GPU를 이용한 깊이 영상기반 렌더링의 가속
(Accelerating Depth Image-Based Rendering Using GPU)

이 만 희 † 박 인 규 ††
(Man Hee Lee) (In Kyu Park)

요 약 본 논문에서는 깊이 영상기반의 3차원 그래픽 객체에 대하여 그래픽 처리 장치(Graphics Processing Unit, GPU)의 가속을 이용한 고속의 렌더링 기법을 제안한다. 제안하는 알고리즘은 최근의 그래픽 처리 장치의 새로운 특징과 프로그래밍이 가능한 쉐이더 기법을 이용하여, 속도가 느리거나 정적인 조명과 같은 기존의 일반적인 깊이 영상기반 렌더링 방법이 갖고 있는 단점을 극복할 수 있다. 깊이 영상기반 데이타의 3차원 변환 및 조명에 의한 효과 연산은 정점 쉐이더(vertex shader)에서 수행을 하고, 점 데이타의 적응적인 스플래팅(splatting)은 화소 쉐이더(fragment shader)에서 수행된다. 모의 실험결과, 소프트웨어 렌더링 또는 OpenGL 기반의 렌더링과 비교해서 괄목할 만한 렌더링 속도의 향상이 이루어졌다.
키워드 : 깊이 영상 기반, 3차원 그래픽 객체, 그래픽 처리 장치, 정점 쉐이더, 화소 쉐이더

Abstract In this paper, we propose a practical method for hardware-accelerated rendering of the depth image-based representation (DIBR) of 3D graphic object using graphic processing unit (GPU). The proposed method overcomes the drawbacks of the conventional rendering, i.e. it is slow since it is hardly assisted by graphics hardware and surface lighting is static. Utilizing the new features of modern GPU and programmable shader support, we develop an efficient hardware-accelerating rendering algorithm of depth image-based 3D object. Surface rendering in response of varying illumination is performed inside the vertex shader while adaptive point splatting is performed inside the fragment shader. Experimental results show that the rendering speed increases considerably compared with the software-based rendering and the conventional OpenGL-based rendering method.
Key words : depth image-based, 3D graphic object, graphic processing unit (GPU), vertex shader, pixel shader

1. 서 론

영상기반 렌더링은 소수의 실사 기준 영상을 이용하여 복잡한 3D 모델을 사실적으로 묘사할 수 있기 때문에 컴퓨터 그래픽스와 비전 분야의 연구자들로부터 넓은 관심을 받아왔다. 카메라의 시점이 변화되었을 때, 새로운 시점에서 바라보는 영상은 실제 3차원의 모습을 복원할 필요 없이 기준영상의 카메라 정보를 이용하여 기준 영상들을 합성함으로써 얻을 수 있다[1]. 이러한 특징으로 인해 영상기반 렌더링은 일반적인 다각형 기반의 모델링과 비교하여 사실적인 3차원 모델을 표현하기에 단순하고 효과적인 방법으로 알려져 있다.

그러나, 영상기반 렌더링에도 몇 가지 단점이 존재한다. 그 중의 하나가 바로 하드웨어의 가속을 받기 어렵다는 것이다. 영상기반 렌더링은 다각형의 내부를 가로방향으로 채우는 일반적인 렌더링 방식을 사용하지 않기 때문에, 점 스플래팅(point splatting)에서의 스플랫 크기 조절 및 구멍 메우기(hole filling)등과 같은 기본적인 렌더링은 소프트웨어의 처리과정에서 수행되어야 한다. 영상기반 렌더링에서 점의 크기를 최적화 하는 것과 스플랫을 렌더링 하는 것은 일반적인 점 기반 렌더링(point-based rendering)에서와 마찬가지로 가장 공통적이고 기본적인 동작이다[2]. 또한, 영상기반 렌더링의 다른 단점은 조명이 정적이라는 것이다. 이로 인해 시점이 변하였을 경우, 표면의 조명효과나 반사광, 그림자 등이 변하지 않기 때문에 새로운 시점에서의 장면은 부자연스럽게 보인다.

반면에, 최근 그래픽 가속기(Graphics Processing Unit, GPU)의 장점으로 두 가지의 발전방향이 보여지고

· 이 논문은 2004년도 인하대학교의 지원에 의하여 연구되었음(INHA-31451)
† 학생회원 : 인하대학교 정보통신공학과
maninara@hotmail.com
†† 정 회 원 : 인하대학교 정보통신공학부 교수
pik@inha.ac.kr
논문접수 : 2006년 6월 1일
심사완료 : 2006년 8월 21일

있다. 그 중 하나는 프로세서의 처리능력이 급격히 증가하고 있다는 것이다. 3GHz의 Pentium 4 CPU가 단지 6 Giga FLOPS로 동작하는 것에 비하여 GeForce FX 5900의 GPU는 20 Giga FLOPS의 속도로 연산을 수행할 수 있다. 최근 발전 추세의 다른 하나는 GPU 내부 파이프라인의 기능을 사용자가 프로그래밍 할 수 있도록 허용한다는 것이다. 이러한 방법은 정점 쉐이더(vertex shader)와 화소 쉐이더(fragment shader)등의 기법으로 알려져 있다. 예를 들어, 투영변환이나 조명모델 또는 텍스처 매핑 방법의 종류를 변화시키거나 기존의 고정된 파이프라인에서는 다룰 수 없었던 좀더 사실적인 그래픽 효과가 쉐이더를 이용하여 GPU에서 그래픽스 파이프라인의 기능을 재정의 함으로써 가능하게 되었다.

본 논문에서는 GPU와 쉐이더를 이용하여 영상기반 렌더링을 가속하는 실용적인 방법을 제안한다. 일반적인 영상기반 렌더링의 가속에 GPU와 쉐이더가 이용될 수 있지만, 본 논문에서는 MPEG-4 AFX의 깊이 영상기반 랜더링(Depth Image-Based Rendering, DIBR)에 정의된 SimpleTexture 형식에 대하여 하드웨어 가속을 수행한다[3,4]. *SimpleTexture*는 3차원 물체에서 보여지는 면을 기준영상으로 하여 그것들의 집합으로 이루어진다. 기존의 Relief texture[5]와 유사하게, 각각의 기준 영상들은 텍스처 영상과 대응되는 깊이 영상으로 구성되고, 깊이 영상에서 각각의 픽셀 값은 카메라 평면으로부터 물체의 표면까지의 거리를 의미한다. 그림 1에 *SimpleTexture* 형식의 3차원 모델의 한 예를 도시하였다. 본 논문에서는 DIBR을 사용하여 *SimpleTexture* 형식을 묘사하고, 이러한 방법은 큰 변화 없이 *SimpleTexture* 이외의 다른 영상기반 렌더링 알고리즘에 확장될 수 있다.

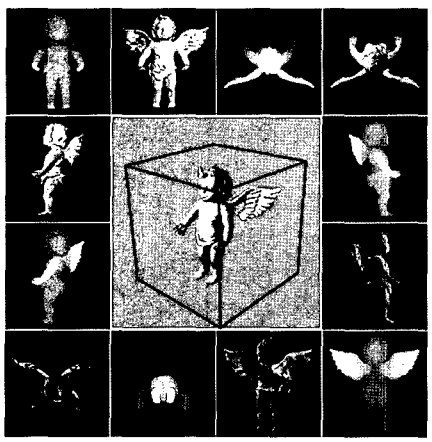


그림 1 DIBR 형식(*SimpleTexture* 모델)의 기준 영상과 임의의 시점에서 바라본 결과

본 논문의 구성은 다음과 같다. 제2장에서는 영상기반 렌더링과 관련된 기존 연구를 소개하고, 제3장에서는 제안하는 알고리즘을 기술한다. 제4장에서는 실험 결과를 제시 및 분석하고, 마지막으로 제5장에서 결론을 맺는다.

2. 기존의 연구

깊이 영상기반 3차원 모델은 깊이 영상과 텍스처 영상에서 규칙적으로 배열된 3차원 공간 상의 점들로 저장된다[3,4]. 즉, relief texture[5]와 유사하게, 픽셀의 값이 3차원 공간의 점과 카메라 평면 사이의 거리를 의미하게 된다. 일반적으로 기하학적인 변환이나 splatting 같은 3차원 워핑 알고리즘[1]을 이용하여 DIBR의 렌더링을 수행할 수 있지만, 하드웨어의 가속을 받을 수 없기 때문에 속도가 느리다는 단점이 있다.

영상기반 랜더링에는 광 필드(light field)렌더링과 점 기반 렌더링이라는 두 가지의 주목할만한 렌더링 방법론이 존재한다. 광 필드 렌더링 방법과 표면 광 필드(surface light field)[6,7]과 같은 확장된 렌더링 방법이 다양한 조명효과가 적용 된 높은 수준의 렌더링 결과를 제공하지만, 이러한 방법으로 구현된 프로그램은 많은 입력 영상을 필요로 하고 처리하는 데이타의 양이 많기 때문에 비실용적이다. 따라서, 본 논문에서는 적은 수의 영상으로 다양한 시점의 영상기반 렌더링이 가능한 표현 기법인 깊이 영상기반 표현 기법을 이용하여, 점 기반 렌더링[2]으로 영상기반 렌더링을 수행한다. 기존의 연구로 점 기반 렌더링에서의 스플랫 혼합이나 크기 및 형상 제어를 위하여 GPU 가속을 이용한 기법이 소개되었다[8,9].

기존의 연구 중 주목할 만한 연구로 Botsch와 Kobbelt의 점 기반 기하 표현[8]을 위한 GPU 가속에 기반한 렌더링 기법이 있다. 이 연구에서는 높은 질의 영상과 빠른 렌더링을 위하여 스플랫 크기와 모양의 계산, 스플랫 필터링, 픽셀 단위 정규화를 포함한 점 기반 렌더링의 대부분의 과정을 GPU 상에서 수행하였다. 본 논문에서는 점 기반 렌더링이 영상의 규칙적인 구조가 사라지고 전체를 점들의 집합으로 보는 것 대신, 영상의 카메라 정보와 같은 메타 데이타와 규칙적인 배열을 갖는 깊이 영상기반 렌더링의 특성을 이용하여 GPU에서 접근이 가능한 고속의 텍스처 버퍼를 활용한 깊이 영상기반 렌더링의 효율적인 알고리즘을 제안한다.

3. 제안하는 기법

본 논문에서는 GPU의 가속을 이용한 깊이 영상기반 3차원 물체의 고속 렌더링 알고리즘을 제안한다. 제안하는 기법은 기준영상 설정, 투영 변환, 조명 효과, 점 스플래팅의 순서로 이루어져 있다.

3.1 기본 프로그램의 구성

그림 2에 제안하는 렌더링 프레임웍의 구조를 제시하였다. 우선 통합 프로그램에서는 기준 영상을 재구성 하고 텍스처를 생성한다. 또한, 깊이 영상으로부터 법선 벡터 맵을 계산하고, 재구성된 기준 영상들과 법선 벡터 맵을 정점 쉐이더에서 접근이 가능하도록 하기 위하여 정점 텍스처로 설정한다. 그 후에 정점 쉐이더에서 전체 변환행렬을 계산하고, 조명에 의한 정점의 색과 스플랫의 크기를 결정한다. 마지막으로 화소 쉐이더에서 최종 vertex의 색을 출력하게 된다.

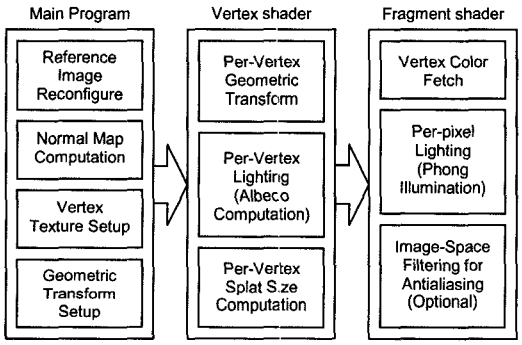


그림 2 제안하는 GPU 기반 렌더링의 전체 구조

3.2 정점 쉐이더와 화소 쉐이더

정점 쉐이더는 기하변환, 조명효과, 스플랫 크기의 계산을 위한 정점당(per-vertex) 연산을 수행한다. 이러한 연산을 위해 필요한 모든 데이타는 정점 텍스처의 형태로 저장되어 비디오 메모리에 저장되기 때문에, 일반적인 렌더링 시스템에서 발생하는 병목현상과는 달리 GPU와 시스템 메모리 사이의 데이타 이동이 적어지게 된다.

화소 쉐이더에서는 프레임 버퍼에 저장될 최종 픽셀 컬러를 계산하기 위한 화소당(per-fragment) 연산이 수행된다. 스크린 좌표계에서의 위치나 광원의 반사율, 그리고 스플랫의 크기와 같이 정점 쉐이더에서 연산된 결과들이 화소 쉐이더로 이동된다. 스플랫의 크기에 따라, 각각의 픽셀은 보통 몇 개의 스플랫에 의해 겹쳐지게 되고, 화소 쉐이더에서 Phong 조명 모델을 이용하여 각각의 스플랫의 색을 결정하게 된다.

3.3 텍스처 버퍼를 이용한 기준 영상의 저장

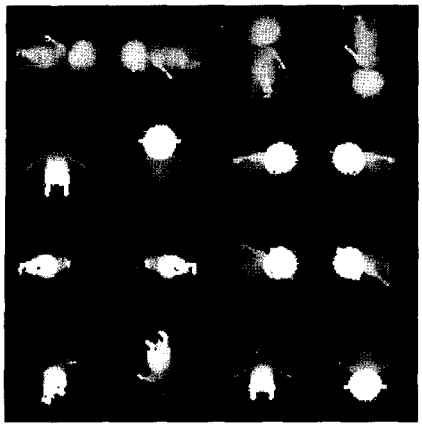
깊이 영상기반 그래픽 데이타의 기준 영상을 텍스처로 간주하고 그 데이타를 그래픽 카드의 텍스처 버퍼에 저장할 수 있기 때문에 GPU를 이용하여 하드웨어 가속을 수행하기에 적합하다. 기준 영상을 시스템 메모리에 저장하는 기존의 렌더링 방법과 비교하여, 제안하는 알고리즘은 AGP나 PCI-E와 같은 그래픽 하드웨어의 특수한 버스를 통하여 이동하는 데이타의 양을 줄여주기 때문에 렌더링 속도의 향상이 가능하다. 즉, 기하변환, 조명에 의한 음영 효과, 스플래팅을 포함한 모든 렌더링 과정이 GPU와 텍스처 버퍼를 통해서 이루어지게 된다. 이에 따라 CPU의 부담을 줄여주고 CPU가 다른 프로세스의 처리를 수행하도록 할 수 있다.

본 논문에서는 정점 쉐이더에서 텍스처 버퍼에 접근하기 위하여 정점 텍스처(Vertex Texture)[10]라는 쉐이더 모델 3.0에서 정의된 새로운 특징을 사용한다. 정점 텍스처는 기존에 화소 쉐이더에서만 가능했던 텍스처 데이타로의 접근이 정점 쉐이더에서도 가능하도록 하는 것으로써, 쉐이더 모델2.0에서는 지원되지 않았던 새로운 특징이다. 그러나 정점 텍스처의 한가지 문제점은 정점 쉐이더에서 접근할 수 있는 텍스처의 최대 개수가 4개라는 점이다. 이러한 제약으로 인해 큐브 맵(cube map)과 같이 12개 이상의 기준영상이 사용되는 깊이 영상기반 그래픽 데이타의 모든 기준영상을 텍스처 버퍼에 함께 저장하지 못하게 된다. 또한, 복수의 텍스처를 이용하는 경우 활성화된 텍스처의 잦은 변경으로 인해 렌더링 속도의 저하가 일어난다. 이러한 문제점을 해결하기 위하여, 본 논문에서는 기준 영상을 네 개 이하의 텍스처 영상으로 병합하여 이용한다. 그림 3(a)와 3(b)에서, 32개의 기준 영상을 하나의 깊이 영상과 컬러 영상으로 병합한 결과를 보여주고 있다. 이 때, 병합된 영상의 해상도는 GPU가 허용하는 텍스처의 최대 해상도를 초과할 수 없다.

3.4 기하 변환 및 조명

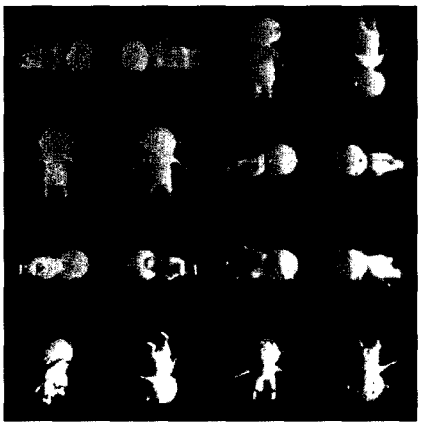
기하 변환은 영상의 기준 좌표계에서 월드 좌표계로의 변환 $T_{R\to W}$, 월드 좌표계에서 카메라 좌표계로의 변환 $T_{W\to C}$, 카메라 투영 변환 T_P의 세 개의 중간 변환으로 구성되어 있다. 카메라의 위치와 바라보는 시점이 항상 변하기 때문에 본 논문에서는 변환 $T_{R\to W}$는 고정된 상태에서 최종 변환 행렬 $T_{PTW\to C}T_{R\to W}$는 정점 쉐이더에서 계산하고 기준 영상의 점 데이타에 적용된다.

법선 벡터는 깊이 영상을 이용하여 전처리 과정에서 계산된다. 법선 벡터는 조명 모델에 의한 곡면 음영을 계산할 때 사용되어야 하기 때문에 정점 쉐이더에서 법선 벡터의 접근이 가능하여야만 한다. 본 논문에서는 기준 영상의 각각의 점에 대응하는 법선 벡터를 깊이 영상을 위하여 계산한다. 이 때, 법선 벡터(x, y, z)의 각 원소는 [-1, 1]의 범위를 가지므로 이를 [0, 255]의 범위로 변환하여 영상의 RGB 채널로 저장하기 때문에 이를 텍스처로 지정하여 비디오 메모리 상의 텍스처 버퍼에 저장할 수 있다. 그림 3(c)에 그림 3(a)와 3(b)의 기준 영상에 대한 법선 벡터 맵을 영상으로 도시하였다. 이러한 방법으로 법선 벡터 맵은 정점 쉐이더에서 접근이



(a)

(b)



(c)

그림 3 해상도 1024×1024으로 병합된 기준영상들과 정점 텍스처를 위한 법선 벡터 맵. 병합되기 이전의 기준영상들의 해상도는 256×256이다. (a) 병합된 깊이 영상. (b) 병합된 컬러 영상. (c) 기준 영상들에 의해 만들어진 법선 벡터 맵

가능하게 되고, 각각의 점에 대하여 조명 효과에 의한 음영을 계산할 때 사용되게 된다. 본 논문에서는 OpenGL의 성능과 비교하기 위하여 일반적인 Phong 조명 모델을 사용하였으나, 임의의 조명 모델을 정점 쉐이더에서 구현할 수 있으므로 다른 조명 모델로의 확장이 가능하다.

본 논문에서는 법선 벡터의 각 원소를 8비트로 양자화하여 근사하였다. 현재 쉐이더 모델 3.0의 경우 채널당 16비트의 부동 소수점 텍스처를 지원하지만, 깊이 영상으로부터 법선 벡터를 추정할 때의 오차가 존재함을 고려할 때 본 논문에는 이를 사용할 근거가 떨어진다. 또한 이 경우 법선 벡터 맵을 위한 메모리 공간이 2배 이상 필요하게 되므로 본 논문에서 사용한 8비트 양자화를 이용하면 불필요한 메모리의 사용을 막을 수 있다.

3.5 스플랫 크기 변환

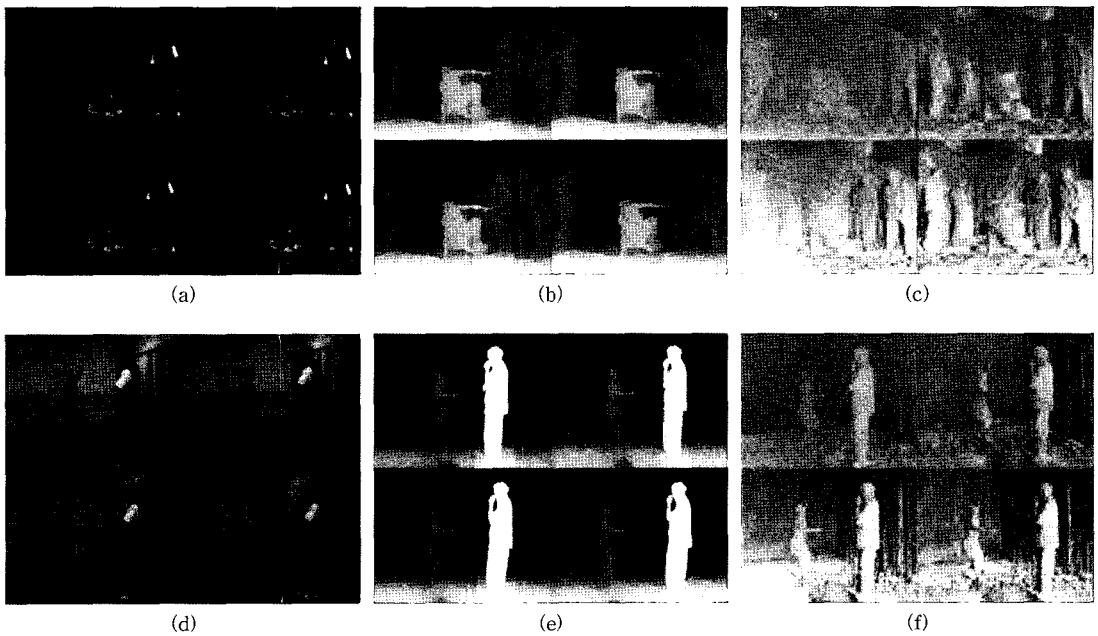
영상 기반 렌더링은 영상의 화소를 하나의 3차원 점으로 간주하여 점 기반 렌더링을 수행하기 때문에 객체를 확대하는 경우 물체 표면에 구멍이 관측되는 오류가 발생하게 된다. 이러한 문제를 해결하기 위하여 스플랫의 크기를 시점으로부터의 거리에 따라 적응적으로 조절하여야 하며 이는 매 화소에 대해 수행해야 한다. 본 연구의 목적은 최적의 스플랫의 크기 및 모양을 설계하는 것은 아니므로, 본 논문에서는 식 (1)과 같은 시점으로부터의 거리의 2차식에 반비례하는 일반적인 스플랫 크기 조절 관계식에 의해 경험적으로 스플랫의 크기를 결정하도록 한다. 식 (1)에서, W_S는 스플랫의 폭을 나타내고, d는 시점과 스플랫 사이의 거리를 나타내며, 비례상수 α, β, γ, δ는 경험적으로 계산되었다. 이 때, 유리 2차식의 연산이 GPU에 의해 수행되므로 CPU에서 스플랫의 크기를 계산하는 것에 비해 매우 빠른 처리가 가능하다.

$$W_S = \frac{\alpha}{\beta d^2 + \gamma d + \delta} \tag{1}$$

4. 실험결과

제안하는 알고리즘의 성능을 평가하기 위하여, 그림 3과 같은 합성 데이타와 그림 4에 제시된 실제 영상에 대하여 실험을 수행하였다. 알고리즘의 성능평가는 nVidia GeForce 7800GTX GPU와 2G 바이트의 메모리를 장착한 2.0GHz Athlon 64 CPU상에서 수행되었다. 렌더링과 쉐이더 프로그래밍은 OpenGL 2.0[11]에 포함된 GLSL(GL shading language)[12]을 이용하여 구현하였다.

표 1과 표 2는 제안하는 알고리즘과 일반적인 렌더링 방법들과의 통계적인 수치를 비교하여 보여주고 있다. 표에서 보여주는 바와 같이, 소프트웨어 렌더링이나 일반적인 OpenGL 렌더링과는 대조적으로 제안하는 방법



(a) (b) (c)

(d) (e) (f)

그림 4 실제 테스트 데이타[13]. 각 영상의 해상도는 2048×1536이다. (a) Break-dance - 컬러 영상. (b) Break-dance - 깊이 영상. (c) Break-dance - 법선 벡터 맵. (d) Ballet - 컬러 영상. (e) Ballet - 깊이 영상. (f) Ballet - 법선 벡터 맵

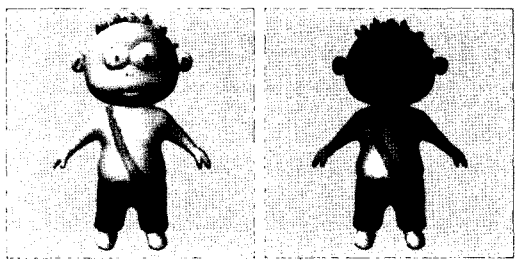
표 1 렌더링 속도의 비교(합성 데이타[4])

기준 영상의 개수	# of points	Frames Per Second (FPS) Points Per Second (MPPS)		
		Shader	OpenGL	Software
4	45,012	559.84	94.17	14.59
		25.19	4.24	0.66
9	83,521	310.18	50.96	7.89
		25.74	4.23	0.65
16	167,980	152.32	25.25	3.93
		25.44	4.22	0.66

표 2 렌더링 속도의 비교(실제 데이타 - Ballet[13])

기준 영상의 개수	# of points	Frames Per Second (FPS) Points Per Second (MPPS)		
		Shader	OpenGL	Software
1	786,432	33.7	5.44	0.85
		26.35	4.25	0.67
4	3,145,728	8.29	1.35	0.22
		26.02	4.23	0.69
8	6,291,456	4.78	0.78	0.12
		30.04	4.87	0.77

의 프레임 수가 눈에 띄게 증가하는 것을 확인할 수 있다. 실험결과, 쉐이더 기반의 렌더링 방법이 OpenGL에 비하여 여섯 배 이상의 속도 향상을 보여주고 있다. 일반적인 OpenGL 렌더링에서는 쉐이더 기반의 렌더링과



(a) (b)

그림 5 조명 효과의 렌더링 결과. (a) 일반적인 렌더링. (b) 조명 효과 적용 결과

비교해서, 스플랫의 크기를 계산할 때 쉐이더를 사용하지 않는 한 하드웨어의 가속을 받을 수 없다는 차이점이 있다. 본 논문에서는 시점으로부터의 거리의 2차식에 반비례하도록 스플랫의 크기를 결정하였다. 비례상수는 경험적으로 계산되었고, 차후에 다양한 깊이 영상 기반 렌더링에 대해서도 최적의 스플랫 모양과 크기를 찾는 연구가 수행될 필요성이 있다.

그림 5는 동적인 조명효과가 적용된 결과를 보여준다. 그림 5(a)는 일반적인 깊이 영상기반 렌더링의 결과로 모델을 취득할 당시 기준영상에 적용된 조명효과를 보여주고 있고, 그림 5(b)는 본 논문에서 제안 하는 방법에 의해 조명이 모델의 왼쪽 아래에 위치하여 발생하는

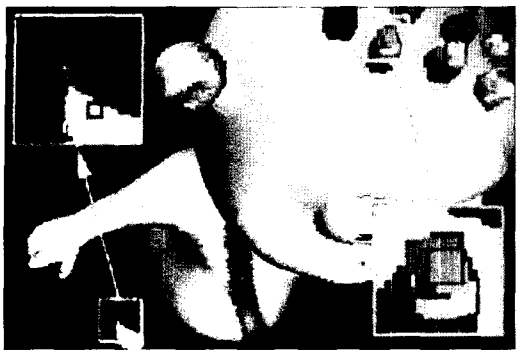


그림 6 GPU 연산에 의한 스플랫 크기의 차이를 보여주는 예제

조명효과를 보여줌으로써 렌더링 결과의 사실성이 증가하는 것을 알 수 있다.

그림 6은 스플랫의 크기가 변화하는 예제를 보여주고 있다. 그림 6에서, 멀리 떨어져있는 면의 스플랫의 크기(그림 왼쪽 위의 확대도)가 가까운 면의 스플랫의 크기(그림 오른쪽 아래의 확대도)보다 더 작은 것을 확인할 수 있다.

5. 결 론

본 논문에서는 깊이 영상기반 3차원 그래픽 데이타의 실용적인 렌더링 기법을 제안하였다. 최근 GPU의 새로운 특징과 프로그래밍이 가능한 쉐이더를 이용하여 영상기반 3차원 물체의 효과적인 하드웨어 가속 알고리즘을 개발하였다. 실험결과, 소프트웨어 기반의 렌더링이나 일반적인 OpenGL 기반의 렌더링 방법과 비교하여 주목할만한 속도의 향상이 이루어졌다.

향후 연구로 임의의 시점에 대한 스플랫의 모양과 크기의 최적화 문제가 가능할 것이다. 또한, 주어진 3차원 모델에서 기준 영상의 개수를 최소로 하는 최적의 기준 영상을 선택하는 효과적인 알고리즘의 디자인 또한 가치가 있을 것이다.

참 고 문 헌

[1] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," *Proc. SIGGRAPH '95*, pp. 39-46, August 1995.

[2] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "Surface splatting," *Proc. SIGGRAPH '01*, pp. 371-378, July 2001.

[3] Information Technology - Coding of Audio-Visual Objects - Part 16: Animation Framework eXtension (AFX), ISO/IEC Standard JTC1/SC29/WG11 14496-16: 2003.

[4] L. Levkovich-Maslyuk et al, "Depth image-based representation and compression for static and animated 3D objects," *IEEE Trans. on Circuits and Systems for Video Technology*, 14(7): 1032-1045, July 2004.

[5] M. Oliveira, G. Bishop, and D. McAllister, "Relief textures mapping," *Proc. SIGGRAPH '00*, pp. 359-368, July 2000.

[6] D. Wood et al, "Surface light fields for 3D photography," *Proc. SIGGRAPH '00*, pp. 359-368, July 2000.

[7] W. Chen et al, "Light field mapping: Efficient representation and hardware rendering of surface light fields," *ACM Trans. on Graphics*, 21(3): pp. 447-456, July 2002.

[8] M. Botsch and L. Kobbelt, "High-quality point-based rendering on modern GPUs," *Proc. 11th Pacific Conference on Computer Graphics and Applications*, October 2003.

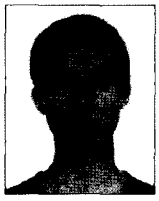
[9] R. Pajarola, M. Sainz, and P. Guidotti, "Confetti: Object-space point blending and splat-ting," *IEEE Trans. on Visualization and Computer Graphics*, 10(5): 598-608, September/October 2004.

[10] NVIDIA GPU Programming Guide Version 2.2.0, http://developer.nvidia.com/object/gpu_programming_guide.html

[11] J. Leech and P. Brown (editors), *The OpenGL Graphics System: A Specification (Version 2.0)*, October 2004.

[12] R. Rost, *OpenGL Shading Language Second Editition*, Addison Wesley, 2006.

[13] C. Zitnick et al, "High-quality video view interpolation using a layered representation," *ACM Trans. on Graphics, 23(3)*: 600-608, August 2004.



이 만 희

2006년 2월 인하대학교 컴퓨터공학과 공학사. 2006년 3월~현재 인하대학교 정보통신공학과 석사과정. 관심분야는 컴퓨터 그래픽스, 멀티미디어 응용

박 인 규

1995년 2월 서울대학교 제어계측공학과 공학사. 1997년 2월 서울대학교 제어계측공학과 공학석사. 2001년 8월 서울대학교 전기컴퓨터공학부 공학박사. 2000년 1월~2000년 4월 미국 Univ. of Michigan, Ann Arbor 방문연구원. 2001년 9월~2004년 3월 삼성종합기술원 멀티미디어랩 전문연구원. 2004년 3월~현재 인하대학교 정보통신공학부 조교수. 관심분야는 컴퓨터 그래픽스 및 비젼, 영상처리, 멀티미디어 응용분야