이화여자대학교 대학원
2007학년도
석사학위 청구논문

그래픽스 하드웨어를 이용한 효과적 가시화 컬링 및 지형 그림자 생성 연구

컴 퓨 터 정 보 통 신 공 학 과

민 혜 정

2 0 0 8

그래픽스 하드웨어를 이용한
효과적 가시화 컬링
및 지형 그림자 생성 연구

이 論文을 碩士學位 論文으로 提出함

2008年 1月

梨花女子大學校 大學院

컴퓨터정보통신공학과 閔 惠 正

閔 惠 正의 碩士學位 論文을 認准함

指導敎授 김 영 준 _____

審査委員 최 병 주 _____

김 미 희 _____

김 영 준 _____

梨花女子大學校 大學院

목 차

그 림 목 차

표 목 차

코 드 목 차

수 식 목 차

논문 개요

본 논문에서는 그래픽스 하드웨어(GPU)의 빠른 래스터라이제이션(rasterization) 기반의 효율적인 가시화 계산을 대용량 CAD 모델의 효과적인 렌더링을 위한 전역적 가시화 컬링(visibility culling) 알고리즘과 대용량 지형 모델의 그림자 맵(shadow map) 생성 알고리즘에 적용한 방법을 제시한다.

전역적 가시화 컬링 알고리즘은 수 백 만개 혹은 수 천 만개 이상의 삼각형으로 구성된 CAD 모델을 외부에서만 관찰할 때는 내부의 보이지 않는 삼각형은 렌더링 하지 않아도 된다는 점에 착안하여, 전역적 가시화 컬링 문제를 주어진 모델의 어레인지먼트(arrangement) 문제로 재구성하였다. 어레인지먼트의 효과적인 계산을 위하여, 본 논문에서는 주어진 모델의 유향 거리 장(directed distance field)을 GPU를 이용하여 구축하고, 이를 바탕으로 패스트 마칭 방법(fast marching method)을 적용하여 어레인지먼트의 최 외각 엔벌롭(outer envelope)의 근사치를 구하였다. 그리고 본 논문에서는 대용량 지형 모델의 자체 그림자(self-shadow)를 위한 그림자 맵의 빠른 생성을 위하여 GPU의 차폐 질의문(occlusion query)을 사용한 광선 추적법(ray-tracing) 적용 방법을 연구하였다.

본 논문에 제시된 전역적 가시화 컬링 알고리즘을 약 3백만 개 이상의 삼각형으로 구성된 CAD 모델에 적용하여, 70%이상의 가시화 컬링 효과를 거둘 수 있었고, 전체 계산 시간은 nVIDIA GeForce 7900GX2를 장착한 Dual

AMD Athlon 64 2.6GHz PC에서 평균 6.5초의 시간이 소요되었다. 또한 1024×1024 크기의 높이 맵(height map)이 이용된 지형 그림자 맵 생성은 nVIDIA GeForce 6800을 장착한 AMD Athlon 64 2.2GHz PC에서 평균 7.32초가 소요되었다.

Ⅰ 서 론

1.1 연구의 배경

1.1.1 그래픽스 하드웨어의 발전 과정

지난 몇 년간 컴퓨터 그래픽스 산업은 게임에서부터 영화, 애니메이션(animation), 가상 현실(virtual reality) 등에 걸쳐 폭넓게 확대되며 성장해오고 있다. 더불어 자연스러운 영상을 만들어내기 위한 3차원 그래픽스 기술도 발전하고 있다. 따라서 현재의 3차원 그래픽스 분야에서는 뛰어난 연산 수행 능력과 넓은 메모리 대역폭을 가진 그래픽스 하드웨어(GPU)에 대한 요구가 증대 되었고, 이는 GPU 기술의 발전으로 이어져 그래픽스 처리의 가속화를 가능하게 했다. 과거 데스크탑 PC용 그래픽 카드를 중심으로 성장해온 GPU는 최신 기술이 동원되어 그래픽 성능을 극대화시키며 디지털 TV, 게임 콘솔뿐 만 아니라 노트북, 모바일 기기에도 그 사용이 꾸준히 증가 되고 있다.

일반적으로 3차원 그래픽 시스템은 3단계의 처리 과정으로 구성되어 있다. 응용프로그램 단계(application stage), 지오메트리 단계(geometry stage), 그리고 래스터라이제이션 단계(rasterization stage)이다[35]. 응용프로그램 단계는 사용자 입력 처리, 3D 오브젝트간 충돌과 같은 물리적 연산을 담당한다. 지오메트리 단계에서는 응용프로그램 단계로부터 입력된 정점의 변환

(transformation)과 광원에 의한 색이 계산된다. 정점은 프리미티브(primitive) 단위로 묶인 후 내부를 채운 각 픽셀의 색을 결정하기 위해 래스터라이제이션 단계로 보내진다. 마지막 단계인 래스터라이제이션 단계에서는 픽셀 별 연산을 통해 각 픽셀의 색을 결정하여 최종 결과를 프레임 버퍼(frame buffer)에 저장한다.

1990년대 중반에 등장한 1세대 GPU는 래스터라이제이션 단계만 가속하는 간단한 구조였다. 1990년대 후반부터 2000년대 초반까지의 2세대 GPU는 고정된 기능을 통해 지오메트리 단계를 포함한 전체 그래픽스 연산을 가속하고 빠른 속도로 렌더링 하였다. 2000년대 초반의 3세대 GPU에는 래스터라이제이션 단계를 프로그래밍 할 수 있는 픽셀 셰이더(pixel shader)가 포함되었고 지오메트리 단계도 정점 셰이더(vertex shader)로 프로그래밍 가능해졌다. 그 결과 이전의 고정된 기능의 GPU가 갖던 한계를 극복하여 보다 실제에 가까운 표현이 가능하게 되었다. 이러한 형태의 GPU는 현재 가장 보편적으로 사용되고 있으며 cg[35], HLSL[36], GLSL[37] 등 C언어와 비슷한 셰이딩 언어(shading language)가 개발되었다(그림 1-1). 가장 최근에는 픽셀 셰이더나 정점 셰이더 어느 한쪽에 연산량이 집중되는 문제를 해결하기 위해 기능이 통합된 통합 셰이더(unified shader)를 GPU에 내장하는 기술이 개발되고 있다[38].

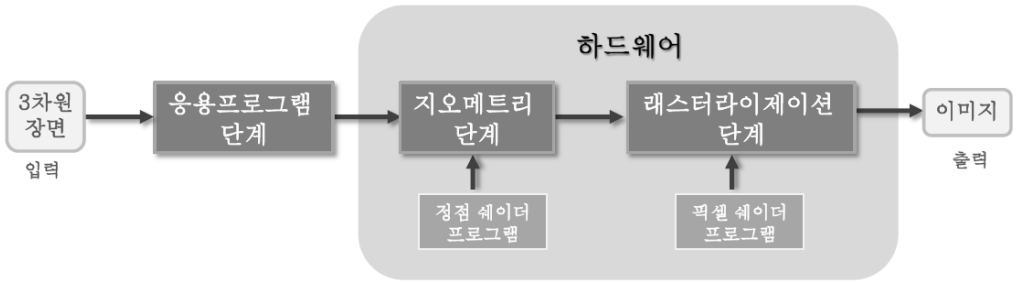


그림 1-1

그래픽스 처리과정과 쉐이더 프로그램

1.1.1.1 래스터라이제이션의 연산

GPU는 그래픽스 처리과정의 마지막 단계인 레스터라이제이션 단계에서 각 픽셀마다 깊이 값, 보간 된(interpolation) 정점 색, 보간 된 텍스쳐(texture) 좌표 등을 계산한다. 이런 정보들과 픽셀 자체의 위치를 하나로 묶어 단편(fragment)이라고 한다. 그림 1-2은 각 단편에 대해 GPU가 수행하는 연산들을 보여준다. 먼저 가위 테스트(scissor test)을 실시하여 가위 사각형이라고 하는 렌더링이 일어나는 영역을 지정하고 그 영역 안의 단편들만 통과시킨다. 통과된 단편들은 알파 테스트(alpha test)를 거친다. 알파 값은 단편의 투명한

정도를 의미한다. 알파 테스트는 단편의 최종 알파 값과 미리 정의한 값의 비교 연산을 하여 단편의 폐기를 결정한다. 알파 테스트를 통과한 단편은 스텐실 테스트(stencil test)로 넘어간다. 스텐실 버퍼(stencil buffer)에서 단편 위치에 해당하는 값을 얻고 그것을 미리 정의한 값과 비교 연산하여 폐기를 결정한다. 통과한 단편은 계속해서 깊이 테스트(depth test, Z test)를 실시한다. 깊이 테스트에서는 단편에 부여된 최종 깊이를 현재 깊이 버퍼(depth buffer)에 저장되어 있는 깊이 값과 비교 연산을 하여 단편의 폐기를 결정한다. 만약 단편의 깊이가 깊이 버퍼에 이미 있는 값보다 작거나 같을 때 단편을 통과시키는 비교 연산을 한다면, 즉 시점에 더 가까운 단편들을 통과시킨다면, 깊이 버퍼를 단편의 깊이로 갱신한다. GPU는 깊이 테스트를 이용하여 차폐 질의문(occlusion query)을 지원한다. 차폐 질의문은 정의한 깊이 비교 연산에 따라 질의의 반환 값으로 깊이 테스트에 통과한 픽셀의 수를 돌려준다. 차폐 질의문을 통해서 현재 화면에 그려지는 객체가 이전에 그려진 객체에 의해 가려지는지의 여부를 알 수 있다. 만약 깊이 비교 연산이 GL_LEQUAL이고 차폐 질의문의 반환 값이 0이라면 이는 렌더링 된 객체가 차폐물(occluder)에 의해 완전히 가려진 것을 뜻한다. 즉 렌더링 된 객체의 깊이 값이 깊이 버퍼에 저장되어 있던 값보다 크기 때문에 비교 연산을 통과하지 못하여 픽셀의 수는 0이 되고, 현재의 시점에서는 픽셀이 보이지 않는 것을 의미하는 것이다. 단편이 앞선 모든 테스트를 모두 통과하면 단편 색과 컬러 버퍼(color buffer)의 값을 혼합(alpha blending)하여 최종 색을 컬러 버퍼에 저장한다.

본 논문에서는 래스터라이제이션의 깊이 테스트 단계에서 이루어지는 깊이 비교 연산(depth buffering)을 응용하여 효율적인 깊이 비교를 통해 높은 성능의 가시화 계산을 하는 두 가지의 애플리케이션을 보여준다.

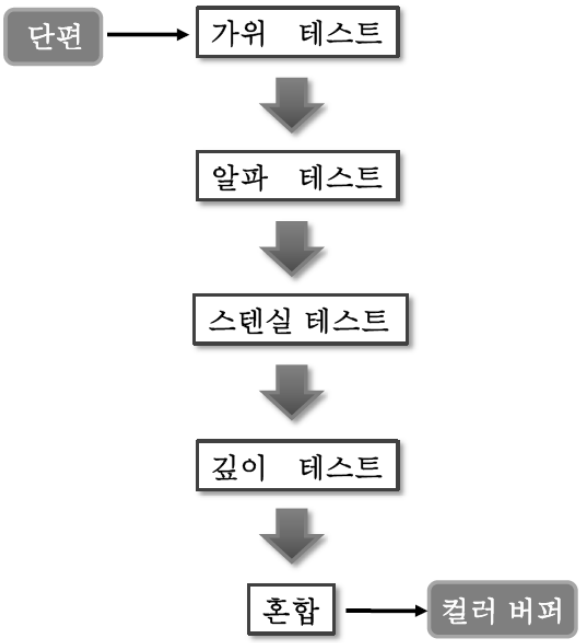


그림 1-2

래스터라이제이션 연산 과정

1.1.2 가시화 컬링

디자인 리뷰, 교육용 시뮬레이션, 시각 감시 장치 등의 애플리케이션에서 사용되는 모델들은 흔히 수백 혹은 수천만 개 이상의 삼각형으로 모델되는 복잡한 부품들로 구성되어 있다. 최근 사용되고 있는 최상위급 그래픽스 하드웨어를 사용하더라도 이런 복잡한 모델을 실시간에 렌더링 하기는 여전히 어려운 일이다.

흔히 매우 복잡한 모델의 렌더링 가속화를 위해서 사용되는 기법으로는 모델 단순화(model simplification)[11], LOD(level of detail) 기법[14], 가시화 컬링(visibility culling)[16] 등의 방법이 있다. 특히 가시화 컬링 기법은 사용자의 시점이 고정되어 있거나 위치의 영역이 알려져 있는 경우에 효과적으로 시점으로부터 가려져 있는 기하 요소를 제거하여 렌더링 성능을 향상 시킬 수 있는 방법이다.

CAD 애플리케이션에서는 주어진 CAD 모델을 오직 외부에서만 관찰하는 경우가 빈번한데, 이 경우 모델 내부에 가려져 있는 기하 요소들을 제거하여도 최종 렌더링 결과에는 영향이 없다. 이러한 내부 요소의 제거를 수작업으로 행할 수도 있으나 그럴 경우 막대한 시간과 노력이 들어가게 되므로 이를 자동적으로 수행하는 것이 효과적이다. 이 문제는 렌더링 가속화의 여러 기법 중 가시화 컬링과 연관되는 문제이나, 기존의 가시화 컬링 문제와는 달리 주어진 모델을 무한히 많은 카메라 시점에서 볼 때 가려지는 부분을 찾는 문제라는 점에서 상이하다. 복잡한 디지털 모델에서

내부의 보이지 않는 부분을 제거하는 방법 중 하나로 무한히 많은 시점에서 모델을 관측하여 보이는 부분이 어디인지 판단한 다음 그렇지 않은 부분을 제거하는 방법이 있다. 하지만 이 방법은 보이지 않는 요소를 찾아내기 위한 확률은 높지만 결정적인 방법이 되지는 못한다. 또한 많은 샘플링이 필요하기 때문에 속도가 느려질 수 있고, 구현 역시 쉽지 않다[15].

1.1.3 지형 렌더링과 그림자 생성

지형(terrain)은 영화나 게임, 가상 현실, GIS(geographic information system), 비행 시뮬레이션(flight simulation) 등에서 외부 환경을 표현하는데 중요한 요소 중 하나이다. 위성사진이나 지도 등에서 얻어진 지형 데이터는 2차원 평면상의 좌표와 그 점에서의 높이 값의 집합으로 구성되어 있다. 지형 데이터는 대부분 그 크기가 방대하고 지형을 사실적으로 렌더링하기 위해 필요한 삼각형의 개수는 수백만 개 혹은 수 천만 개에 이르기도 한다. 이런 특성 때문에 대용량 지형 모델의 렌더링을 위해 가시화 컬링이나 LOD 기법[41]을 이용하여 가속화한다. 특히 지형 렌더링의 LOD기법은 시점에서 가까운 곳은 높은 상세도로 표현하고 시점에서 먼 곳이나 보이지 않는 곳은 상세도를 낮추어 지형을 단순화 하여 계산 비용을 줄이는 기법이다(그림 1-3).

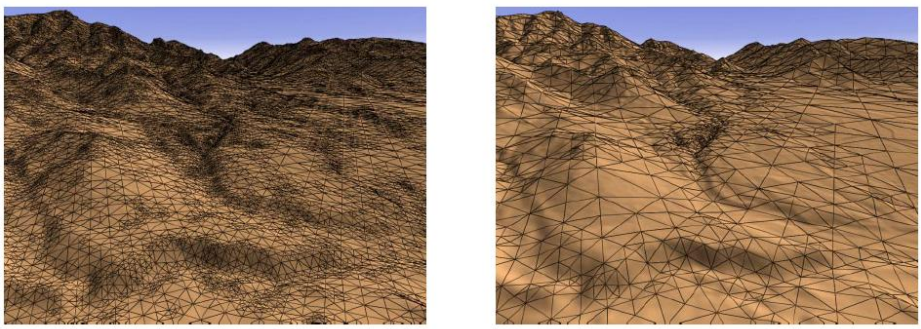


그림 1-3

LOD기법을 사용하여 렌더링한 지형 모델[41]

그림자는 빛이 물체에 가려져서 생기는 어두운 영역으로 장면을 사실감 있게 표현하고, 공간감과 물체들간의 상대적인 관계를 파악하도록 해주며 더 풍부한 표현을 가능하게 해주는 역할을 한다(그림 1-4)[19]. 대표적인 그림자 생성 기법으로는 그림자 맵(shadow map)[20], 광선 추적법(ray-tracing)[28,29], 그림자 다각형(shadow polygon)[30], 그림자 볼륨(shadow volume)[31] 등이 있다. 특히 지형의 그림자를 생성할 때에는 기본적으로 지형의 경사에 근거하여 그림자를 계산하거나 지형에 의한 자체 그림자가 계산되어야 한다. 현재 지형 모델의 그림자를 생성하고 렌더링하는 기법의 연구는 부족한 상황이다. 지형 모델은 대부분 수백 혹은 수천 만 개가 넘는 삼각형들로 구성된 대용량 모델이기 때문에 이런 지형의 그림자를 생성하고 렌더링을 하는데 많은 시간이 소요되기 때문이다.

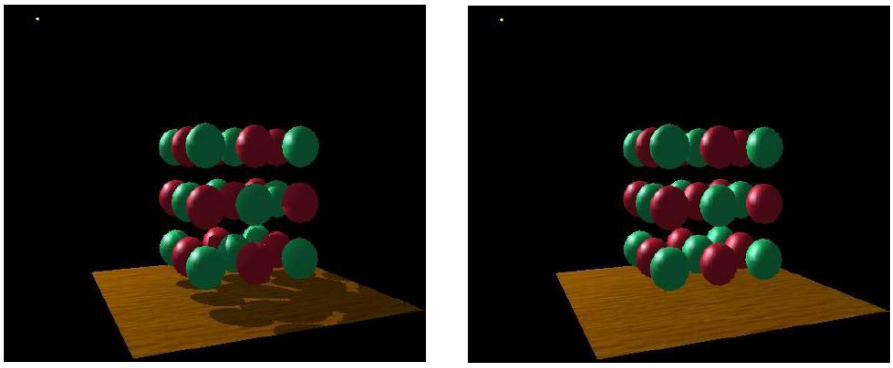


그림 1-4

그림자 효과. (왼쪽)그림자를 이용해 사실감을 표현 (오른쪽)그림자가 없을 경우[39]

1.2 연구의 목표

본 논문에서는 GPU를 이용해 효율적으로 가시화 계산을 한 전역적 가시화 컬링 알고리즘과 빠른 그림자 맵 생성 알고리즘을 제시한다.

대용량 모델 렌더링을 위한 전역적 가시화 컬링 알고리즘은 외부에서 수 백 만개 혹은 수 천 만개 이상의 삼각형으로 구성된 CAD 모델을 관찰할 때는 내부의 보이지 않는 삼각형은 렌더링 하지 않아도 된다는 점에 착안하여 내부의 삼각형을 제거하는 전역적 가시화 컬링 문제를 주어진 모델의 어레인지먼트 문제로 재구성하였다. 어레인지먼트의 효과적인 계산을 위하여, GPU를 이용하여 주어진 모델의 바운딩 박스(bounding box)를 기본으로 한 유향 거리장을 구축하고, 이를 바탕으로 패스트 마칭 방법을 적용하여 어레인지먼트의 최 외각 엔벌롭의 근사치를 구한다.

또한 본 논문에서는 수 백 만개 혹은 수 천 만개가 넘는 삼각형으로 구성된 지형 모델의 자체 그림자를 짧은 시간에 생성고자 광선 추적법을 적용하여 그림자 맵을 빠르게 생성한다. 먼저 주어진 지형 모델의 높이 맵을 렌더링 한 후, 높이 맵의 각각의 격자 점에서 광원까지 광선을 발생시켜 광선과 지형의 충돌 여부를 검사한다. 만약 충돌했다면 광선을 발생시킨 격자 점은 그림자라고 간주한다. 충돌 검사를 위하여, 광선 렌더링 시 차폐 질의문을 사용하여 GL_GREATER로 정의된 깊이 연산에 따라 지형의 깊이 값보다 광선의 깊이 값이 더 큰 픽셀 개수를 반환한다. 만약 반환 값이 1 이상이라면

광선은 지형에 의해 가려졌음을 알 수 있고 지형에 충돌했다고 볼 수 있다. 즉 그림자 맵 생성 알고리즘은 픽셀 단위의 계산이 이루어지는 이미지 프리시젼(image precision)기법이라고 볼 수 있다. 높이 맵의 모든 격자점에서 광원까지 발생시킨 광선에 차폐 질의문을 실시하여 충돌 여부를 검사하고 그 정보를 바탕으로 높이 맵에 대응되는 그림자 맵을 생성한다. 최종적으로 생성된 그림자 맵을 바탕으로 그래픽스 처리과정 중의 하나인 지오메트리 단계를 프로그래밍하는 cg 정점 셰이더를 이용하여 사실감 있는 지형 그림자를 렌더링한다.

II 관련 연구

2.1 가시화 컬링

2.1.1 가시화 계산

주어진 시점에서 씬의 보이는 부분을 계산(visibility)하는 문제는 컴퓨터 그래픽스, 계산 기하학, 컴퓨터 비전 분야에서 다루는 기본적인 문제이다[3]. 가시화 컬링은 가려진 서피스를 제거하는 과정을 수행하기 전에 보이는 않는 불필요한 오브젝트들을 그리지 않도록 하는 작업이다[16]. 이것은 적어도 스크린의 한 픽셀을 차지하는 프리미티브의 부분 집합, 즉 보이는 부분만을 그리는 작업과 동일하다. 가시화 컬링 연구와 관련하여 보이는 부분의 근사치를 계산하는 알고리즘들이 많이 제시되어 왔다. 고전적인 가시화 컬링은 크게 두 종류로 분리할 수 있다. 뒷면 제거 기법(back-face culling)은 시선과 반대 방향의 면을 제외하고 렌더링하는 것이고, 시야 절두체 컬링 기법(viewing-frustum culling)은 관측 공간 밖의 물체를 제외하고 렌더링하는 것이다. 이를 구현한 효과적인 기술들과 최적화 기법이 개발되었다[2,13].

보이는 물체를 확실히 찾아내기 위하여 보이지 않는 물체를 정확하게 판단하는 방법은 계산 시간이 많이 소모되므로 대부분의 알고리즘은 잠재적으로 보이는 부분(potentially visible set, PVS)을 찾는데 주력한다[6,18]. PVS는

보이지 않는 부분을 포함하더라도 모든 보이는 부분을 찾아낼 수 있는 방법으로 포괄적 가시성 계산(conservative visibility)이라고도 불린다. 지금까지 연구되어온 가시화 알고리즘들은 복잡한 환경에서 PVS를 실시간에 판단하기 어렵다[5]. 최근에 등장한 PVS의 계산 수행 능력을 향상시키기 위한 몇 가지 접근 방법으로는 지역 기반 가시화 알고리즘[4], 하드웨어를 이용한 차폐 질의문의 사용[7], 병렬적 렌더링 파이프 라인 사용[8] 등이 있다. 또 실시간에 크기가 큰 3차원 데이터 집합을 디스플레이하기 위해 이미지 제너레이션의 속도를 높이기 위한 가시화 문제를 해결하는 알고리즘들이 제시되고 있다[1,9,17,18].

Ernst의 논문[15]에서는 복잡한 메쉬에서 보이지 않는 삼각형을 제거하기 위해 3가지 방법으로 보이는 삼각형을 찾아내었다. 첫 번째로 먼저 메쉬 주변에 여러 대의 카메라를 위치시킨 다음 보이는 삼각형을 찾아내었고, 두 번째로 아직 발견되지 않은 보이는 삼각형을 찾아내기 위해 삼각형의 중심에 가상의 정육면체를 놓아 여섯 면으로 렌더링하여 만약 어떤 이미지가 렌더링이 되었다면 그 삼각형은 보이는 것으로 판단한다. 마지막으로 발견되지 않은 남은 삼각형 중 보이는 것을 찾아내기 위해 Monte Carlo 광선 추적 법을 사용하였는데, 어떤 삼각형으로부터 광선을 방사했을 때 무한히 먼 외부로 방사될 수 있다면 보이는 삼각형으로 간주하는 것이다.

2.1.2 어레인지먼트 계산

R^d공간에 있는 유한개의 기하 오브젝트들에 대해서 이들의 어레인지먼트란 R^d를 연결된 열린 셀(open cell)로 분할(decomposition)하는 것을 말한다[10].

일반적으로 R^d상의 n개의 서피스들의 어레인지먼트는 $O(n^d)$의 조합 복잡도를 가진다고 알려져 있고, 특히 R^3에서의 어레인지먼트는 $O(n\lambda_q(n)\log n + V\log n)$의 계산 복잡도를 가진다고 알려져 있다[10]. 여기서 V는 수직 분할(vertical decomposition)의 조합 복잡도이고, q는 서피스들의 차수에 비례하는 상수이며, $\lambda_q(n)$은 (n,q) Davenport-Schinzel 순열의 최대 길이이다. 하지만 다양한 예외적 상황들과 수치 연산의 에러 문제 때문에 어레인지먼트를 3차원 이상의 고차원에서 정확하고 강건하게 구하는 것은 어렵다고 알려져 있다.

2.2 그림자 생성

기본적인 그림자 생성 방법은 Woo의 논문에 체계적으로 설명이 되어 있다[19]. 일반적으로 그림자 생성 알고리즘은 이미지 프리시젼 방법(image precision method)과 오브젝트 프리시젼 방법(object presicion method)으로 나뉜다.

2.2.1 이미지 프리시젼 방법

그림자 맵 방법은 광원을 시점으로 하여 깊이 이미지(depth image)를 얻어낸 후 렌더링 할 때 각 픽셀에 대응되는 물체의 한 점의 깊이 값과 깊이 이미지에 저장된 값을 비교해서 그림자 여부를 판단하는 방법이다[20]. 현재의 GPU는 그림자 맵을 구현하는 방법을 지원하고 있다[21,22].

그림자 맵의 큰 단점은 그림자 경계에서 앨리아싱(aliasing)이 생긴다는 것이다. 앨리아싱은 그림자의 경계(edge)에서 발생하는데, 크게 원근 앨리아싱(perspective aliasing)과 투영 앨리아싱(projective aliasing)으로 나눌 수 있다[23]. 원근 앨리아싱은 광원보다 시점에 아주 가까운 지점에서 발생하고 투영 앨리아싱은 시점방향보다 광원방향의 표면 법선의 각도가 더 클 때 발생한다. 앨리아싱 문제를 해결하기 위한 많은 방법이 제안 되었다. Reeves는 일반적인 필터링 과정 대신 퍼센티지 클로져 필터링(percentage closer-filtering) 방법으로 그림자 경계를 블러링하여 앨리아싱을 줄였다[24]. Brabec는 퍼센

티지 클로져 필터링을 하드웨어 기반의 그림자 맵 렌더링에 사용했다[25]. 앨리아싱 문제는 그림자 맵의 레졸루션이 부족한 경우에 발생하므로 이를 보완하기 위한 방법으로 그림자 맵의 레졸루션을 효과적으로 늘리는 적응적 그림자 맵(adaptive shadow map) 기법도 연구된다[26]. 그러나 이 기법은 큰 모델을 인터랙티브하게 렌더링을 하기엔 너무 느리고 움직이는 광원을 표현하기 힘들다. 적응적 그림자 맵을 응용하여 다양한 레졸루션의 멀티 그림자 맵을 사용하기도 한다[27]. 하나의 그림자 맵을 이용하면서도 필요한 부분에만 그림자 맵에 정보를 자세히 표현할 수 있는 방법인 원근 그림자 맵(perspective shadow map)도 연구된다[23].

그림자를 생성하기 위한 이미지 프리시젼 방법으로 광선 추적법(ray-tracing)을 사용하기도 한다. 빠른 광선 추적법을 위해 PCs의 클러스터나 공유메모리 멀티프로세서 시스템을 이용 하여 계산 속도를 줄인다[28,29]. 본 논문에서는 광선 추적법을 적용한 기존의 그림자 맵 생성 방법에 GPU의 래스터라이제이션 기능을 적용하여 그림자 맵을 빠르게 계산하는 방법을 제시한다[38].

2.2.2 오브젝트 프리시젼 방법

오브젝트 프리시젼 방법은 그림자 경계를 정확히 계산함으로써 경계의 앨리아싱 문제를 해결한다. Blinn은 그림자가 드리워지는 면에 차폐물의 정점을 투영시켜서 생기는 그림자 다각형(shadow polygon)을 어둡게

렌더링하여 그림자를 표현했다[30]. 그러나 그림자가 드리워지는 모델이 복잡할 경우 여러 번 투영해야 하고 그림자 다각형과 기존 다각형들이 같은 평면상에 존재하기 때문에 역시 앨리아싱이 생길 수 있다.

오브젝트 프리시젼의 대표적인 기법으로는 그림자 볼륨(shadow volume) 알고리즘이 있다[31]. 그림자 볼륨 알고리즘은 그림자가 될 가능성이 있는 영역을 볼륨의 형태로 저장하여 그림자를 생성하는 방법이다. 그림자 볼륨은 그림자를 생성하는 다각형과 빛의 위치에서 다각형의 각 변을 빛의 방향으로 확장하여 생성되는 사각형들로 정의한다. 그림자 볼륨을 표현하기 위해 BSP 트리를 사용하기도 한다[32,33]. 이 기법은 광원이 동적일 경우 광원이 이동할 때 마다 트리 전체가 재구성 되어야 하는 단점이 있다. Heidmann은 그래픽스 하드웨어의 스텐실 버퍼를 사용하여 다각형이 그림자 볼륨과 교차하는지 검사하여 효율적으로 그림자를 생성할 수 있는 방법을 제안했다[34].

III 대용량 모델 렌더링을 위한 전역적 가시화 컬링 알고리즘

본 장에서는 GPU 를 이용한 대용량 모델 렌더링을 위한 전역적 가시화 컬링 알고리즘을 제시한다. 수 백 만개 혹은 수 천 만개 이상의 삼각형으로 구성된 CAD 모델을 외부에서만 관찰할 때는 내부의 보이지 않는 삼각형은 렌더링 하지 않아도 된다는 점에 착안하여 전역적 가시화 컬링 문제를 주어진 모델의 어레인지먼트 문제로 재구성하였다. 어레인지먼트의 효과적인 계산을 위하여, 주어진 모델의 유향 거리 장을 GPU 를 이용하여 구축하고, 이를 바탕으로 패스트 마칭 방법을 적용하여 어레인지먼트의 최 외각 엔벌롭의 근사치를 구한다. 최종적으로 최 외각 엔벌롭에 포함되어 있지 않는 삼각형 서피스를 구하고, 그 부분을 보이지 않는 삼각형 서피스로 간주해 제거한다.

3.1 가시화 컬링 알고리즘

어레인지먼트를 구성하는 셀들 중에서 최 외각 셀 혹은 최 외각 엔벌롭은 외부 또는 무한대의 지점에서 연속적인 경로를 따를 때 도달 가능한 셀로 정의된다(그림 3-1). 본 논문에서는 보이지 않는 요소를 제거하기 위해 삼각형 메쉬로 구성된 서피스의 최 외각 엔벌롭을 계산하여 최 외각 엔벌롭을

형성하는 삼각형 집합을 구하고자 한다. 왜냐하면 최 외각 엔벌롭을 제외한 나머지 부분을 제거하는 것은 주어진 삼각형 메쉬의 보이지 않는 부분을 제거하는 것과 같기 때문이다. 본 논문에서는 최 외곽 엔벌롭을 계산하기 위해 강건하고 효과적인 방법으로 주어진 서피스를 이산화하고, 이에 기초하여 최 외각 엔벌롭 경계의 근사치를 구한다[12].

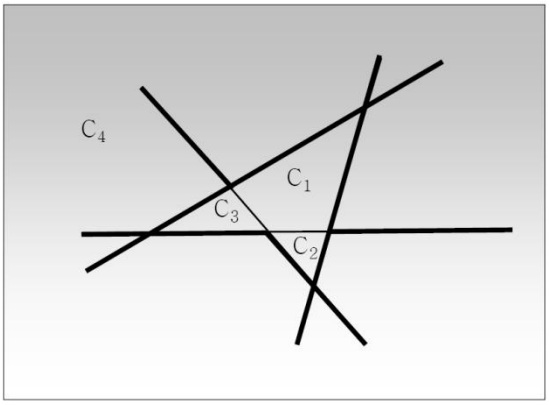


그림 3-1
어레인지먼트: 5 개의 선분을 이용하여 R^2공간을 c1, c2, c3, c4 의 셀로 분할하였다. 최 외각 엔벌롭은 무한대의 공간에서 도달할 수 있는 셀의 경계이고, 굵은 선으로 표시하였다.

본 논문에서 제안하는 가시화 컬링 알고리즘은 크게 3 단계로 구성된다. 1 단계에서는 GPU 를 이용하여 주어진 삼각형 서피스의 이산화된 형태를 유향 거리 장로 나타낸다. 2 단계에서는 생성된 유향 거리 장을 패스트 마칭

기법을 이용하여 서피스 어레인지먼트의 최 외각 엔벌롭을 구한다. 3 단계에서는 앞서 계산된 최 외각 엔벌롭에 포함되어 있지 않는 삼각형 서피스를 구하고, 그 부분을 보이지 않는 삼각형 서피스로 간주해 제거한다.

3.1.1 거리 장 표현법

흔히 거리 장(distance field)은 3 차원에서의 격자 점들(혹은 복셀)에서 계산되고, 각각의 복셀은 서피스까지의 최단거리의 스칼라 값을 가지게 된다. 본 논문에서 사용한 유향 거리 장(directed distance field)에서의 복셀은 각각 x-, x+, y-, y+, z-, z+의 주축 방향을 향한 최단거리 값을 가진다. 이러한 유향 거리 장을 구하기 위해 거리 장을 슬라이스로 나누고, 이 슬라이스를 거리 장에서 격자 크기와 동일한 크기만큼 이동시키며 순서대로 하나씩 2 차원 슬라이스를 생성한다. 여기에서 연속된 두 슬라이스에 대응하는 평면을 슬랩이라고 부르고, 각 슬랩마다 교차하는 서피스 프리미티브 집합의 거리 장을 계산하게 된다. 각 슬라이스의 거리 장을 만들기 위해 GPU 를 이용하여 슬랩에 교차하는 서피스 프리미티브들을 슬라이스를 품고 있는 2 차원 평면으로 직교 투영하여 렌더링한다. 그 결과, 현재 슬라이스의 격자점에 해당하는 프레임 버퍼의 각 픽셀에서의 깊이 버퍼에는 그 점에서의 거리 값이 저장된다(그림 3-2).

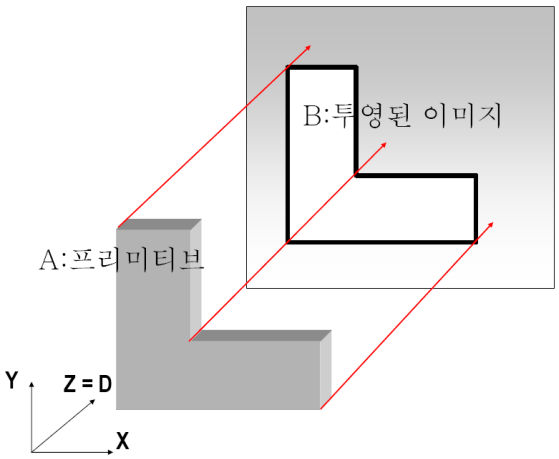


그림 3-2

유향 거리장 구하기: A 프리미티브가 이미지 평면인 슬라이스에 D방향으로 직교 투영되어 렌더링되면 깊이 버퍼에는 A의 거리 값이 저장된다. B는 A 프리미티브가 슬라이스에 직교 투영된 모습이다.

또한 거리 장을 계산하기 전에 각 삼각형 서피스에 고유한 컬러 값을 할당한다. 유향 거리 장 계산 시 렌더링 된 컬러 버퍼에는 할당한 삼각형 서피스의 고유 컬러 값이 저장되고, 이는 최종적으로 보이는 서피스를 구별하기 위한 인덱스로 사용된다.

3.1.2 패스트 마칭

본 연구의 목적은 서피스 집합의 유향 거리 장을 생성하고, 이를 이용하여 서피스의 최 외각 엔벌롭을 찾는 것이다. 이를 위해서 유향 거리장의 모든

격자 점이 최 외각 엔벌롭을 기준으로 내부/외부인지를 분류해야 한다. 그러기 위해서 유향 거리 장에서 모델을 감싸고 있는 가장 먼 격자 점을 프런트(front)라고 하고 그 프런트들로부터 엔벌롭을 향해 프런트 확장(front propagation)을 진행한다(그림 3-3).

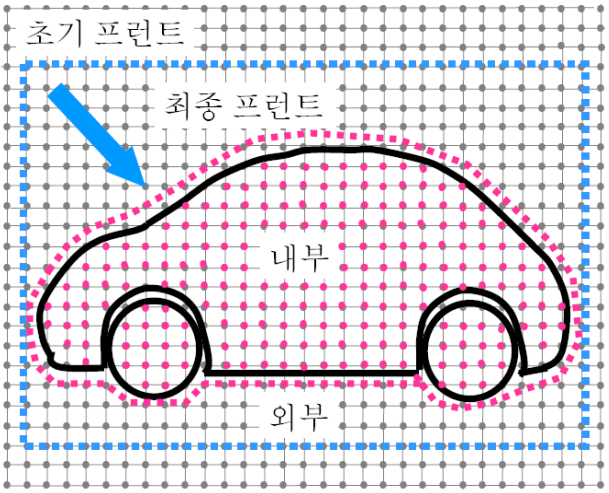


그림 3-3

프런트 확장

프런트 확장을 수행하기 위하여, 현재의 프런트에 '이미 방문되었는지(K)', '지금 방문 중인지(T)', 혹은 '방문 될 것인지(F)'을 구분하는 태그를 붙이고, 서피스 외부의 격자 점을 방문할 때마다 태그 정보를 업데이트한다. 초기 프런트를 구성하는 격자 점의 태그는 T이다. 또한 각 격자 점은 내부/외부를 구분하는 표식을 가지고 있고, 모든 격자 점의 표식은 내부라고 초기화 한다. 그런 후 프런트 확장을 진행하면서 다음과 같은 업데이트 과정을 거친다:

1. 프런트를 구성하는 격자 점을 선택하고 그 점을 프런트에서 제거하며 태그를 K로 정한다. 이 점을 *P*라고 하자.
2. *P*의 이웃 점을 *Q*라고 한다. *Q*의 태그가 K이라면 업데이트를 하지 않고, 그렇지 않다면 *P*의 유향 거리 장이 *P*와 *Q*사이의 거리보다 큰지 검사한다. *P*의 유향 거리 장이 크다면 *Q*는 서피스 경계의 외부의 점이라고 볼 수 있다. 그러므로 프런트를 *Q*로 전진시키고 *Q*는 새로운 프런트가 된다. 그리고 *Q*의 표식은 외부로 정한다.
3. 1-2 과정을 반복하여 프런트가 서피스 외부의 모든 격자 점을 방문할 때까지 프런트 확장을 수행한다. 프런트 확장이 종료되면 모든 격자 점이 서피스의 내부/외부인지가 결정된다.

3.1.3 보이는 서피스 추출

앞서 수행한 프런트 확장의 결과 최 외각 엔벌롭의 근사치를 구성하는 격자 점이 계산되었고, 최 외각 엔벌롭을 기준으로 유향 거리 장의 모든 격자 점이 내부/외부인지 분류되었다(그림 3-3). 최 외각 엔벌롭을 구성하는 격자 점의 유향 거리 장 값이 외부에서 내부로 바뀐다는 것은 그 거리 장 방향에 서피스의 경계가 존재한다는 것을 의미한다. 이때 컬러 버퍼에서 컬러 값을 읽고(그림 3-4), 이 컬러 값을 가진 삼각형 서피스의 집합이 최종적으로 보이는 서피스가 된다.

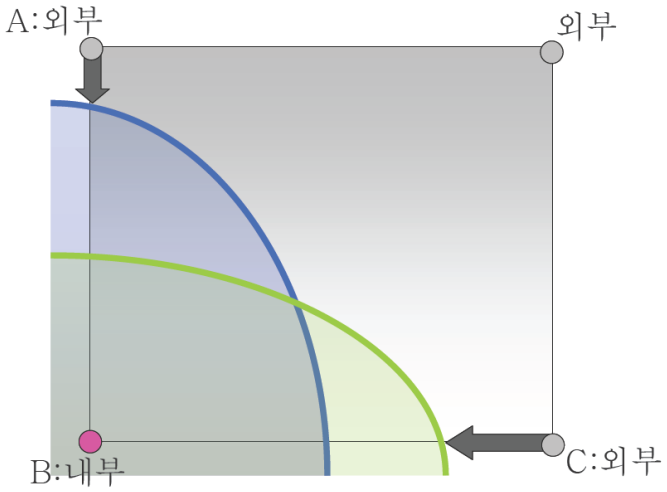


그림 3-4

격자 점에서의 서피스 경계: 최종 프런트인 A, C 에서 내부로의 방향만 고려하여 서피스 경계를 찾아내고 그 때의 씬 그래프 노드의 컬러 값을 읽는다.

본 논문에서 제시한 알고리즘은 격자 해상도를 조절함으로써 가시화의 정확도를 결정하는 근사적 가시화 컬링을 수행할 수 있다. 즉 높은 해상도에서 가시화 컬링을 수행할 경우, 계산 시간이 더 투여되는 대신 정확한 가시화 컬링 결과를 도출 할 수 있고, 낮은 해상도에서 수행할 경우 빠른 계산 시간 내에 근사적인 가시화 결과를 계산해 낼 수 있다.

IV 대용량 지형 모델의 빠른 지형 그림자 맵 생성 알고리즘

본 장에서는 GPU의 효과적인 가시화 계산을 적용한 지형 그림자 생성 알고리즘을 제시한다. 지형 모델은 대부분 수 백만 개 혹은 수 천만 개가 넘는 삼각형들로 구성된 대용량 모델이기 때문에 지형의 그림자를 생성하는데 많은 시간이 소요된다. 본 논문에서는 대용량 지형 모델의 그림자 맵을 빠른 시간에 생성하기 위해 GPU의 래스터라이제이션 기능을 적용하여 이미지 프리시젼 기법으로 그림자를 생성하고자 한다.

빠른 지형 그림자 맵 생성 알고리즘은 크게 두 단계로 구성된다. 1단계에서는 광선 추적법을 적용하기 위해서 GPU가 지원하는 차폐 질의문을 사용하여 광선과 지형의 충돌 검사를 하고, 검사 결과를 토대로 그림자 유무를 결정하여 높이 맵에 대응하는 그림자 맵을 생성한다. 2단계에서는 생성된 그림자 맵을 이용하여 최종적으로 그림자가 드리워진 지형을 사실적으로 렌더링하게 된다.

4.1 지형 그림자 맵 생성

그림자 맵을 생성하기 위해서 주어진 지형 데이터, 즉 높이 맵의 각 격자

점이 그림자인지 아닌지 구별한 후 대응되는 그림자 맵의 위치에 그림자 유무를 저장해야 한다. 여기서 광원의 종류는 방향성 광원(directional light), 스폿 광원(spot light), 점 광원(point light) 중 어떤 것을 사용하던 무관하나 본 논문에서는 점 광원을 사용한다.

4.1.1 충돌 검사

먼저 높이 맵을 렌더링 한 후, 각각의 격자 점에서 광원까지 광선을 발생시킨다. 만약 광선이 지형 상의 어느 지점에 가로 막혀 광원에 도달하지 못했다면 광선을 발생시킨 격자 점은 그림자가 드리워지는 부분이라고 간주한다(그림 4-1). 광선과 지형의 충돌 검사를 위해 GPU가 지원하는 차폐 질의문을 사용한다.

차폐 질의문은 정의한 깊이 비교 연산에 따라 깊이 테스트에 통과한 픽셀의 수를 반환 값으로 돌려주기 때문에 현재 화면에 그려지는 객체가 이전에 그려진 객체에 의해 가려지는지의 여부를 알 수 있다. 만약 깊이 비교 연산이 GL_LEQUAL이고 차폐 질의문의 반환 값이 0이라면 이는 렌더링 된 객체가 차폐물(occluder)에 의해 완전히 가려진 것을 뜻한다. 즉 렌더링 된 객체의 깊이 값이 깊이 버퍼에 저장되어 있던 값보다 크기 때문에 비교 연산을 통과하지 못하여 반환된 픽셀의 수는 0이 되고, 시점에서 객체가 보이지 않는 것을 의미하는 것이다.

본 논문에서는 깊이 비교 연산을 GL_GREATER로 정의하고 지형의 깊이

값보다 광선의 깊이 값이 더 큰 픽셀 개수를 반환한다. 만약 반환 값이 1 이상이라면 광선은 지형에 의해 가려졌음을 알 수 있고 지형에 충돌했다고 볼 수 있다. 높이 맵의 모든 격자 점에서 광원까지 발생시킨 광선에 차폐 질의문을 실시하여 충돌 여부를 검사하고 그 정보를 바탕으로 그림자 유무를 정하여 높이 맵에 대응되는 그림자 맵을 생성한다(표 4-1).

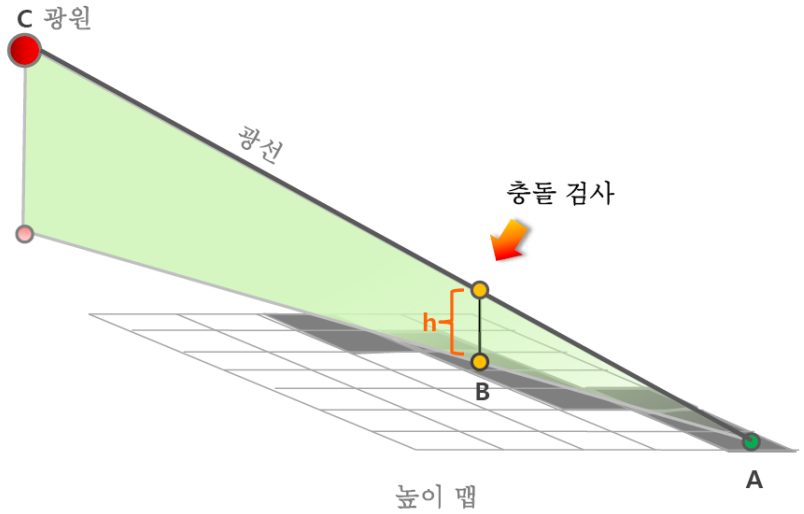


그림 4-1

광선의 충돌 검사: 격자 점 A에서 광원 C로 광선을 발생시켜 광선의 높이와 지형의 높이를 비교한다. 예를 들어 격자 점 B에 저장된 지형의 높이 값과 그 때의 광선의 높이 h를 비교하여 광선의 높이가 더 낮을 경우 지형과 충돌하였다고 본다.

```
glDepthMask(GL_FALSE);
glDepthFunc(GL_GREATER);
for (int X=0;    X < MAP_SIZE;    X += STEP_SIZE )
for (int Y=0;    Y < MAP_SIZE;    Y += STEP_SIZE )
{
                int x, y, z;
                x = X;
                y = Y;
                z = Height(pHeightMap, X, Y );
                glBeginOcclusionQueryNV(occlusionQueries[0]);
                glBegin (GL_LINES);
                  glVertex3i(x, y, z);
                  glVertex3f( lightPosition[0], lightPosition[1], lightPosition[2] );
                glEnd();
              glEndOcclusionQueryNV();
              glGetOcclusionQueryuivNV(occlusionQueries[0],
                                                GL_PIXEL_COUNT_NV, &pixelCount);
              if(pixelCount >= 1)    fprintf(opFile_shadow, "%d\n", 1);
              else if (pixelCount == 0 )    fprintf(opFile_shadow, "%d\n", 0);
}
glDepthMask(GL_TRUE);
glDepthFunc(GL_LEQUAL);
```

코드 4-1

차폐 질의문: 높이맵을 렌더링 한 후 광선 렌더링시 차폐 질의문을 사용하여 깊이 테스트에 통과한 픽셀 수을 반환한다.

4.2 지형 그림자 렌더링

생성한 지형 그림자 맵을 이용하여 최종적으로 그림자가 드리워진 지형을 렌더링한다. 그림자 맵의 격자 점이 그림자라고 판정됐다면 디퓨즈(diffuse)와 스펙큘러(specular)값에 그림자색을 저장하여 렌더링하고 그림자가 아니라면 원래의 지형 색으로 렌더링한다(수식 4-1).

$$\boldsymbol{Illumination(A)} = \boldsymbol{I_{ambient}} + \boldsymbol{I_{diffuse}} + \boldsymbol{I_{specular}}$$

$$= \boldsymbol{k_a I_a} + \boldsymbol{k_d I_l}(\mathbf{N} \cdot \mathbf{L}) + \boldsymbol{k_s I_l}(\mathbf{N} \cdot \mathbf{H})^{\boldsymbol{n_s}}$$

$\boldsymbol{k_a}$ *ambient reflection coefficient*	$\boldsymbol{I_a}$ ambient intensity
$\boldsymbol{k_d}$ *diffuse reflectivity*	$\boldsymbol{I_l}$ *light source intensity*
N *surface normal*	**L** *light source direction*
$\boldsymbol{k_s}$ *specular reflection coefficient*	**H** = (L+V)/|L+V|
V *view direction*	$\boldsymbol{n_s}$ *specular reflection exponent*

수식 4-1

3D 그래픽스 처리 과정의 지오메트리 단계를 프로그래밍하는 cg 정점 셰이더 프로그램(cg vertex program)을 사용하여 각각의 격자 점의 라이팅 효과를 다르게 주었다(표 4-4).

```
void VertexLight(float4 position : POSITION,
                 float3 normal : NORMAL,
                 out float4 oPosition : POSITION,
                 out float4 color : COLOR,
                 uniform float4x4 modelViewProj,
                 uniform float3 globalAmbient,
                 uniform float3 lightColor,
                 float3 lightPosition,
                 float3 eyePosition,
                 float3 Ka,
                 float3 Kd,
                 float 3 Ks,
                 float shininess )
{
        oPosition = mul(modelViewProj, position);
        float3 P = position.xyz;
        float3 N = normal;
        float3 ambient = Ka * globalAmbient;
        float3 L = normalize(lightPosition - P);
        float diffuseLight = max(dot(N, L), 0);
        float3 diffuse = Kd * lightColor * diffuseLight;
        float3 V = normalize(eyePosition - P);
        float3 H = normalize(L + V);
        float specularLight = pow(max(dot(N, H), 0), shininess);
        if (diffuseLight <= 0) specularLight = 0;
        float3 specular = Ks * lightColor * specularLight;
```

```
        color.xyz = ambient + diffuse + specular;
        color.w = 1;
}
```

코드 4-2

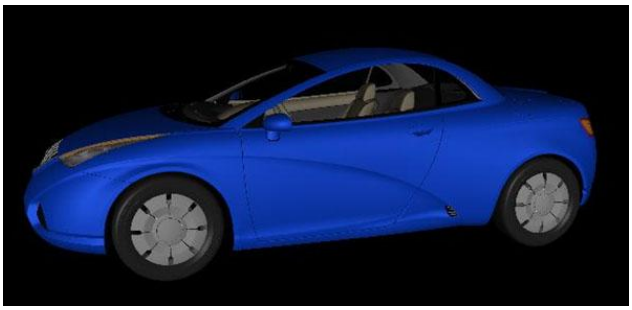
쉐이더 프로그램: 그림자 색은 디퓨즈를 나타내는 변수 Kd와 스펙큘러를 나타내는 Ks에 전달하고, 앰비언트(ambient)와 함께 최종 렌더링 색을 계산한다.

V 구현 및 결과

본 논문에서는 GPU를 이용한 효율적 가시화 계산을 전역적 가시화 컬링 알고리즘과 지형 그림자 맵 생성 알고리즘에 적용하였다. 본 장에서는 알고리즘의 구현, 벤치 마킹 모델, 알고리즘 성능 평가 그리고 결과를 보여준다.

5.1 전역적 가시화 컬링 알고리즘의 구현 및 결과

본 논문에서는 전역적 가시화 컬링 알고리즘을 구현하기 위하여 Microsoft Visual C++.Net 과 OpenGL, OpenSG 라이브러리를 사용하였으며, nVIDIA GeForce 7900 GX2 의 그래픽 카드가 장착된 Dual AMD Athlon 64 2.6GHz PC 를 사용 하였다. 본 논문에서 제시한 대용량 모델 렌더링을 위한 전역적 가시화 컬링 알고리즘의 성능 평가를 위해서 자동차와 엔진 모델을 사용하였다(그림 5-1).



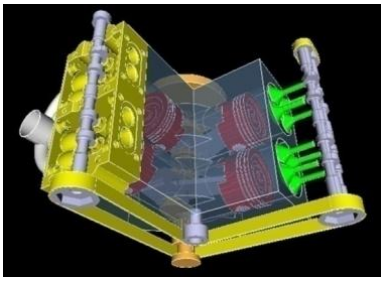


그림 5-1

벤치 마킹 모델. (위)자동차. 247 씬 그래프 노드, 3 백만 삼각형 (아래)엔진. 7177 씬 그래프 노드, 2 십 7 만 삼각형

그 결과 그림 5-2, 5-3 과 같은 결과를 도출 하였고, 이 그림에서 모델의 보이는 부분은 투명한 파란색으로 나타내었고, 보이지 않는 부분(컬링 되어야 할 부분)은 불투명한 빨간색으로 나타내었다. 자동차 모델은 247 개의 씬 그래프 노드, 약 3 백 만개의 삼각형으로 이루어져 있고 성능 평가 결과 격자 해상도가 128 일 때 씬 그래프 은닉 노드 수는 175 개이며 제거율은 71% 이다. 그리고 엔진 모델은 7177 개의 씬 그래프 노드, 약 2 십 7 만개의 삼각형으로 이루어져 있고 성능 평가 결과 격자 해상도가 64 일 때 씬 그래프 은닉 노드 수는 5915 개이며 제거율은 82%에 다다른다(표 5-1).

모델	격자 해상도	씬그래프 은닉노드수	제거율	유향거리장 계산시간	프런트 확장수행시간
자동차(247 노드, 3백만 삼각형)	128	175	71%	3.42 초	0.05 초
	256	170	69%	6.59 초	0.10 초
엔진(7177 노드, 2 십 7 만 삼각형)	64	5915	82%	1.85 초	0.05 초
	128	4611	64%	4.26 초	0.40 초

표 5-1

알고리즘의 성능 평가

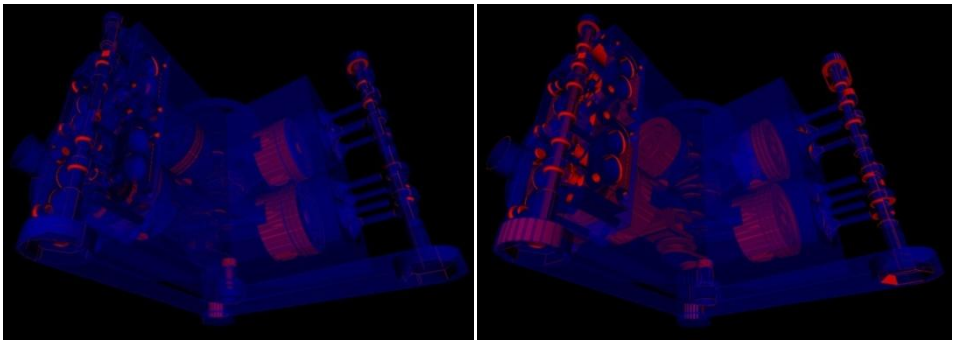


그림 5-2

엔진 모델 가시화 컬링 결과. (왼쪽) 격자 해상도 64 의 결과 (오른쪽)격자 해상도 128 의 결과

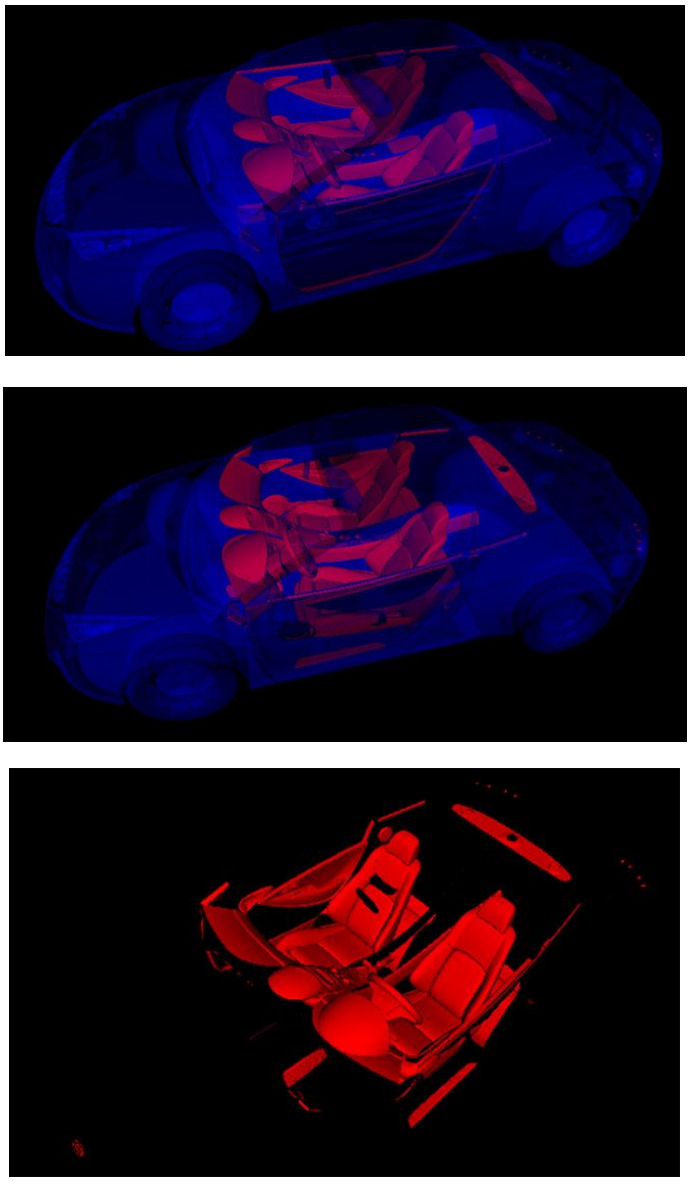


그림 5-3

자동차 모델 가시화 컬링 결과. 컬링 알고리즘을 테스트하기 위하여 차의 유리창은 불투명하여 밖에서 봤을 때 차의 안쪽은 보이지 않을 것이라고 가정하였다. (위)격자 해상도 128 의 결과 (가운데)격자 해상도 128 일 때 제거된 부분 (아래)격자 해상도 256 의 결과

5.2 빠른 지형 그림자 맵 생성 알고리즘의 구현 및 결과

지형 그림자 맵 생성 알고리즘을 구현하기 위하여 Microsoft Visual C++.Net 과 OpenGL 라이브러리, cg 정점 쉐이더 프로그램을 사용하였으며, nVIDIA GeForce 6800 의 그래픽 카드가 장착된 AMD Athlon 64 2.2GHz PC 를 사용하였다. 본 논문에서 제시한 지형 그림자 맵 생성과 렌더링의 성능 평가를 위해서 높이맵은 256×256, 512×512, 1024×1024 를 사용하였다(그림 5-4).

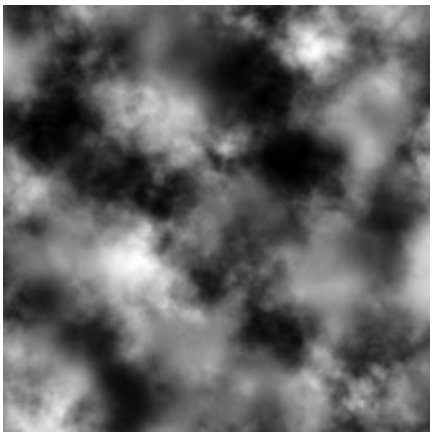


그림 5-4
지형으로 사용된 높이 맵

그 결과 그림 5-5, 5-6 과 같은 결과를 도출 하였고, 이 그림에서 높이가 낮은 부분은 어두운 녹색으로, 높은 지형으로 갈수록 밝은 녹색, 갈색, 흰색

순으로 고도를 표현했다. 성능 평가 결과 높이 맵이 256×256 일 경우 그림자 맵 생성 속도는 0.50 초, 512×512 일 경우 1.81 초, 1024×1024 일 경우 7.32 초로 기존의 방법인 CPU 를 이용하여 그림자 맵을 생성한 속도 보다 약 5 배 이상 빠른 것을 알 수 있다(표 5-2).

높이 맵	CPU 이용 그림자 맵 생성 시간	GPU 이용 그림자 맵 생성 시간
256	2.39 초	0.50 초
512	9.56 초	1.81 초
1024	38.24 초	7.32 초

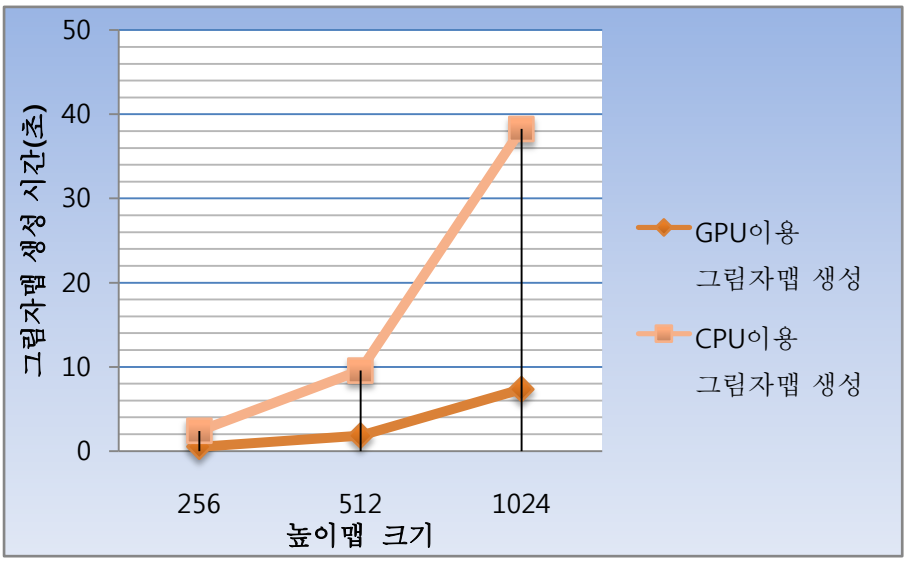


표 5-2

알고리즘의 성능 평가

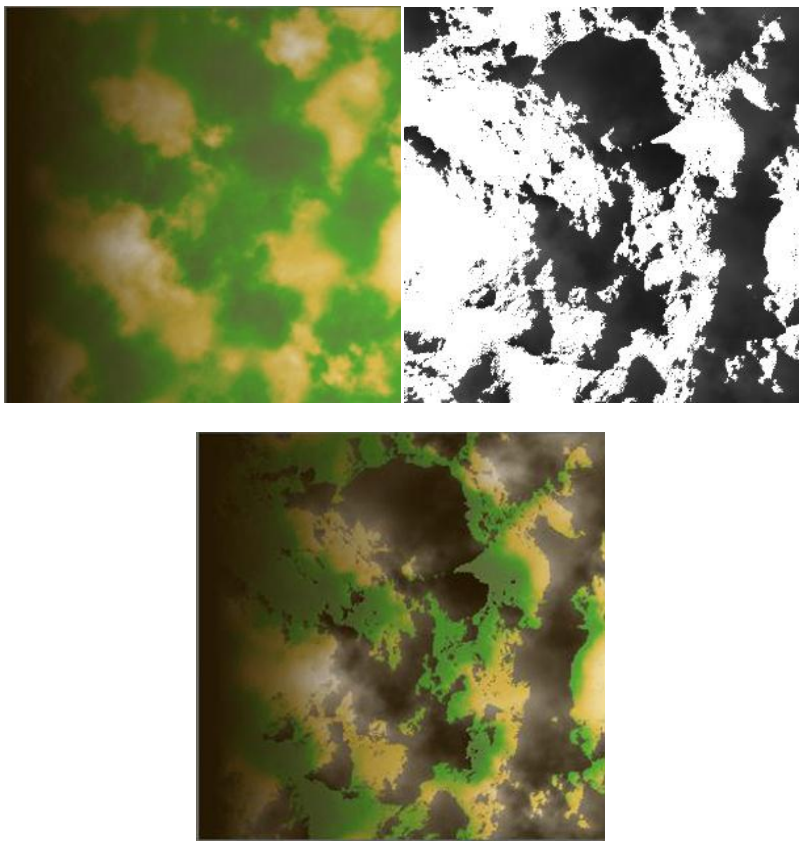


그림 5-5

지형 그림자.(왼쪽위)지형 (오른쪽위)그림자 맵 (아래)지형 그림자

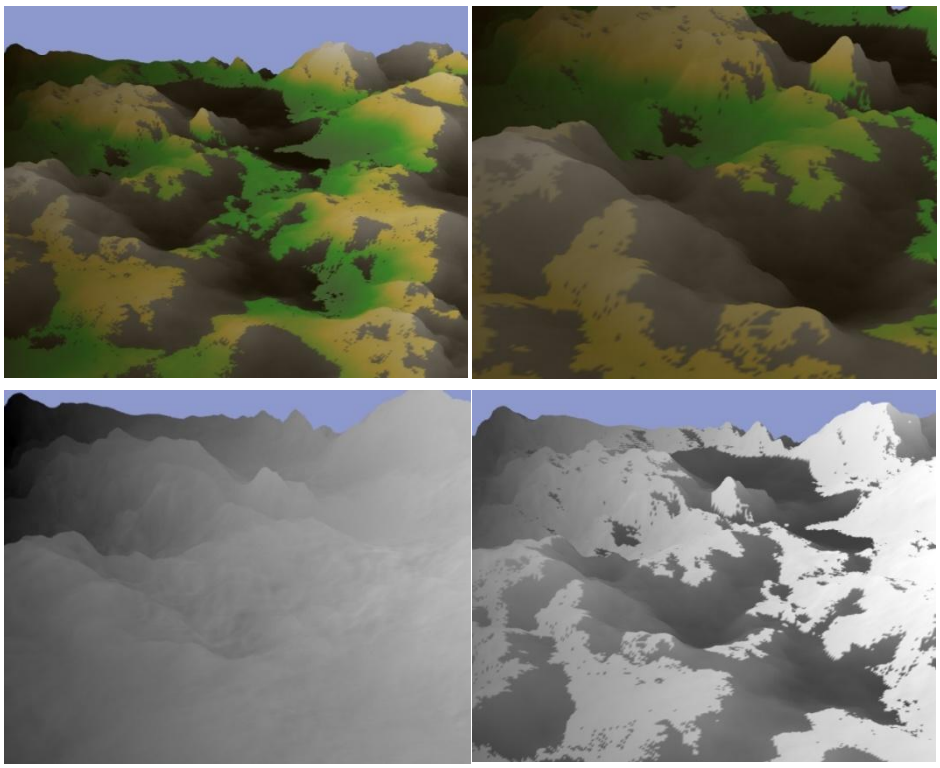


그림 5-6

지형 그림자 렌더링 결과. (왼쪽위)지형 그림자 (오른쪽위)확대한 결과 (왼쪽아래)그레이 스케일로 본 지형 (오른쪽아래)그레이 스케일의 지형 그림자

VI 결론

6.1 전역적 가시화 컬링 알고리즘

대용량 모델 렌더링을 위한 전역적 가시화 컬링 알고리즘을 통해 모델의 보이지 않는 내부 요소를 제거함으로써 렌더링 시간을 줄이고 프레임 비율을 높일 수 있었다.

향후 과제로는 유향 거리 장을 구축하는 속도를 더 빠르게 할 수 있게 가시성 순서(visibility ordering)를 적용하는 것이 될 수 있다. 가시성 순서는 주어진 시점에서 각 픽셀에 저장된 물체의 깊이 정보를 GPU의 차폐 질의 기능을 이용하여 비교 작업을 수행한 결과로 물체의 렌더링 순서를 재정리하는 방법이다. 주어진 물체의 가시성 순서를 계산하여 유향 거리 장을 구하는 속도를 빠르게 할 수 있을 것이다.

6.2 빠른 지형 그림자 맵 생성 알고리즘

빠른 지형 그림자 맵 생성 알고리즘은 광선 추적법 적용 시 GPU 가 지원하는 차폐 질의문을 사용하여 충돌 검사를 함으로서, CPU 를 이용하여

그림자 맵을 생성할 때 보다 훨씬 빠른 계산 속도로 그림자 맵을 생성하고 사실감 있는 지형 그림자를 렌더링 할 수 있었다.

향후 과제로는 뷰 절두체 컬링(view frustum culling)과의 연계, 정점 버퍼 객체(vertex buffer object)를 이용한 그림자 광선 최적화(shadow ray optimization), 그림자 광선의 멀티 레졸루션(multi-resolution)을 통해 더욱 효율적으로 그림자 맵을 생성 하는 방법을 들 수 있다.

참 고 문 헌

[1] J. M. Airey, J. H. Rohlf and Frederick P. Brooks, Jr, Towards image realism with interactive update rates in complex virtual building environments, In Computer Graphics(1990 Symposium on Interactive 3D Graphics), volume 24, pages 41-50, March, 1990.

[2] U. Assarsson and T. Moller, Optimized view frustum culling algorithms for bounding boxes, Journal of Graphics Tool, 5(1):9-22, 2000.

[3] D. Cohen-Or, Y. Chrysanthou and C. Silva, A Survey of Visibility for Walkthrough Applications, In IEEE Transaction on Visualization and Computer Graphics, 2002.

[4] F. Durand, G. Drettakis, J. Thollot and C. Puech, Conservative Visibility Preprocessing using Extended Projections, In ACM SIGGRAPH, pages 239-248, 2000.

[5] J. El-Sana, E. Azanli and A. Varshney, Integrating occlusion culling with view-dependent rendering, In IEEE Visualization, 2001.

[6] N. Govindaraju, B. Lloyd, S. Yoon, A. Sud and D. Manocha, Interactive Shadow Generation in Complex Environments, In ACM SIGGRAPH/ACM Trans. Graphics, volume 22, pages 501-510, 2003.

[7] N. Govindaraju, A. Sud, S.-E. Yoon and D. Manocha, Parallel Occlusion Culling for Interactive Walkthroughs using Multiple GPUs, Technical report, Univ. of North Carolina at Chapel Hill, 2002.

[8] N. Govindaraju, A. Sud, S.-E. Yoon and D. Manocha, Interactive visibility culling in complex environments using occlusion-switches, In ACM SIGGRAPH Symposium on Interactive 3D

Graphics, 2003.

[9] N. Greene, M. Kass and G. Miller, Hierarchical z-buffer visibility, In Proceedings of SIGGRAPH 93, pages 231-240,1993.

[10] D. Halperin, Handbook of Discrete and Computational Geometry, chapter 21, pages 389-412, CRC Press LLC, Boca Raton, FL.1997.

[11] H. Hoppe, Progressive Meshes, In Proceedings of SIGGRAPH, pages 99-108, 1996.

[12] Y. J. Kim, G. Varadhan, M. C. Lin and D. Manocha, Fast Swept Volume Approximation of Complex Polyhedral Models, In ACM Symposium on Solid Modeling and Applications, pages 11-22, Seattle, Washington, 2003.

[13] S. Kumar, D. Manocha, W. Garrett and M. Lin, Hierarchical back-face computation, Computers and Graphics, 23(5):681-692, October, 1999.

[14] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson and R. Huebner, Level of Detail for 3D Graphics, Morgan Kaufmann as part of their series in Computer Graphics and Geometric Modeling, 2003.

[15] E. Manfred, F. Firsching and R. Grosso, ENTKERNER: A System for Removal of Globally Invisible Triangles from Large Meshes, In Proceedings of 13th International Meshing Roundtable, 2004.

[16] T. Moller and E. Haines, Real-Time Rendering, chapter 7, A.K. Peters Ltd, 1999.

[17] S. Teller, Visibility Computations in Densely Occluded Environments, PhD thesis, Univ. of California, Chapel Hill, 1991.

[18] S. J. Teller and C. H. Sequin, Visibility preprocessing for interactive walkthroughs, Computer Graphics (Proceedings of SIGGRAPH 91), 25(4):61-69, July, 1991.

[19] A. Woo , P. Poulin , A. Fournier, A Survey of Shadow Algorithms, IEEE Computer Graphics and Applications, 10(6):13-32, November, 1990.

[20] L. Williams, Casting curved shadows on curved surfaces, In Computer Graphics (SIGGRAPH '78 Proceedings), volume 12, pages 270-274, August 1978.

[21] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli, Fast shadows and lighting e ects using texture mapping, In Computer Graphics (SIGGRAPH '92 Proceedings), volume 26, pages 249-252, 1992.

[22] W. Heidrich, H. P. Seidel, Realistic hardware-accelerated shading and lighting, In Proc. of ACM SIGGRAPH, 171-178, 1999.

[23] M. Stamminger, and G. Drettakis, Perspective shadow maps, In Proceedings of ACM SIGGRAPH 2002, pages 557-562, 2002.

[24] W. Reeves, D. Salesin, and R. Cook, Rendering antialiased shadows with depth maps, In Computer Graphics (ACM SIGGRAPH '87 proceedings), volume 21, pages 283-291, 1987.

[25] S. Brabec, T. Annen, and H. Seidel, Hardware-accelerated rendering of antialiased shadows, In Proc. of Computer Graphics International, 2001.

[26] R. Fernando, S. Fernandez, K. Bala, and D. Greenberg, Adaptive shadow maps, In Proceedings of ACM SIGGRAPH 2001, pages 387-390, 2001.

[27] K. Tadamura, X. Qin, G. Jiao, and E. Nakamae, Rendering optimal solar shadows with plural sunlight depth buffers, The Visual Computer 17(2), 2001.

[28] I. Wald, P. Slusallek, and C. Benthin, Interactive distributed ray-tracing of highly complex models, In Rendering Techniques, pages 274-285, 2001.

[29] S. Parker, W. Martic, P. Sloan, P. Shirley, B. Smits, and C. Hansen, Interactive ray tracing, Symposium on Interactive 3D Graphics,

1999.

[30] J. Blinn, Jim blinn's corner: Me and my (fake) shadow, IEEE Computer Graphics and Applications 8(1):82-86, January, 1988.

[31] F. Crow, Shadow algorithms for computer graphics, volume 11, pages 242-248, 1977.

[32] N. Chin, and S. Feiner, Near real-time shadow generation using BSP trees, In Computer Graphics (SIGGRAPH '89 Proceedings), volume. 23, pages 99-106, 1989.

[33] Y. Chrysanthou, and M. Slater, Shadow volume BSP trees for computation of shadows in dynamic scenes, In 1995 Symposium on Interactive 3D Graphics, pages 45-50, 1995.

[34] T. Heidmann, Real shadows real time, IRIS Universal volume 18, 1991.

[35] W.R. Mark, R.S. Glanville, K. Akeley, and M.J. Kilgard, Cg: A System for Programming GraphicsHardware in a C-like Language, ACM Transactions on Graphics, 22(3):896-907, July, 2003.

[36] Microsoft HLSL, http://msdn2.microsoft.com/en-us/library/bb509638.aspx

[37] The OpenGL Shading Language, http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf

[38] V. Moya, C. Gonzalez, J. Roca, A. Fernandez, and R. Espasa, Shader Performance Analysis on a Modern GPU Architecture, Microarchitecture, MICRO-38. Proc. 38th Annual IEEE/ACM Int'l Symp., 2005.

[39] M. J. Kilgard, Interactive Geometric Computations Using Graphics Hardware, SIGGRAPH '02 Course 31, 2002

[40] M. Marghidanu, Fast computation of terrain shadow maps, http://www.nervus.org, 2000

[41] D. Koller and P. Lindstrom, Real-Time, Continuous LOD Rendering of Height Fields, Siggraph 1996, pages 109-118, August, 1996.

ABSTRACT

This dissertation addresses how one can exploit the fast rasterization and visibility functions available in graphics hardware (GPU) for efficient visibility culling for large CAD model and shadow generation for a large terrain model.

The goal of our visibility culling is to remove the invisible triangles of a CAD model, consisting of millions of triangles, when the camera is located outside of the given model. We pose this problem as arrangement computation, and the arrangement of the given model is approximated by generating the directed distance field of the given model by using GPU and finding the outer envelope by fast marching method. From our experimentation, we could cull away more than 70% of hidden geometry by using our algorithm, taking 6.3 seconds on nVIDIA GeForce 7900GX2 for a large CAD model consisting of three millions of triangles.

Furthermore, we also compute shadow map for a large terrain model by combining ray tracing with GPU-supported occlusion query. The shadow map generation took 7.31 seconds for 1024^2 height map on nVIDIA GeForce 6800.

감사의 글

부족한 제게 항상 많은 가르침과 깨우침을 주시고 때론 꾸지람과 격려로 저를 더 강한 사람으로 만들어주신 김영준 교수님께 감사 드리고, 언제나 올곧으신 교수님을 존경합니다.

그리고 이화에서 공부하며 제가 더 성장할 수 있게 기회와 가르침을 주신 용환승 교수님, 채기준 교수님, 최병주 교수님, 김명 교수님, 김명희 교수님, 이미정 교수님, 이상호 교수님, 박승수 교수님, 조동섭 교수님, 반효경 교수님, 이민수 교수님, 박현석 교수님, 이정원 교수님께 감사 드립니다.

같이 공부하며 울고 웃었던 정든 연구실 식구들, 수정, 민경, 연희, 성실한 모습으로 귀감을 주시는 장박사님, 고민해결사 고마운 미경언니 모두 감사 드립니다.

제게 많은 경험과 공부를 할 수 있게 기회를 주신 Virtual Builders의 최진원 교수님, 김영 차장님 그리고 따뜻한 VB 식구들에게도 감사 드립니다.

같이 대학원 생활을 하며 웃으며 인사하는 것만으로도 힘이 되었던 미경, 재원 그리고 옆방 소윤언니, 진영, VR랩의 명보, 성희 그리고 같이 공부했던 모든 분들에게 고맙다고 말하고 싶습니다.

멀리서 저를 위해 늘 기도해주는 친구 정은, 자주는 못 보지만 편한 수다 친구 혜진, 착하고 재미있는 동생들 혜진이와 나라에게도 고마움을 전합니다.

언제나 든든한 저의 버팀목 재홍오빠, 고맙습니다.

그리고 딸내미 공부한다고 매일같이 응원해주시고 힘이 되어주시고 기도해주시는 부모님, 너무나 감사 드립니다. 사랑해요... 더운 나라에서 고생하고 있는 오빠에게도 고맙고, 건강하길 기도합니다.

저를 위해 늘 기도해주시는 제 주변의 모든 분들께 감사 드립니다. 앞으로 더 크게 성장하기 위해 열심히 노력하며 살겠습니다. 감사합니다.