영상기반 렌더링 기법을 이용한 실시간 그림자 생성 기법

이중연 임인성
서강대학교 컴퓨터학과 컴퓨터그래픽스연구실

Real-Time Shadow Generation Using Image-Based Rendering Technique

Joong-Youn Lee Insung Ihm
Computer Graphics Lab., Dept. of Computer Science, Sogang Univ.

요 약

3D 컴퓨터 그래픽스에서 그림자는 매우 중요한 요소이다. 그림자가 없으면 물체와 물체간의 정확한 위치를 파악하기가 어렵기 때문이다. 즉 물체가 다른 물체에 바로 붙어있는지 조금 떨어져 있는지를 알아내기가 곤란한 것이다. 이러한 그림자의 중요성에도 불구하고 현존하는 3D 가속 하드웨어들은 그림자의 생성을 전혀 지원하지 못하고 있다. 또한, 이것을 보완하기 위해 많은 그림자 생성 알고리즘들이 제시되었지만 모두 그 한계가 명확하였다. 많은 그림자 생성 기법 중 텍스춰 하드웨어를 이용한 그림자 지도 기법은 광원과 물체가 고정되어 있을 경우 매우 빠른 속도로 그림자를 생성시킬 수 있지만 광원이나 물체가 움직일 경우에는 속도가 느려지게 된다. 본 논문에서는 그림자 지도 기법에 영상기반 렌더링 기법을 적용하여 실시간으로 그림자를 생성하고자 하였다. 본 논문의 기법은 매우 많은 메모리를 필요로 하기 때문에 데이터를 적당히 샘플링하여 웨이블릿 기반 압축 기법으로 압축하였고, 그림자를 생성할 때는 가장 가까운 저장된 그림자들을 2차원 보간하였다.

1. 서론

3D 가속 그래픽스 카드를 이용하여 실시간으로 렌더링하는 경우에는 하드웨어의 한계로 그림자나 반사, 굴절 효과와 같은 실세계의 현상들은 제대로 표현하지 못하였다. 이에 따라 소프트웨어적으로 이러한 현상들을 처리해주는 알고리즘들이 꾸준히 연구되어 지고 있다. 본 논문에서는 이러한 현상들 중 그림자 효과에 대해서 영상기반 렌더링(image-based rendering) 기법과 3D 가속 하드웨어의 텍스춰 맵핑(texture mapping) 기법을 이용하여 비교적 정확한 그림자를 실시간으로 생성하고자 하였다. 본 논문의 기법은 [1]의 텍스춰 맵핑 하드웨어를 이용한 그림자 생성 방법에 기반하고 있다. 기존 방법의 문제점인 전경의 물체나 광원이 움직이는 경우에 그림자의 생성 속도가 늦어지는 점을 보완하기 위해 영상기반 렌더링 기법을 적용하였다. 또한, 이에 따른 메모리의 낭비는 [12]에서 제안한 웨이블릿(wavelet) 기반 3차원 압축기법을 이용하여 해결하였다.

2. 관련 연구

그림자를 생성하는 기법은 크게 3가지로 분류할 수 있다[3,4,9]. 그림자의 투영(projection), 그림자 볼륨(shadow volume), 그림자 지도(shadow map) 기법이 그것이다. 투영 기법은 빠르게 그림자를 생성할 때 보통 많이 쓰는 기법이다. 그러나 이 기법은 평평하지 않은 표면에 적용시킬 수 없을 뿐만 아니라 물체의 복잡도에 따라 그림자의 생성 속도가 달라지는 단점이 존재한다. 그림자 볼륨 기법은 3D 가속 하드웨어의 스텐실 버퍼(stencil buffer)를 이용하면 매우 빠른 속도로 그림자를 생성할 수 있다[3,6,9,10]. 그러나 이 방법 역시 물체의 복잡도에 따라 그림자의 생성 속도가 달라지게 된다.

그림자 지도 기법을 이용하면 텍스춰 맵핑 하드웨어를 이용하여 빠르게 그림자를 생성시킬 수 있는데[1], 본 논문에서 제안하는 기법은 이 방법에 기반하였다. Segal은 기존의 그림자 지도 기법에서의 Z-버퍼에 저장된 깊이 정보를 텍스춰에 저장한 뒤 전경에 투영하여 빠르게 그림자를 생성하였다. 이러한 일련의 과정들은 OpenGL을 이용해서 구현이 가능하기 때문에 OpenGL 가속을 지원하는 하드웨어를 이용할 경우 매우 빠른 속도로 그림자를 표현할 수 있다. 이 기법은 전경에 텍스춰를 붙이는 작업만으로 그림자의 생성이 가능하여 기존의 어떠한 그림자 생성 알고리즘보다도 빠른 시간 안에 그림자를 생성시킬 수 있게 된다.

3. 향상된 그림자 지도 기법

3.1. 기존 기법의 문제점

[1,9]에서 제안된 텍스춰 하드웨어를 이용한 그림자 생성 기법은 물체와 광원이 고정된 경우에는 상수 시간에 그림자를 생성할 수 있지만 물체나 광원이 움직일 경우 그림자 지도의 재계산이 필요하여 속도가 현저히 저하되는 문제점을 가지고 있다. 물론 이것은 대부분의 그림자 생성 기법이 공통적으로 안고 있는 문제이기도 하다. 이러한 문제점을 극복하기 위해 본 논문에서는 영상기반 렌더링 기법을 응용하여 기존의 그림자 지도 기법을 확장시키고자 하였다.

3.2. 그림자 지도 기법의 확장

텍스춰 하드웨어를 이용한 그림자 지도 기법에서 그림자를 생성하는 데 걸리는 시간은 대부분 광원에서 전경을 렌더링하여 그림자 지도를 생성하는 데 할애된다. 따라서, 영상기반 렌더링 기법에서와 같이 위치할 수 있는 모든 광원과 물체에서 미리 그림자 지도를 생성하여 저장하였다가 텍스춰 하드웨어를

이용하여 그림자를 생성할 때 필요한 그림자 지도를 읽어 들여 사용하면 실시간으로 그림자를 생성할 수 있게 된다.

보통 그림자 이미지는 RGBA의 색상 요소를 모두 가질 필요가 없이 단색의 요소만 가지면 되므로 텍스춰 형태로 저장할 때 그 크기가 비교적 적다. 한 픽셀이 8비트로 구성된 256×256 크기의 그림자 텍스춰의 경우 약 64K 바이트의 크기를 가지게 된다. 또한 광원과 대부분의 물체는 고정되어 있고 특정한 몇 물체만 움직이는 경우, 움직이는 몇 물체에 대해서 각 위치에서의 그림자를 미리 생성하여 텍스춰로 저장해 놓고 실제 렌더링 시 읽어와서 투영 텍스춰(pojective texture) 기법[9]으로 그림자를 생성한다면 실시간으로 그림자를 생성할 수 있다. 이 경우 영상기반 렌더링의 경우와 마찬가지로 물체의 복잡도와 관계없이 빠른 시간 안에 그림자를 생성할 수 있다.

3.2.1. 저장된 그림자의 선택 방법

저장된 그림자를 선택하는 방법은 편의상 2가지 경우로 나누었다. 광원이 고정되어 있는 경우와 광원이 고정되어 있지 않은 경우이다. 광원이 고정되어 있다고 가정하고 물체만 움직인다면 미리 생성된 그림자 텍스춰들 중 물체의 현재 위치에 맞는 그림자 텍스춰를 선택하는 것은 매우 간단하다. 물체의 현재 위치 좌표를 얻어 낸 뒤, 샘플링된 좌표들 중 가장 가까운 좌표에 위치한 그림자를 읽어와서 그림자 텍스춰로 사용하면 된다. 이 경우, 약간의 오차는 있겠지만 가장 가까운 그림자를 사용하였기 때문에 어느 정도 비슷한 모양으로 그림자가 생성되게 된다. 그림자 텍스춰들을 2차원 배열에 저장할 경우, 그림자 텍스춰 자체도 2차원 배열이므로 전체적으로는 4차원 배열이 된다. 이렇게 구현할 경우 샘플링한 위치들과 2차원 배열이 1:1 대응이 되므로 상수 시간 안에 어느 그림자를 읽어야 하는지 알 수 있다.

물체가 고정되어 있고 광원이 움직일 경우에는 물체의 위치에 따른 그림자 텍스춰들을 이용해서 광원이 움직였을 때 변화된 그림자를 생성할 수 있다. 그림 3.1의 왼쪽 그림은 광원이 고정되어 있고 물체가 오른쪽으로 움직이는 경우의 그림자의 변화이고, 오른쪽 그림은 물체가 고정되어 있고 광원이 왼쪽으로 움직이는 경우의 그림자의 변화이다. 왼쪽과 오른쪽 그림의 그림자가 모두 똑같은 것을 쉽게 알 수 있다. 이와 같이 광원이 움직이는 경우에는 미리 저장해 놓은 그림자 텍스춰에서 광원이 움직이는 방향과 반대쪽 방향으로 이동한 경우의 그림자 텍스춰를 사용하면 되므로 큰 문제가 되지 않는다.

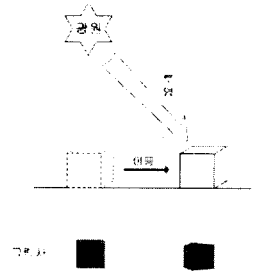

그림 3.1. 물체가 이동하는 경우와 광원이 움직이는 경우의 그림자의 변화

3.2.2. 보간(interpolation) 방법

영상기반 렌더링에 기초한 확장된 그림자 텍스춰 기법은 속도는 굉장히 빠르지만 물체가 위치할 수 있는 모든 위치에서의 그림자를 미리 생성하여 저장해야 하기 때문에 메모리를 매우 많이 필요로 한다. 따라서, 물체의 위치들을 적당히 샘플링하여 저장한 뒤, 실제 렌더링 때에는 물체의 현재 위치와 가장 가까운 4개의 그림자 지도를 2차원 보간(bilinear interpolation)하여 사용하면 메모리를 절약할 수 있다. 보간을 수행할 때에는 가장 가까운 그림자 텍스춰들 한꺼번에 4개씩 읽고 이 네 그림자 텍스춰들을 2차원 보간하면 된다.

2차원 보간을 한번 수행하기 위해서는 모두 3번의 선형 보간(linear interpolation)을 해야 한다. 따라서, 그림자 텍스춰의 모든 픽셀에 대해서 보간 작업을 하는 경우 모두 256×256×3번 즉, 196608번 선형 보간을 해야 한다. 이 작업은 196608번의 곱셈, 393216번의 덧셈 연산을 해야 하므로 그림자의 생성 속도가 매우 느려지게 된다. 이에 따라 본 논문에서는 그래픽스 하드웨어의 알파 블렌딩 기능을 이용해서 2차원 보간 작업을 빠르게 하고자 하였다. 두 그림자 텍스춰 중 뒤의 그림자 텍스춰를 백 버퍼(back buffer)에 그리고 일정한 블렌드 색(constant blend color)을 설정한 후 블렌딩을 가능하게 하여 그리면 백 버퍼에 선형 보간을 한 효과가 저장되게 된다[1]. 백 버퍼에 저장된 값을 임의의 배열에 읽어 들여서 비트맵으로 만든 뒤, 버퍼를 스왑하지 않고 원래 그림을 다시 그리면 화면에는 아무 그림도 나타나지 않고 하드웨어적으로 선형 보간을 할 수 있게 된다. 이 방법은 최근에 출시된 대부분의 그래픽스 하드웨어가 가지고 있는 알파 버퍼를 이용하여 보간 작업을 가속시킬 수 있다.

4. 웨이블릿(wavelet) 기반 압축

본 논문의 방법과 같이 모든 그림자 텍스춰들을 저장하기 위해서 많은 메모리를 사용해야 하므로 데이터의 압축이 필요해지게 된다. 본 논문에서 그림자를 생성하는 기법은 미리 생성된 그림자 텍스춰들을 압축할 경우, 높은 압축률을 가지고 빠른 복원이 가능해야 하기 때문에 이러한 특성을 지닌 제로비트 인코딩 기법을 이용하였다[12].

물체의 위치에 따라서 미리 생성해 놓은 그림자 텍스춰들은 4차원 데이터에 해당한다. 그러나 제로비트 인코딩 스킴은 3차원이기 때문에 이에 맞추기 위해 이 데이터를 다시 3차원 데이터로 변환시켜야 한다. 가장 간단하게 변환시키는 방법은 각 그림자 텍스춰들을 한 방향으로 쌓는 것이다. 이 경우, 응집성을 최대한 살리지 못할 뿐만 아니라 불필요한 계산이 많아지게 된다.

본 논문에서는 그림자 텍스춰들의 응집성을 최대한 살리기 위해서 인접한 4개의 그림자 텍스춰들을 재배열하여 압축한다. 즉, 4개의 그림자 텍스춰들을 4×4의 크기로 잘라서 4×4×4의 셀을 만든다. 이러한 셀을 4×4×4 (= 64)개를 모아서 단위 블럭을 만든 뒤 압축하면 상하, 좌우의 응집성을 모두 이용할 수 있어 압축률이 높아지고 하나의 그림자 텍스춰를 얻기 위해서 4개의 그림자 텍스춰의 압축만을 풀면 되므로 불필요한 계산도 줄일 수 있다. 또한, 앞장에서 살펴보았듯이 그림자 텍스춰들을 물체가 이동할 수 있는 모든 위치에서 생성하여 저장하지 않고 일정한 간격으로 샘플링한 뒤 2차원 보간을 하므로 현재 위치와 가장 가까운 4개의 그림자 텍스춰를 필요로 하게 된다. 이 압축 스킴에서는 무조건 4개의 그림자 텍스춰를 한번에 풀게 되므로 본 논문의 기법에 적합하다는 것을 알 수 있다. 물론, 필요한 4개의 텍스춰가 서로 다른 단위 블럭으로 압축되어 있는 경우에는 총 4번 압축을 풀어야 하게 된다.

5. 실험 및 결과

본 논문에서 제시한 영상기반 렌더링 기법을 이용한 그림자 지도 기법은 Intel Pentium II 450MHz의 CPU와 364MB의 메인 메모리, 128MB의 그래픽스 메모리를 가지고 있는 SGI 320 Visual Workstation에서 구현되었다. 사용된 컴파일러는 Microsoft Visual C++ 6.0이고 OpenGL 라이브러리를 사용하였다.

렌더링에는 1152개의 다각형을 가진 주전자, 1464개의 다각형을 가진 의자, 5804개의 다각형을 가진 소, 12078개의 다각형을 가진 드래곤 등의 데이터가 사용되었고, 렌더링 속도와 화질에 영향을 미칠 수 있는 여러 요소들을

1) **일정한 블렌드 색(constant blend color)** : OpenGL 1.1에서는 EXT_blend_color 확장을 지원하는 그래픽스 하드웨어에서만 가능하다.

바꾸어 가면서 실험을 하였다. 이러한 요소에는 그림자 텍스춰의 크기, 압축 정도, 캐쉬의 크기 등이 있다.

각 전경을 렌더링한 속도는 표 5.1과 같다. 여기서, 모든 단위는 ms(밀리초)이다. 표 5.1을 보면 그림자 텍스춰를 모두 메모리에 올린 뒤 렌더링하는 경우에는 기존의 방법보다 최대 93.3배 빨라짐을 알 수 있다. S/W와 H/W 보간의 경우에는 한번에 4개의 그림자 텍스춰를 읽어와서 2차원 보간을 한 것인데, 보간을 하지 않고 가장 가까운 것을 읽어 들이는 경우보다는 느리지만, 기존의 방법보다는 최대 18.6배까지 빨라진다.

데이터	그림자 없이	기존 방법	모두 로드	S/W 보간	H/W 보간
주전자	5.2	213.3	5.75	52.5	27.45
의자	5.53	227.1	5.9	52.65	29.25
소	16.65	239.8	19	65.95	40.75
드래곤	39.1	270	39.34	85.21	60.54

표 5.1. 새로운 그림자 기법의 수행 속도

그림자 텍스춰의 크기는 그림자를 생성하는 속도와 화질을 결정하는 중요한 요소이다. 그림자 텍스춰의 크기가 작으면 작을 수록 그림자를 생성하는 속도가 빨라지지만 그림자의 화질은 떨어지게 되고, 텍스춰의 크기가 커지면 커질수록 속도는 느려지지만 보다 나은 화질을 가지게 된다. 각 그림자 텍스춰의 크기에 따른 속도는 표 5.2에서 볼 수 있다. 표 5.2는 5804개의 다각형을 가진 소를 이용해서 측정한 것이다.

텍스춰 크기	기존 방법	모두 로드	S/W 보간	H/W 보간
64×64	111.2	19	20.1	20.9
128×128	178.24	19	32.25	25.34
256×256	239.81	19	65.95	40.75
512×512	898.21	44.24	248.9	104

표 5.2 그림자 텍스춰의 크기에 따른 그림자 생성 속도

본 논문의 방법은 메모리를 너무 많이 필요로 하기 때문에 데이터의 압축은 꼭 필요한 사항이다. 그림자 텍스춰 하나가 256×256 크기이고, 전체 전경에서 32×32만큼 샘플링을 하였을 경우 전체 그림자 텍스춰의 크기는 64MB에 달하게 된다. 본 논문에서 적용한 웨이블릿 기반 압축기법인 제로비트 인코딩은 데이터의 대부분 값이 0인 불륨 데이터를 기본으로 하여 고안되었기 때문에 데이터에 0이 많아야 압축률이 높아지게 된다. 그러나 본 논문에서 압축할 대상인 그림자 데이터는 데이터의 대부분이 255 값을 가지기 때문에 이 압축기법을 적용하기에는 적합하지 않다고 할 수 있다. 본 논문에서는 8비트에 저장할 수 있는 최대 값인 255에 그림자 데이터의 값을 뺀 값을 압축하게 하여 압축률을 높였다.

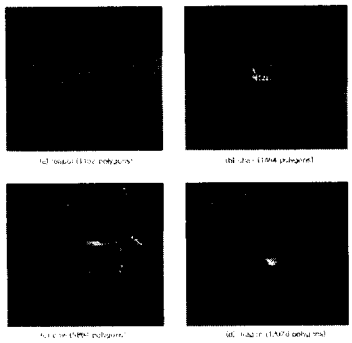

그림 5.1. 각 전경의 모습

6. 결론

3차원 컴퓨터 그래픽스에서 그림자는 매우 중요한 요소이다. 그러나 현존하는 대부분의 그래픽스 가속 하드웨어에서는 그림자를 전혀 지원하지 못한다. 따라서 이를 보완하는 그림자 생성 알고리즘들이 많이 제시되었지만 모두 그 한계가 명확하였다. 본 논문에서는 영상기반 렌더링 기법을 이용한 그림자 지도 기법을 제안하였다. 텍스춰 맵핑 그래픽스 하드웨어를 이용하여 그림자의 생성을 가속시켰고, 메모리의 낭비를 막기 위해서 웨이블릿 기반의 압축 기법을 사용하였다. 기존의 그림자 지도 기법은 매 프레임마다 광원에서부터 물체를 렌더링한 뒤 Z-버퍼의 깊이 값을 이용하여 그림자를 생성해 내는 것이었지만, 새로운 그림자 생성 기법에서는 깊이 값을 미리 생성하여 텍스춰로 저장해 놓고 웨이블릿으로 압축을 한 뒤 필요할 때 빠르게 복원하여 투영된 텍스춰 기법으로 그림자를 생성해 내었다. 또한, OpenGL SGIX_shadow 확장을 지원하는 텍스춰 맵핑 하드웨어를 가지고 있을 경우 텍스춰에 깊이 정보를 저장하여 그림자를 생성해 낼 수 있지만, 보통의 그래픽스 하드웨어에서는 이러한 기능이 없기 때문에 깊이 정보를 저장하지 않고 그림자의 모양을 그대로 저장하였다가 그림자가 생성되는 것처럼 흉내내게 하였다. 영상기반 렌더링 기법을 그림자 지도 기법에 접목시켰을 때, 얻을 수 있는 최대의 장점은 속도의 향상이었다. 또한, 전경의 복잡도에 관계없이 일정 속도로 그림자를 생성해 낼 수 있었다. 웨이블릿 기반 압축기법으로 그림자 텍스춰들을 압축할 때에는 각 그림자 텍스춰들의 응집성과 빠른 복원 속도를 모두 충족시킬 수 있는 새로운 압축 스킴을 이용하였다. 그러나 본 논문에서 제시한 방법으로는 물체가 물체 좌표계 안에서 회전 등의 변환을 할 경우에는 그림자를 다시 생성해 주어야 하는 문제점이 존재한다. 이것은 물체의 자체 회전 등도 고려하여 그림자 텍스춰를 5차원, 혹은 6차원으로 생성할 경우 빠른 그림자의 생성이 가능해 지게 된다.

참고문헌

[1] M. Segal, C. Korobkin, R. Widenfelt, J. Foran and P. Haeberli, "Fast Shadows and Lighting Effects using Texture Mapping", Proceedings of the 1992 SIGGRAPH, pp. 249-252, 1992.
[2] H. Zhang, "Forward Shadow Mapping", Proceedings of the 1998 Eurographics Workshop on Rendering, 1998.
[3] H. Batagelo, I. Junior, "Real-Time Shadow Generation using BSP Trees and Stencil Buffers", Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing, 1998.
[4] A. Woo, P. Poulin, A. Fournier, "A Survey of Shadow Algorithms", IEEE Computer Graphics and Applications, Vol. 10, Num. 6, pp. 13-32, 1990.
[5] L. Williams, "Casting Curved Shadows on Curved Surfaces", Proceedings of the 1978 SIGGRAPH, pp. 270-274, 1978.
[6] T. Heidmann, "Real Shadows, Real Time", Iris Universe, Vol. 18, pages 28-31, 1991.
[7] T. Tessman, "Casting Shadows on Flat Surfaces", Iris Universe, Vol. 16, 1989.
[8] J. Neider, T. Davis, M. Woo, "OpenGL Programming Guide, 2nd Edition", Addison-Wesley, Reading MA, 1993.
[9] T. McReynolds, "Advanced Graphics Programming Techniques using OpenGL", SIGGRAPH 1998 Course notes, 1998.
[10] M. Kilgard, "OpenGL-based Real-Time Shadows", Silicon Graphics Inc., (http://reality.sgi.com/mjk_asd/tips/rts/).
[11] E. Stollnitz, T. DeRose and D. Salesin, "Wavelets for Computer Graphics: A Primer, Part 1.", IEEE Computer Graphics and Applications, Vol. 15, Num. 3, pp. 76-84, 1995.
[12] I. Ihm and S. Park "Wavelet-based 3D Compression Scheme for Very Large Volume Data", Proceedings of Graphics Interface '98, pp. 107-116, Vancouver, Canada, June 1998.
[13] S. Park, G. Koo, and I. Ihm, "Wavelet-based 3D Compression Schemes for the Visible Human Dataset and Their Applications", CD-ROM Proceedings of the 2nd Visible Human Project Conference, 1998.