한국게임학회 논문집
제1권 제1호 2001년 11월

OpenGL을 이용한 대용량 Polygon Model의 View-Frustum Culling 기법

조두연*, 정성준*, 이규열**, 김태완***, 최항순**, 성우제**
* 서울대학교 조선해양공학과 대학원,
whendus1@snu.ac.kr, mildness@hananet.net
** 서울대학교 조선해양공학과 및 해양시스템
공학연구소, kylee@plaza.snu.ac.kr,
wseong@snu.ac.kr, hschoi@plaza.snu.ac.kr
*** 세종대학교 디지털콘텐츠학과, tkim@sejong.ac.kr

A View-Frustum Culling Technique Using OpenGL for Large Polygon Models

Doo-Yeoun Cho*, Sung-Jun Jung*, Kyu-Yeul Lee**, Tae-Wan Kim***, Hang-Soon Choi**,and Woo-Jae Seong**

Abstract

With rapid development of graphic hardware, researches on Virtual Reality and 3D Games have received more attention than before. For more realistic 3D graphic scene, objects were to be presented with lots of polygons and the number of objects shown in a scene was remarkably increased. Therefore, for effective visualization of large polygon models like this, view-frustum culling method, that visualizes only objects shown in the screen, has been widely used. In general, the bounding boxes that include objects are generated firstly, and the boxes are intersected with view-frustum to check whether object is in the visible area or not. Recently, an algorithm that can check in-out test of objects using OpenGL' s selection mode, which is originally used to select the objects in the screen, is suggested. This algorithm is fast because it can use hardware acceleration. In this study, by implementing and applying this algorithm to large polygon models, we showed the efficiency of OpenGL assisted View-Frustum Culling algorithm. If this algorithm is applied to 3D games that have to process more complicated characters and landscapes, performance improvement can be expected.

Key Words: *View-Frustum Culling, OpenGL, Polygon Model, Bounding Box*

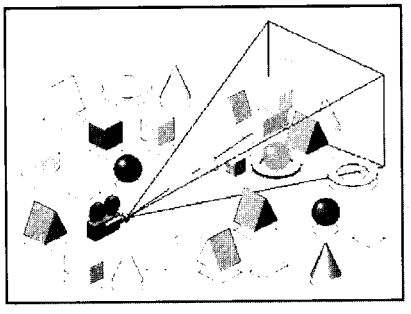
1. 서 론

최근 컴퓨터 하드웨어와 그래픽 카드의 급속한 발달로 인하여 예전에는 워크스테이션급에서나 가능했던 3차원 그래픽화면을 이제 PC에서도 손쉽게 얻을 수 있게 되면서, 현실의 물체 및 환경을 컴퓨터 상에서 구현하기 위한 가상현실 구축 및 3차원 컴퓨터 게임 등이 각광을 받고 있다. 그런데, 3차원 그래픽의 현실감에 대한 사람들의 기대와 수준이 점점 높아짐에 따라서, 물체들을 예전과는 비교할 수 없을 정도로 많은 다각형(Polygon)들로 표현하게 되었고, 또한 화면에 등장하게 되는 물체의 수 또한 증가하였다.

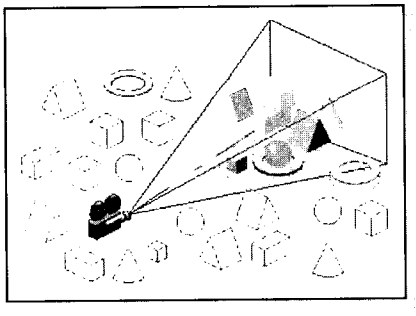
따라서 이와 같은 많은 양의 다각형 모델(Polygon model)을 모두 가시화 하려면 그래픽 파이프라인(3차원 물체를 2차원 화면으로 가시화를 수행하는 전체 과정)에 과도한 부하가 걸리게 된다. 왜냐하면 아무리 강력한 3D가속 그래픽 카드라도 단위 시간에 처리할 수 있는 다각형의 수는 제한

이 있기 때문이다. 이와 같은 문제점을 해결하기 위하여 그래픽 파이프라인이 처리해야 할 다각형의 수를 줄여주는 다양한 방법들이 많이 연구되어 왔다[1-3].

이러한 방법들 중에 하나인 View-Frustum Culling은 그림 1에서와 같이, 모든 물체를 가시화하는 것이 아니라 현재의 시야영역에 보이는 물체만을 찾아내 가시화 하여 그래픽 파이프라인에서 처리해야 할 다각형의 수를 줄이는 방법이다.



Without View-Frustum Culling



With View-Frustum Culling

그림 1. View-frustum Culling의 개념

물체가 시야영역 내에 있는지를 판단하기 위해서는 먼저 물체를 감싸는 bounding box를 생성하고, 이 bounding box가 시야영역 안에 존재하는지의 여부를 계산할 수 있어야 한다. 만약 bounding box가 시야영역 밖에 존재한다고 판단되면 그 물체는 가시화를 하지 않게 된다.

기존의 연구들은 bounding box가 시야영역 안에 포함되는지의 여부를 주로 소프트웨어적인 점, 선, 면간의 교차계산으로 구현하였다.

그런데, OpenGL의 기본기능 중 화면의 물체를 선택하기 위하여 사용되는 Selection 기능을 약간 변형하면, 교차계산을 통하지 않고 단순히 bounding box를 Selection mode로 그려주기만 하면, 시야영역에 존재하는지 아닌지를 간단하게 판단할 수 있는 방법이 최근 발표되었다[2]. 특히 이 방법은 소프트웨어적인 교차계산대신 그래픽 하드웨어를 이용하는 것이므로 구현이 간단하고 속도 또한 매우 빠른 장점이 있다.

본 연구에서는 이러한 OpenGL의 Selection mode를 이용한 View-Frustum Culling방법을 실제로 구현하여 여러 경우에 대하여 테스트를 해 보았다.

2. 그래픽 파이프 라인과 View-Frustum Culling 방법

3차원 물체를 2차원 화면에 가시화 하기 위한 그래픽 파이프라인은 크게 분류하면 그림 2와 같이 구성되어 있다[4][5]. 먼저, 물체를 이루는 다각형(Polygon)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 가지고 삼각형의 안을 채워 넣는 단계인 "Rasterization"을 거치면 화면에 그림이 그려지게 된다.

이 중 "Transform"과 "Lighting" 과정은 예전에는 컴퓨터 중앙처리장치(CPU)에 의존하였지만, 최근 들어 이들을 하드웨어적으로 지원하여 더욱 빠르게 처리할 수 있는 그래픽 카드가 하나 둘씩 등장하고 있다. 그러나, 현실감을 위해 3차원 물체를 더욱 많은 양의 다각형(Polygon)으로 모델링하는 것이 일반화되면서 대용량 Polygon model의 효율적인 가시화 기법 역시 계속해서 중요시되고 있다.

그림 2에는 물체를 이루는 모든 다각형을 가시화하는 경우와 View-Frustum Culling을 거쳐서 시야영역에 존재하는 다각형만 가시화하는 경우의 차이를 나타내고 있다.

View-Frustum Culling 은 복잡한 연산을 많이 해야 하는 그래픽 파이프라인의 작업을 줄이려는 방법이다. View-Frustum Culling을 하지 않을 때는, 전체 Scene에 있는 모든 물체를 구성하는 다각형(Polygon)이 그림 2의 그래픽 파이프라인을 거치면서 "Clipping" 단계에서야 시야에 안 들어오는 다각형이 제거되지만, View-Frustum Culling을 수행하

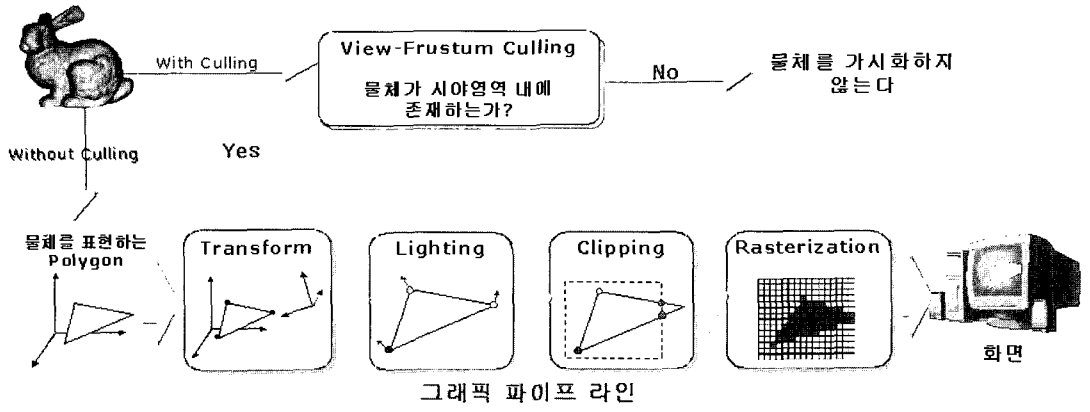


그림 2. 그래픽 파이프라인과 View-Frustum Culling의 관계

면 각 물체의 bounding box가 시야영역 내에 존재하는 지를 검사하고 나서, 시야영역에 밖에 있는 물체들은 가시화를 하지 않음으로써 그림 2의 "Transform" 부터 화면에 그려질 때까지의 모든 과정을 거치지 않게 되어 속도의 향상을 얻을 수 있게 된다.

기존의 많은 View-Frustum Culling 방법들은 물체를 감싸는 bounding box가 시야영역에 존재하는지를 판단하기 위해 직선과 평면간의 소프트웨어적인 교차계산을 수행하였다. 그런데, 최근 이러한 소프트웨어적인 방법이 아닌, OpenGL의 Selection mode를 이용하여, 그래픽 하드웨어의 파이프라인을 이용하는 View-Frustum Culling 방법이 연구되었다[2].

3. OpenGL을 이용한 View-Frustum Culling

3.1 OpenGL의 Selection mode

OpenGL에서는 화면상에 그려지는 물체를 마우스로 선택할 수 있도록 하기 위해서 Selection mode를 제공한다[6]. 일반적으로, 이 기능을 사용하기 위해서는 다음과 같은 과정을 거쳐야 한다. 먼저, 마우스가 클릭된 위치 주위로 작은 사각형 영역을 설정한다. 그리고, 각 물체마다 고유의 ID를 부여하여 Selection mode로 렌더링 모드를 전환한 후 다시 그린다(그림 3 참조). 그러면, OpenGL에서는 각 물체를 그림 2의 그래픽 파이프라인의 "Transform" 부터 "Rasterization" 까지의 과정(실제로 화면에 그리지는 않는다)을 거치면서 설정한 작은 사각형 영역 안의 내용을 변화시키는지를 조사하여, 그 물체의 ID를 selection buffer안에 저장한다. 이 과정을 모두 마치고 나면, selection buffer안에 ID에 해당하는 물체가 선택된 물체가 되는 것이다.

그림 3을 보면, ID가 1과 2인 물체는 선택영역 안에 그려지기 때문에 selection buffer안에 저장이 되었지만, ID 3인 물체는 선택영역 바깥쪽에 있기 때문에 아무런 영향을 미치지 못하여 selection buffer에 저장이 되지 않은 것을 알 수 있다.

3.2 Selection mode를 이용한 View-Frustum Culling

이러한 OpenGL의 Selection mode를 이용하여 View-Frustum Culling을 수행하는 방법의 기본적인 아이디어는 다음과 같다. 먼저 OpenGL selection을 위한 선택영역을 시야영역에 해당하는 전체 화면크기로 잡고, 물체를 감싸는 bounding box에 각각의 ID를 부여하여 Selection mode로 다시 그린다.

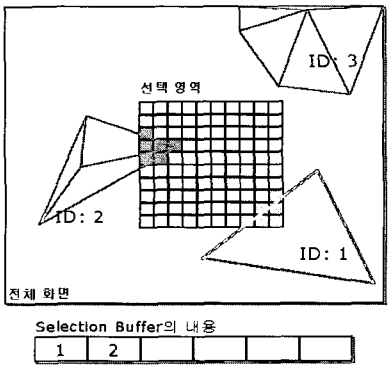


그림 3. Selection mode의 작동원리

그러면, 시야영역 내에 그려지는 bounding box의 ID만 select buffer안에 저장될 것이고, 이 ID에 해당하는 물체들만을 가시화하면, 아주 손쉽게 View-Frustum Culling을 할 수 있는 것이다(그림 4 참조).

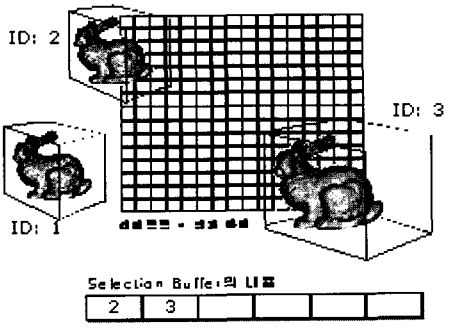


그림 4. Selection mode를 이용한 View-Frustum Culling의 원리

기존의 View-Frustum Culling 방법이 소프트웨어적으로 bounding box와 시야영역을 소프트웨어적인 교차계산을 통하여 구현되어 추가적인 CPU 처리시간을 필요로 하는 것에 비해, OpenGL의 Selection mode를 이용하는 방법은, 최근의 3D 그래픽 카드에서 하드웨어적으로 지원하는 그래픽 파이프라인을 이용하여 bounding box를 Selection mode로 한번 더 그려주기만 하면 되기 때문에 간단하게 구현이 가능하고 속도 면에 있어서도 많은 향상이 기대할 수 있다.

4. OpenGL을 이용한 View-Frustum Culling 구현 예

본 연구에서는 그래픽스분야에서 널리 사용되는 토끼 모델(bunny model)을 대상으로 OpenGL을 이용한 View-Frustum Culling을 구현하여 테스트를 실시하였다.

테스트 프로그램은 Geforce3 Ti 200 그래픽 보드를 장착한 512MB 메모리의 Pentium3 866 MHz PC상에서 Visual C++과 OpenGL 1.2 Version으로 구현되었고, 모든 테스트 scene은 650(650 pixel의 크기로 렌더링 되었다.

그림 5는 테스트에 사용된 5,000개의 polygon으로 구성된 토끼모델과 이 토끼모델 400개로 이루어진 장면을 보여주고 있다.

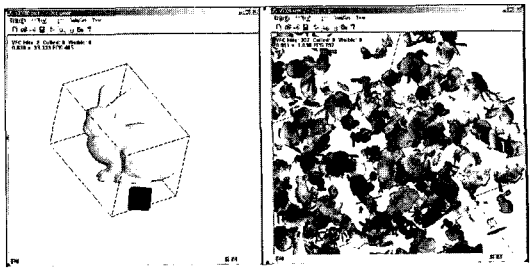


그림 5 테스트에 사용된 토끼모델
(1개당5000개 * 400개 = 200만 polygon)

그림 6은 토끼모델 400개를 임의의 순차적인 40개의 시점에 대하여, Culling과정을 거치지 않고 직접 렌더링한 결과(DR: direct rendering)와 View-Frutum Culling을 적용한 렌더링 결과(VFC)를 정리하였다.

테스트 결과 Culling을 수행하지 않고 모든 물체를 렌더링

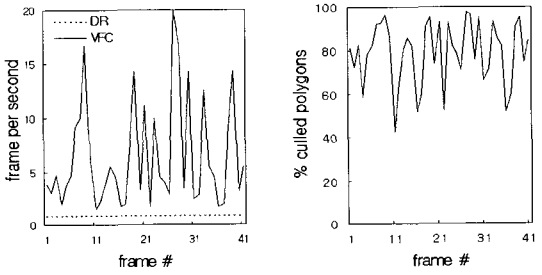


그림 6 토끼모델의 View-Frustum Culling 결과

한 결과는 시점에 관계없이 0.83 fps의 속도였다.

반면, View-Frustum Culling을 수행한 결과는 시점에 따라 다르지만 최대 33 fps, 평균 10 fps의 렌더링 속도로, Culling을 수행하지 않은 결과에 비해 약 12 배 빠른 결과를 얻을 수 있었다. 이는 평균적으로 전체의 약 79 %의 polygon이 View-Frustum Culling에 의하여 렌더링에서 제외되었기 때문이다.

그러나, View-Frustum Culling을 수행하는 데에 걸린 시간은 전체 렌더링 시간의 4%에 불과하여, 매우 효율적임을 알 수 있었다.

일반적으로 3차원 게임에서 모든 물체가 한 화면에 나타나는 경우는 흔하지 않기 때문에, 이러한 View-Frustum Culling을 적용하여, 화면상에 나타날 가능성이 있는 물체만 렌더링하게 된다면, 테스트 결과와 같이 상당한 성능향상을 기대할 수 있을 것이다.

5. 결 론

본 연구에서는 대용량 Polygon model을 실시간으로 가시화 하기 위한 기법 중 하나인 View-Frustum Culling을 OpenGL에서 제공하는 기능인 Selection mode를 이용해 구현하였다. 이 방법은 물체를 감싸는 bounding box가 시야 영역에 들어오는지 판단하기 위한 복잡한 교차계산을 할 필요가 없고 그래픽 카드의 하드웨어 3D 가속 기능을 그대로 사용할 수 있으므로, 가시화 성능에 큰 도움을 줄 수 있음을 확인했다.

향후과제로 이와 같은 OpenGL을 이용한 View-Frustum Culling 기법이 Occlusion Culling (앞의 물체에 가려진 물체를 가시화 하지 않는 방법), LOD (Level-Of-Detail; 물체의 중요도와 거리에 따라서 모델을 표현하는 Polygon수를 변화시키는 방법) 등과 함께 사용된다면, 3차원 그래픽 기술을 이용하는 가상현실, 3차원 게임제작 등에 도움을 줄 수 있으리라 생각된다.

감사의 글

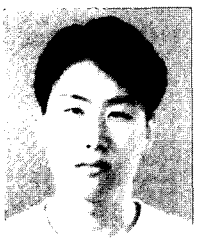
이 연구는 서울대학교 공과대학 해양시스템공학연구소 (RIMSE)의 지원과 과학기술부가 지원하는 국가지정 연구실인 서울대학교 공과대학 해양공학연구실에서 수행 중인 "해양 탐사정의 음향계측 및 지능제어 기술" 과제의 지원으로 수행되었습니다.

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. 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
4. Vera B. Anand 원저, 이현찬, 채수원, 최 영 공역, 컴퓨터 그래픽스 및 형상 모델링, 시그마프레스, 1996
5. Alan Watt and Fabio Policarpo, 3D Computer Graphics, Addison-Wesley, 1993
6. Woo, M., Neider, J., Davis, T., and Shreiner, D., OpenGL Programming Guide Third Edition, Addison Wesley, 1999

조두연

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

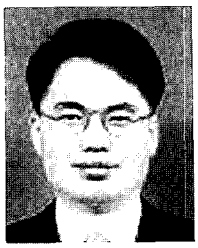


정성준

2000년 서울대학교 조선해양공학과 학사
2000년 - 서울대학교 조선해양공학과 석사과정
관심분야 : Computer Graphics, Virtual Reality, Simulation Based Design

이규열

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



김태완

1985년 한양대학교 산업공학사
1993년 Arizona State Univ. 컴퓨터과학석사
1996년 Arizona State Univ. 컴퓨터과학박사
1996년~1999년 SDRC, USA 소프트웨어 엔지니어
1999년~2001년 서울대학교 특별연구원
2001년 ~ 현재 세종대학교 디지털콘텐츠학과 학과장
관심분야: NURBS 곡선 및 곡면, 컴퓨터그래픽스, 디지털콘텐츠

최항순

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

성우제

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