GPU를 이용한 영상기반 렌더링의 가속

이만희° 박인규
인하대학교 정보통신공학부 디지털미디어연구실
{maninara@hotmail.com, pik@inha.ac.kr}

GPU-based Acceleration of Image-based Rendering

Man Hee Lee° In Kyu Park
Digital Media Laboratory, School of Information and Communication Engineering, INHA University

요 약

본 논문에서는 깊이 영상기반 3차원 물체(depth image-based 3-D object)의 고속 렌더링 기법을 제안한다. 제안하는 알고리즘은 그래픽 가속기가 지원하는 shader programming 기법을 이용하여 하드웨어 가속을 직접 이용하도록 설계되었다. 또한, 기존의 영상 기반 렌더링의 한계를 극복하여 조명 효과를 표현할 수 있으며 렌더링시 각 화소당 splat 크기를 하드웨어에서 직접 조절하여 고속 렌더링이 가능하다. 모의 실험결과, 소프트웨어 렌더링 또는 OpenGL 기반의 렌더링에 비해 괄목할 만한 렌더링 속도의 향상이 이루어졌다.

1. 서론

최근 그래픽 가속기(Graphics Processing Unit : GPU)의 성능이 급격히 발전하고 GPU 자체의 프로그래밍 가능성이 점차 확대됨에 따라 3차원 그래픽스의 기본적인 기하 변환과 rasterization 구조의 변경 및 이들 이외의 범용 목적으로 GPU를 활용할 수 있는 가능성이 크게 대두되었다. 즉, 최근의 GPU는 vertex shader와 fragment shader를 지원하여 사용자가 원하는 방법으로 기하변환 및 rasterization 파이프라인의 기능을 변경할 수 있다. 특히, 최근 발표된 DirectX 9.0과 OpenGL 2.0에서는 각각 HLSL과 GLSL [8]이라는 고급 shading 언어를 지원하여, 기존의 어셈블리 언어를 이용할 경우의 불편함을 극복하였다. 이에 따라, 관련 하드웨어 및 어플리케이션의 개발이 급속히 가속화 될 전망이며, 각종 3차원 컴퓨터 게임에서는 범프 매핑, 환경 매핑, 파티클 효과등의 특수 효과를 이미 shader를 이용하여 구현하고 있다.

한편, 복잡한 형상의 3차원 물체를 적은 량의 데이터로 효율적으로 표현하기 위하여 MPEG-4 국제 표준화 기구에서는 깊이 영상 기반 3차원 물체의 표현 방법을 제정하였다 [1]. MPEG-4 AFX [1]에 포함된 표현 기법은 SimpleTexture, PointTexture, OctreeImage라는 세 가지 포맷으로 정의되어 있으며 텍스쳐 영상과 깊이 영상 또는 3차원 복셀 모델로 표현되는 기하 정보를 가지고 있어 다각형의 이용없이 3차원 형상을 표현할 수 있다. 이러한 모델의 렌더링은 기존의 3차원 워핑(warping) 알고리즘[2]를 이용하여 수행된다. 그러나, 이러한 렌더링 기법은 GPU의 가속을 이용하기 어렵기 때문에 고속의 GPU가 장착되어 있는 시스템에서는 다각형 렌더링에 비해 느리다는 단점을 가지고 있다. 이로 인해 깊이 영상 기반 물체에 대해 고속 렌더링 알고리즘 개발의 필요성이 대두되고 있다.

2. 기존의 연구

깊이 영상 기반 3차원 모델은 규칙적으로 배열된 3차원 공간상의 점 모델로 구성되어 있다는 특징이 있다. 즉, 격자 형태의 영상의 각 픽셀에 위치한 점들이 카메라의 위치와 시선 방향에 따라 3차원 공간상에 존재하는 형태이다 [1][5]. 따라서 이러한 모델의 렌더링은 일반적인 그래픽스의 다각형(polygon) 렌더링이 아닌 점 기반 렌더링(point-based rendering)[3][4]의 방식으로 수행되어야 한다.

점 기반 렌더링 기법은 기존의 다각형 기반 렌더링과 달리 삼각형(triangle) 대신 점 자체를 렌더링의 기본 요소로 이용한다. 이와 같은 방식은 3차원 모델이 다각형의 형태를 가지고 있지 않거나 모델의 해상도가 프레임 버퍼의 해상도에 비해 높을 때 매우 유용한 렌더링 기법이다. 최근에는 이러한 특성에 기반하여 영상 기반 3차원 모델의 렌더링을 모바일 기기에서 고속으로 수행하기 위한 연구도 수행되었다 [7].

점 기반 렌더링시 hole과 aliasing 현상이 일어나지 않도록 하기 위하여 렌더링의 가장 기본 요소로서의 프레임 버퍼상에 그려주는 점(splat)의 크기와 형태를 최적화하여야 한다. 이를 위해 다양한 연구가 수행되었으며 법선 벡터를 가지는 타원 디스크 형태[3][4]의 splat이 가장 일반적으로 이용되고 있다. 최근에는 일반적인 점 구름(point cloud)형태의 3차원 데이터에 대해 수행하는 점 기반 렌더링을 그래픽 가속기를 이용하여 하드웨어적으로 고속 렌더링을 수행하는 연구가 초기 단계에서 진행되고 있다 [4][6].

3. 제안하는 렌더링 기법

본 논문에서는 GPU의 가속을 이용한 깊이 영상 기반 3차원 물체의 고속 렌더링 알고리즘을 제안한다. 특히, 깊이 영상 기반 표현 기법중 SimpleTexture에 대해, GLSL을 이용한 GPU의 렌더링 파이프 라인의 변경을 통해 렌더링의 가속을 꾀한다. 즉, GPU가 일정 개수의 텍스쳐 영상을 직접 비디오 메모리에 저장하여 처리할 수 있는 특징을 이용하여 SimpleTexture의 기준 영상을 구성하는 깊이 영상과 컬러 영상을 텍스쳐 버퍼에 저장한 후, 기하 변환, 조명에 의한 shading 효과, 그리고 rasterization

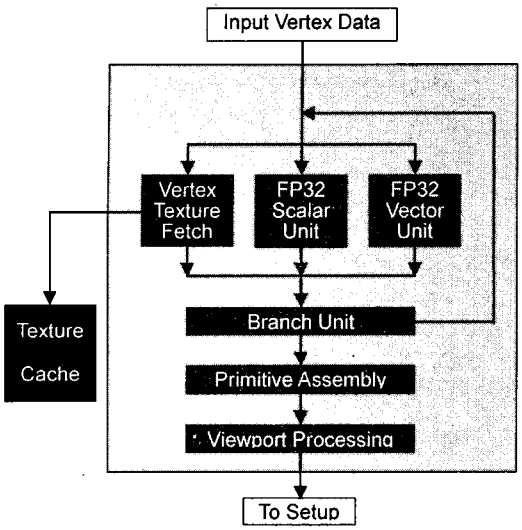


그림 1. Vertex Shader Pipeline

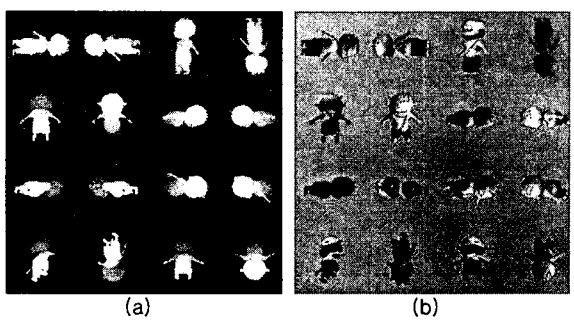
을 GPU에서 직접 처리함으로써 렌더링을 가속화한다. 이 때, CPU와 GPU상의 버스를 통한 불필요한 데이터 전송이 최소화되고 GPU의 강력한 처리 능력을 최대한 활용할 수 있다.

3.1 기준 영상의 재구성을 통한 데이터 이동의 최소화

Vertex shader에서 텍스쳐에 접근하기 위해서 shader 3.0에서 제공하는 vertex texture를 이용 한다. 그림 1에서와 같이 vertex texture는 pixel shader와 마찬가지로 vertex shader가 텍스쳐 데이터에 접근할 수 있도록 한다. Shader 3.0 버전에서는 최대 4개의 vertex texture를 제공한다. 그러나 깊이 영상 기반 3차원 모델에서 사용하는 기준 영상의 개수가 일반적으로 6개 이상이기 때문에, 이 경우 기준 영상 모두를 vertex texture로서 사용할 수 없다. 이러한 문제를 해결하기 위하여 본 논문에서는 그림 2과 같이 기준 영상을 공통의 텍스쳐 데이터로 병합한 새로운 기준 영상을 사용하여 vertex texture로서 사용할 수 있도록 한다.

3.2 기하 변환 및 조명 연산의 가속

제안하는 알고리즘에서는 기준 영상의 정보와 현재 바라보는 시점을 이용하여 기하 변환 행렬을 설정하고 이를 shade로 넘겨주어 GPU상에서 기하 변환을 수행한다. 또한 vertex shader



(a) (b)

그림 2. Vertex texture를 위한 새로운 기준 영상. (a) 병합된 깊이 영상. (b) 병합된 컬러 영상.

에서 해당 화소의 주변 화소들을 이용하여 자신의 3차원 공간에서의 법선 벡터를 계산하고 광원의 위치와 법선 벡터를 이용하여 조명에 의한 컬러를 계산함으로써 하드웨어적으로 T&L 가속을 구현한다. 이 때, 조명 모델로서는 Phong의 모델을 이용한다.

3.3 적응적 splatting

렌더링시 물체를 확대, 축소할 때, hole을 발생시키지 않으면서 aliasing 현상을 동시에 최소화시키기 위해 splat의 크기를 적응적으로 변경하여야 한다. 본 논문에서는 이를 위하여 vertex shader에서 카메라의 위치로부터 3차원 점까지의 거리를 구하고 거리에 반비례하도록 point size를 결정함으로써 적응적인 splatting을 구현한다. 그러나 향후 연구를 통해 이의 최적화는 필요한 상황이다.

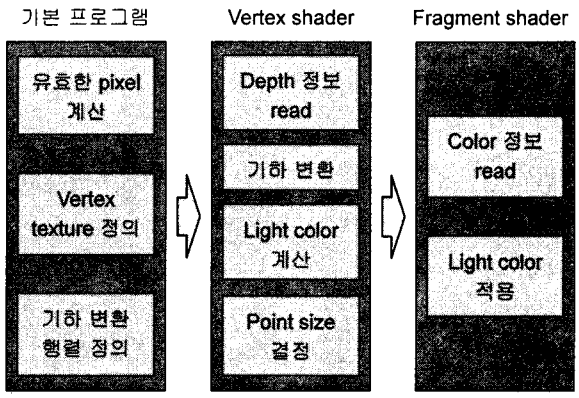


그림 3. 프로그램 구조

4. 실험 결과

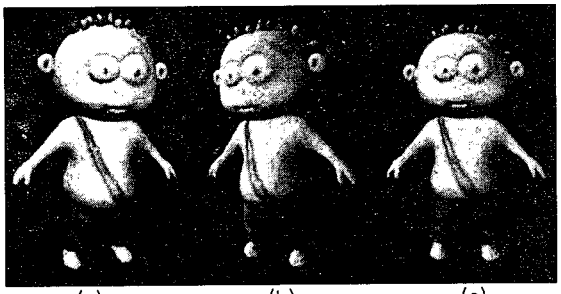
본 실험에서는 shader를 이용하여 GPU의 하드웨어 가속을 이용한 제안하는 깊이 영상기반 렌더링 프로그램을 구현하였다. 본 논문에서 제안한 알고리즘의 성능을 비교하기 위하여 기존의 연구를 하드웨어 T&L 가속만을 이용한 프로그램과 소프트웨어적으로 렌더링을 수행한 프로그램을 추가로 구현하여 성능의 비교를 수행하였다.

4.1 실험 환경

알고리즘의 성능평가는 nVidia GeForce 6800 GPU와 2G 바이트의 메모리를 장착한 2.0GHz Athlon 64 CPU상에서 수행되었다. Windows XP에서 Visual C++ 6.0 툴을 사용하였고 OpenGL 2.0을 이용하여 렌더링 하였다.

4.1.1 기본 프로그램의 구성

그림 3에서 보는 바와 같이 기본 프로그램에서는 깊이 영상을 이용하여 유효한 픽셀을 계산하고 그 정보를 shader로 넘겨준다. 이 경우도 마찬가지로 CPU와 GPU 사이의 데이터 전송을 최소화 하기 위하여 ARB_vertex_buffer_object 확장을 이용한다. 그리고 깊이 영상을 vertex texture로 정의하여 비디오 메모리에 저장하고, 기준 영상의 카메라 정보를 이용하여 기하변환 행렬을 정의한다.



(a) (b) (c)

그림 4. 그림 2 의 기준영상을 렌더링한 결과. (a) 제안하는 렌더링. (b) T&L 가속만을 이용. (c) 소프트웨어 렌더링.

4.1.2 Vertex shader 구성

Vertex shader에서는 vertex texture로 정의된 깊이 정보를 읽고 기하 변환 행렬과의 연산으로 실제 화면에 투영될 좌표를 계산한다. 또한 광원의 위치와 법선 벡터를 이용하여 광원의 유효 강도을 계산 하여 fragment shader로 넘겨주고 카메라의 위치와 현재 vertex 사이의 거리를 이용하여 splat의 크기를 결정한다.

4.1.3 Fragment shader 구성

Fragment shader에서는 텍스쳐로 정의된 컬러 영상의 정보로부터 pixel의 컬러값을 얻고 이를 재질의 반사율로 이용한다. 따라서, vertex shader에서 계산된 광원의 유효 강도과 반사율을 이용하여 Phong 조명 모델로부터 최종적인 컬러값을 계산할 수 있다.

4.2 렌더링 결과

제안하는 알고리즘과 기존의 기법을 비교하기 위하여 그림 4와 표 1에 렌더링의 결과와 속도를 제시하였다. 표 1에서 초당 프레임 수(FPS)는 버퍼를 swap하는 시간을 제외한 순수한 렌더링 시간만을 측정한 자료이다. 표 1에서 보는 바와 같이 shader로 가속하였을 경우 초당 프레임 수가 기존의 렌더링 기법보다 비약적인 성능 향상이 관측 되었다. 이는 CPU와 GPU사이의 데이터 이동을 최소화 함으로써 가능하였다.

또한 그림 5에서 보는 바와 같이 카메라의 위치로부터 가까운 vertex와 멀리 떨어져있는 vertex의 point size가 확연하게 구분되어 적응적 splatting이 잘 적용되는 것을 알 수 있다.

표 1. 렌더링 시간과 픽셀 처리 수 비교

기준영상 개수	# points	초당 프레임 수 (FPS), 초당 픽셀 처리 수 (MPPS)		
		Shader	T&L 가속	가속 없음
1	9196	19484.67	413.71	338.61
		179.18	3.80	3.11
4	45012	384.98	86.42	72.27
		17.33	3.88	3.25
9	83521	119.63	46.76	38.68
		9.99	3.91	3.23
16	167980	51.15	23.34	19.13
		8.59	3.92	3.21

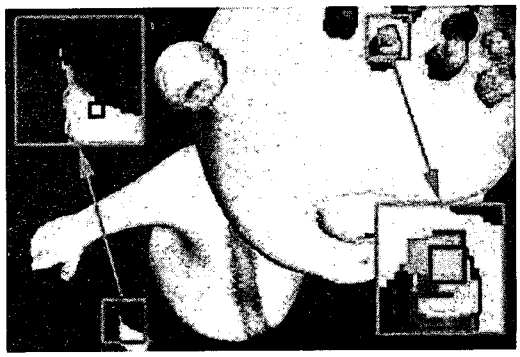


그림 5. 시점에서 가까운 vertex와 멀리 떨어져있는 vertex의 point size 비교

5. 결론

본 논문에서는 깊이 영상기반 3차원 물체의 고속 렌더링 기법을 제안하였다. Shader를 이용하여 T&L 가속을 수행하고 vertex texture를 이용하여 CPU와 GPU사이의 데이터 이동을 최소화 하였다. 또한 적응적으로 splatting을 적용함으로써 확대시 발생할 수 있는 hole을 제거하였다. 실험 결과, 제안된 방법이 기존 연구들에 비해 최소 두 배이상의 속도 향상이 이루어졌음을 알 수 있다.

향후 연구로서는 비디오 메모리의 사용을 줄이는 기존 영상의 최적화와 splat의 형상의 최적화를 수행하여 보다 고화질, 고속의 렌더링을 수행할 수 있도록 할 계획이다.

참고문헌

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

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

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

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

[5] L. Levkovich-Maslyuk, A. Ignatenko, A. Zhirkov, A. Konushin, I. K. Park, M. Han, and Y. Bayakovski, "Depth image-based representation and compression for static and animated 3D objects," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 14, no. 7, pp. 1032-1045, July 2004.

[6] R. Pajarola, M. Sainz, and P. Guidotti, "Confetti: Object-space point blending and splatting," *IEEE Trans. on Visualization and Computer Graphics*, vol. 10, no. 5, pp. 598-608, September/October 2004.

[7] G. J. Jang, K. H. Kim, and I. K. Park, "Depth image-based rendering of 3-D object on mobile device," *Proc. IEEE International Symposium on Intelligent Signal Processing and Communication Systems*, pp. 630-633, Seoul, Korea, November 2004.

[8] R. Rost, *OpenGL® Shading Language*, Addison Wesley, 2004.