 공과대학 조선해양공학과 교수
1985~1986년 Visiting Engineer at Civil Engineering Dept., MIT. Research on Solitons & Slow Drift Motions.
관심분야: Marine Hydrodynamics, Ocean Wave Mechanics, Mooring Dynamics, AUV(Autonomous Underwater Vehicle) development

조 두 연

1997년 서울대학교 조선해양공학과 학사
1999년 서울대학교 조선해양공학과 석사
1999년~현재 서울대학교 조선해양공학과 박사과정
관심분야: Computer Graphics, Virtual Reality, Computer-Aided Geometric Design

이 규 열

1971년 서울대학교 공과대학 조선공학과 학사
1975년 독일 하노버 공과대학 조선공학과 석사(Dipl.-Ing.)
1982년 독일 하노버 공과대학 조선공학과 박사(Dr.-Ing.)
1975~1983년 독일 하노버 공과대학 선박설계 및 이론연구소, 주정부 연구원
1983~1994년 한국기계연구원 선박해양공학연구센터, 선박설계, 생산자동화 연구사업(CSDP)단장
1994~2000년 서울대학교 공과대학 조선해양공학과 부교수
2000년~현재 서울대학교 공과대학 조선해양공학과 교수
관심분야: 최적설계, 형상모델링, CALS

성 우 제

1982년 서울대학교 공과대학 조선공학과 학사
1984년 서울대학교 공과대학 조선공학과 석사
1990년 MIT Department of Ocean Engineering 박사
1996년~현재 서울대학교 공과대학 조선해양공학과 조교수
관심분야: 수중음향 이미징