물체의 구 좌표계 표현을 이용한 효율적인 렌더링 방법*

한은호, 홍현기
중앙대학교 첨단영상대학원 영상학과
ehhan@wm.cau.ac.kr, honghk@cau.ac.kr

An Efficient Rendering Method of Object Representation Based on Spherical Coordinate System

Eunho Han, Hyunki Hong
Dept. of Image, GSAIM Chung-Ang Univ.

요 약

본 논문에서는 보다 효율적인 렌더링을 위해 물체를 구좌표계(sperical coordinate system) 상에서 표현하는 새로운 렌더링 알고리즘이 제안된다. 먼저 직교 좌표로 표현되어 있는 물체의 정점을 구좌표로 변환하고, 카메라의 가시 절두체(frustum) 영역 내의 정점을 판단하기 위해 삼각형의 무게중심, 색인(index), 메모리 접근(access) 맵 등의 자료구조를 구성한다. 제안된 방법은 카메라에 의해 보여지는 영역, 즉 렌더링되는 물체의 가시 영역에 해당하는 정점만으로 렌더링한다. 따라서 렌더링 파이프라인에서 고려되는 정점의 개수를 크게 줄여 전체적인 시스템 성능이 크게 향상되었음을 실험을 통해서 확인하였다.

ABSTRACT

This paper presents a novel rendering algorithm based on sperical coordinate representation of the object. The vertices of the object are transformed into the sperical coordinate system, and we construct additional maps: the centroid and index of the triangle, the memory access table. While OpenGL rendering pipeline touches all vertices of an object, the proposed method takes account of the only visible vertices by examining the visible triangles of the object. Simulation results demonstrated that the proposed method achieve an efficient rendering performace.

Keyword : real-time rendering, sperical coordinate system, Cg programming, visibility

접수일자 : 2008년 7월 11일
심사완료 : 2008년 8월 13일
* 본 연구는 서울시 산학연협력사업, 교육부 2단계 두뇌한국 21(BK21), 2008년도 중앙대학교 우수연구자 연구비 지원으로 수행되었음.

1. 서 론

3차원 공간에서 장면 및 물체에 대한 공간 자료 구조(spatial data structure)를 이용해 렌더링 속도를 개선하는 연구는 컴퓨터 그래픽스 분야에서 중요한 주제이다. 이를 위해 3차원 공간에 대한 BVH(Bounding Volume Hierarchy), BSP(Binary Space Partitioning), 8진 트리, 장면 그래프 등의 가속화 구조를 일반적으로 이용한다[1-3]. 즉, 3차원 공간상에 장면에 대한 효율적인 기하 구조를 사용하여 카메라의 시점으로부터 광선(ray)과 물체 사이의 교차 관계, 가려짐, 광선 추적 등의 연산 시간을 줄일 수 있다.

렌더링 성능을 개선하기 위한 또 다른 방법으로 선별(culling), 상세단계(level of detail) 등이 있다. 카메라에 의해 보이지 않는 부분은 최종 이미지에 기여하지 않으므로 렌더링할 필요가 없기 때문에, 선별 알고리즘은 장면 내에 처리될 필요가 없는 다각형들이 래스터화되지 않도록 한다. 여기에는 후면(backface) 선별, 계층적 시각 절두체(hierarchical view frustum) 선별, 포탈(portal) 선별, 차폐(occlusion) 선별 등의 다양한 알고리즘이 제안되었다[3,4]. 상세단계는 어떤 물체가 작거나 렌더링된 이미지에 별로 기여하지 못할 경우에 선택적으로 간략한 표현을 사용하여 렌더링 속도를 개선하는 연구이다[5].

3차원 공간에 대한 구좌표(spherical coordinate)의 표현은 주로 환경맵(environment map) 및 영상기반(image-based) 기술 등에서 사용된다[6-8]. 주변 환경에 분포하는 조명과 텍스쳐 정보를 해석하기 위해 전방향(omni-directional) 영상을 취득하고 이를 구좌표계로 표현한다. 즉, 합성하고자 하는 가상의 물체 주변에 존재하는 조명 등을 이용하며, 주변의 기하학적인 복잡도에 관계없이 사실적인 장면을 빠른 시간 내에 렌더링할 수 있다.

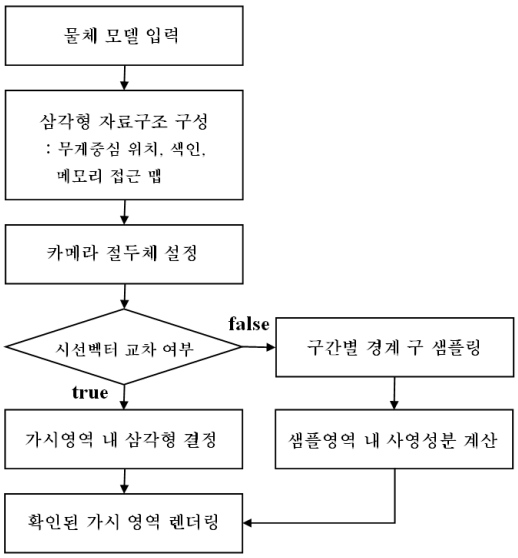


그림 1. 제안된 방법의 흐름도

본 논문에서는 효율적인 렌더링을 위해 물체를 구좌표계로 표현하는 새로운 렌더링 알고리즘이 제안된다. 구 좌표계에서 물체를 구성하는 모든 정점은 방향과 원점으로부터의 거리에 의해 표현되며, 물체의 기하정보를 효과적으로 접근(access)하기 위해 삼각형 무게중심 위치, 삼각형 색인(index), 메모리 접근 맵 등을 구성한다. 카메라의 절두체와 렌더링하고자 하는 물체가 이루는 관계를 분석하고, 절두체의 4개의 모서리 벡터와 물체에 대한 구좌표계 표현 정보를 비교하여 가시 영역을 효과적으로 판단한다. 제안된 방법의 흐름도를 그림 1에 나타내었다.

장면 내 물체를 구성하는 모든 정점을 고려하지 않기 위해 후면 선별 등의 가속화 방법이 제안되었지만, 이는 시선 벡터와 면 법선(normal) 벡터간의 각도를 조사하는 별도의 연산과정이 요구된다[3,9]. 제안된 방법은 사전에 저장된 자료구조를 바탕으로 카메라에 의한 물체의 가시 영역을 별도의 연산과정 없이 판단할 수 있다. 따라서 물체를 구성하는 정점 중에서 렌더링되지 않는 가려진 부분에 대한 연산의 부하를 줄여서 렌더링 효율을 크

게 향상시켰음을 실험을 통해 확인하였다.

2장에서는 물체의 구 좌표 변환과 관련 자료구조의 구성 방법을 소개하고, 3장에서는 제안된 방법에 의해 얻어진 렌더링 영상과 성능을 분석하며, 결론 및 이후 연구를 4장에 기술한다.

2. 제안된 방법

본 장에서는 직각 좌표계에서 정의되는 물체의 정점을 구좌표계로 변환하고, 렌더링 과정에서 삼각형을 효율적으로 탐색하기 위한 자료구조의 생성, 그리고 이를 이용한 렌더링 알고리즘 등을 기술한다.

2.1 물체의 구 좌표계 변환

일반적으로 물체의 모델링 데이터는 직교좌표계로 표현된 정점(x, y, z)과 정점들이 이루는 삼각형 표면, 그리고 각 표면의 법선벡터 정보 등으로 구성된다. 본 논문에서는 효율적인 렌더링을 위해 대상 물체의 모델 데이터를 구 좌표계로 표현한다. 직각 좌표계에서 모델 데이터는 식 (1)을 이용하여 구 좌표계로 변환되며, 여기서 θ, ϕ, R은 고도각(zenith: 0~180°), 방위각(azimuth: 0~360°), 구 중심에서 해당 정점까지의 거리를 의미한다. 직각 좌표계와 구 좌표계 상에서 표현된 모델 데이터를 그림 2에 각각 나타내었다. 복잡한 형태의 물체는 동일 방위각과 고도각의 위치에서도 여러 개의 표면 데이터가 존재할 수 있으며, 이 경우에는 R에 의해 구분된다.

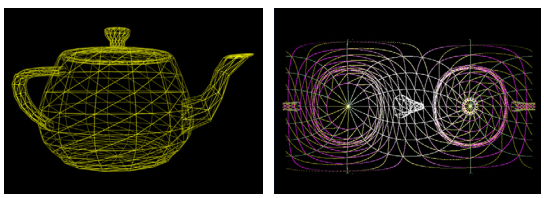


그림 2. 물체 모델(위)과 구 좌표계 상에서의 표현(아래)

제안된 방법에서 물체의 표면을 구성하는 삼각형 데이터를 검색하기 위해 고도각과 방위각을 기준으로 삼각형 표면의 무게중심 위치, 삼각형 색인, 메모리 접근 맵 등의 3개의 자료구조를 생성한다.

$$
\begin{aligned}
&x = R \cdot sin(\theta)cos(\phi),\\
&y = R \cdot sin(\theta)sin(\phi),\\
&z = R \cdot cos(\theta)\\
&R = \sqrt{x^2+y^2+z^2},\ \theta = arctan(\frac{\sqrt{x^2+y^2}}{z}),\\
&\phi = arctan(\frac{y}{x})
\end{aligned}
\qquad (1)
$$

2.2 자료구조 구성

렌더링의 기본적인 기하 요소는 삼각형이며, 제안된 방법에서는 정점들이 구성하는 삼각형의 무게중심값을 계산해 구 좌표계 상에 저장한다. 즉, 물체 표면을 구성하는 삼각형의 무게 중심의 위치를 거리값 및 각도(R, θ, ϕ)에 따라 무게중심 맵에 저장하고 같은 각도에 존재하는 여러 면들은 거리 정보에 따라 정렬된다.

카메라의 시점에서부터 임의의 방향으로 진행하는 시선 벡터가 물체의 경계구(bounding sphere)와 교차하면, 해당 영역에 대한 삼각형 색인 맵을 검색한다. 이 과정에서 물체의 가시 영역을 빠르게 계산하기 위해 구 좌표인 θ, ϕ 공간에 삼각형의 위치 정보를 색인 맵에 저장한다. 이 자료구조에는 삼각형의 무게중심을 기준으로 세 개의 정점으로 구성된 내부 영역을 라인 스캐닝(scanning)하여 현재의 삼각형이 몇 번째인지를 의미하는 고유의 색인정보가 저장된다.

카메라의 가시 영역을 검사하여 물체의 구 좌표상에 특정 영역을 선택한 다음, 여기에 해당하는 삼각형 정보를 그래픽 파이프라인 상에 효율적으로 업로드하기 위해 메모리 접근 맵을 생성한다. 여기에는 해당 정점의 주소값이 저장되며, 정점에 대한 효율적인 데이터에 접근하도록 정점배열(vertex array)과 VBO(Vertex Buffer Object) 기술을 이용한다.

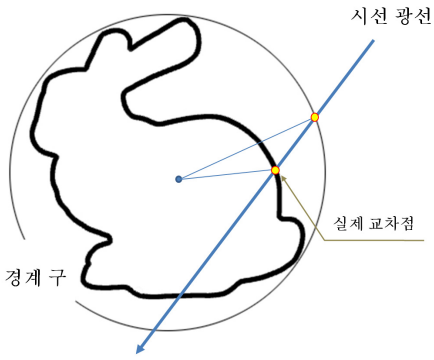


그림 3. 광선의 교차점 계산과정

모델 데이터에 추가적으로 생성되는 세 개의 자료구조를 다음 표 1에 정리하였다. 실험에서 대상으로 하는 토끼(bunny)모델은 144,046개의 면과 72,027개의 정점으로 구성되어 있으며 모델 데이터의 크기는 약 10M 바이트이다. 제안된 방법에서 무게중심 및 색인 맵의 용량은 각각 4.3M, 6.7M이고, 메모리 접근 맵은 생성하는 상황마다 동적으로 할당되며 약 0.51M의 저장공간이 요구된다. 따라서 제안된 방법은 모델 데이터 만큼의 자료구조가 추가로 필요하지만, 이는 그래픽스 하드웨어의 발전에 따라 충분히 지원할 수 있는 수준이라고 볼 수 있다.

표 1. 자료구조의 구성

자료구조	삼각형 무게중심 맵	삼각형 색인 맵	메모리 접근 맵
용량	float형*4* 면 개수	float형*2*360* 180*면 개수	unsigned int형* 4*360*180
내용	- 현재 위치의 데이터 개수 - 데이터 참조용 시작포인트 - 빈 공간인 경우, 예외처리 위해 앞 위치의 참조 데이터 - 현재까지 총 데이터 개수	- 삼각형 색인 데이터 - 무게중심 위치의 거리	- 현재 위치의 데이터 시작 주소 - 현재 위치의 데이터 개수 - 빈 공간인 경우, 예외 처리를 위한 앞, 뒤 위치의 참조

2.3 은면 제거

본 절에서는 앞에서 구성된 자료구조를 이용하여 은면(hidden surface)을 제거하는 과정을 기술한다. 카메라에 의해 대상 물체의 일부 영역만 보이는 경우 시각 절두체를 구성하는 4개의 모서리 벡터와 물체의 경계구 사이에 교차 관계를 확인한다. 교차점의 위치를 계산한 다음, 삼각형 색인 맵을 이용해서 물체의 가시 영역에 해당하는 표면의 삼각형을 찾는다. 색인 맵은 구좌표계 상에 삼각형 정보를 가지고 있으므로, 경계구 및 물체 표면과 시선 광선이 교차될 때의 거리값을 색인 맵에서 각각 비교하여 실제 교차가 이루어진 삼각형의 위치를 판단한다. 이 과정을 그림 3에 나타내었으며, 교차된 삼각형을 서로 연결하여 구 좌표 상에서 물체의 가시영역을 확인한다.

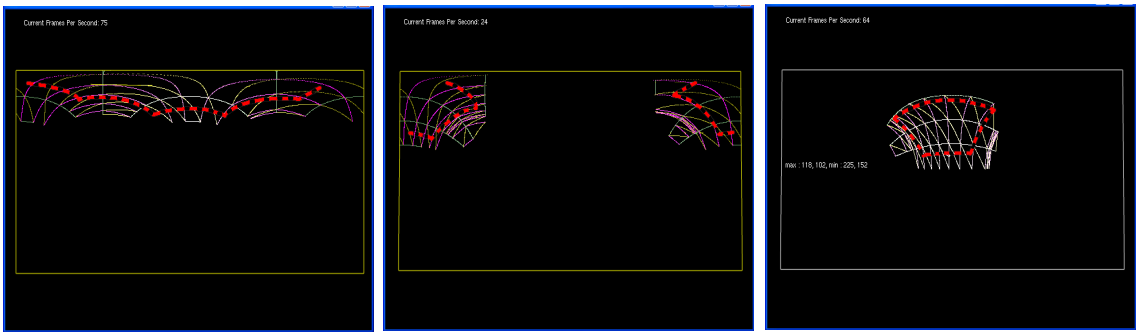


그림 4. 색인 맵에서 물체의 가시영역 분포(경우 1, 2, 3)

위의 과정을 통해 판단된 가시 영역은 물체의 구 좌표계의 표현에서 양쪽으로 넓게 퍼지거나, 중간에 뭉치거나 또는 양쪽으로 각각 분리되는 등의 세 가지 형태로 분포한다. 색인 맵 상의 가시 영역 위치를 붉은 점선으로 그림 4에 나타내었다. 다양한 가시 영역에 존재하는 삼각형 데이터에 효과적으로 접근하기 위해 해당 영역의 최대 및 최소값을 이용한 경계 박스를 구성한다.

판단된 가시 영역을 구성하는 실제 정점 데이터에 대한 접근은 사전에 구성된 메모리 접근 맵을 이용한다. 메모리의 스캔 횟수는 처리 시간에 많은 영향을 주기 때문에, 본 논문에서는 한번의 스캔으로 보다 많은 데이터를 접근하도록 θ 구간을 기준으로 한다. 그림 4의 첫 번째 경우는 θ에 대해서 ϕ 영역이 중간에 끊어짐 없이 연결되어 있으므로 모든 삼각형의 정보를 한번에 그래픽스 파이프라인으로 업로드할 수 있다. 나머지 두 경우는 θ에 대한 ϕ의 최소 및 최대 위치 영역을 단계별로 접근한다. 이 과정을 통해 물체의 가시 영역에 해당하는 정점만을 고려할 수 있다.

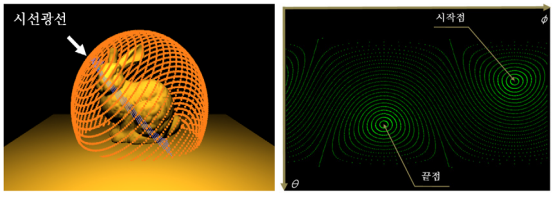


그림 5. 물체 영역 샘플링(위)과 구좌표 표현에서의 양끝점(아래)

카메라의 4개의 시선벡터가 물체와 교차하지 않는 경우, 먼저 물체가 시각 절두체 내에 존재하는지를 확인한다. 가시 영역내에 물체가 존재하면, 시선 광선을 기준으로 렌더링되지 않는 물체의 은면(뒷면)을 제거한다. 카메라의 시점에서부터 임의의 시선 광선이 물체의 경계구와 교차하는 시작과 끝 위치를 확인한다. 시작 위치에서부터 일정 θ, ϕ 단위로 경계구를 샘플링하고, 이 구간에 대한 구좌표 상에서의 표현을 그림 5에 나타내었다. 해당 샘플링 구간은 구 좌표 맵에서 등고선 형태로 분포하며 시작점과 끝점은 경계구에서 정반대에 위치하므로 일정한 각도 차이(θ = 180°)를 갖는다.

그림 6. 샘플링 영역에서 사영성분 계산

구 좌표계 상에서 표현된 샘플링 영역에 해당하는 삼각형 정보는 색인 맵으로부터 얻어진다. 각 샘플링된 영역에 존재하는 정점이 가려지는지 여부를 판단하기 위해 샘플링 구간마다 시선 광선에 수직하는 축을 설정한다. 그리고 분할된 각 영역에서 삼각형의 무게중심에 대해 원점으로부터의 거리를 확인하고 이를 수직축에 사영(projection)하여 시선 벡터에 대한 상대적인 크기 정보(S)를 계산한다. 즉, 사영된 성분이 다른 영역보다 상대적으로 큰 영역이 작은 면들을 가리며, 일정 단위로 검색을 진행한다. 시선벡터에 대해 180° 영역까지는 가려짐이 없는 경우로 간주하여 렌더링 알고리즘이 적용된다. 그리고 경계구의 180° 이후 부분에서는 샘플링된 각 영역에서 계산된 사영정보의 크기를 비교하여 가려짐 여부를 판단한다.

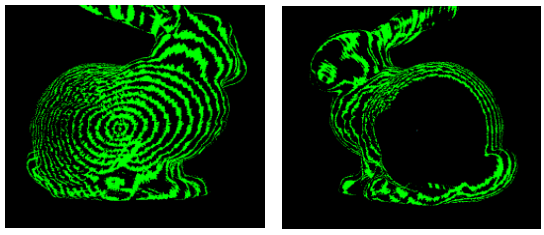


그림 7. 샘플링된 영역의 표현과 은면 제거

은면 제거를 위한 사영성분의 계산과정을 그림 6에 나타내었으며, 샘플링된 영역에 따른 등고선 분포와 이를 고려해 은면을 제거한 결과를 실제 물체에 그림 7과 같이 표시하였다. 시작 위치부터 은면으로 확인된 영역에 해당하는 정점만을 이후 렌더링과정에서 고려한다. 제안된 방법의 슈도(pseudo) 코드를 부록 1에 정리하였다.

3. 실험 및 검토

제안된 알고리즘의 성능을 검증하기 위해 Intel Zeon CPU 2.33Ghz, RAM 4GB, Nvidia Quadro FX5500 그래픽스 하드웨어가 장착된 컴퓨터에서 토끼 모델을 대상으로 실험했으며, 윈도우 XP 기반의 C++와 OpenGL, Nvidia 社의 Cg 셰이더(shader) 언어[10]를 이용했다. 가시 영역에 대한 고려를 하지 않은 경우와 제안된 알고리즘의 성능을 디스플레이 되는 윈도우의 수, 해상도, 물체와 카메라 영역의 모양에 따라 각각 비교하였으며, 렌더링 성능의 직접적인 비교를 위해 람버트 쉐이딩 모델을 기반으로 하였고 그림자 등은 고려하지 않았다.

카메라의 시각 절두체가 물체와 교차해서 구 좌표 맵상에는 그림 4와 같이 세가지 경우가 발생하며, 순서대로 경우 1, 2, 3으로 각각 표시하였다. 각각의 경우에서 내부의 자료 구조를 접근하는 방식이 서로 다르기 때문에 렌더링 속도가 다르다. 320×240 크기의 윈도우에서 카메라의 가시 영역에 해당하는 면의 개수와 렌더링 성능을 표 2에 나타내었다. 렌더링에서 고려되는 면의 개수가 144,078개에서 약 13,000~26,000개로 크게 감소하였음을 확인하였다. 또한 메모리의 접근 방식에 따라 각 경우에서 약간의 차이가 존재하지만, 제안된 방법은 가려진 면들을 고려하지 않는 방식에 비해 상대적으로 약 100fps(frame per second) 이상의 개선된 성능을 보인다.

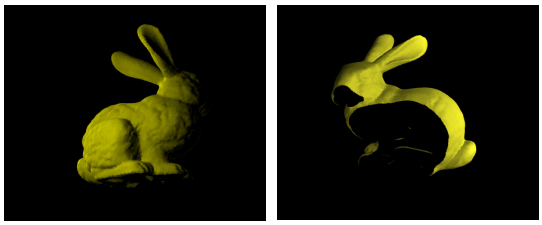


그림 8. 은면 제거된 렌더링 영상(앞면과 뒷면)

4개의 윈도우에 대해 해상도를 320×240, 640×480으로 설정하여 얻어진 렌더링 성능을 표 3에 나타내었다. 해상도가 증가함에 따라 레스터라이즈되는 화소 개수가 많아지기 때문에 상대적으로 렌더링 속도가 감소하였다. 표 4는 320×240 크기의 윈도우를 동시에 각각 4개, 8개씩을 렌더링할 때의 성능을 나타내며, 모든 경우에서 실시간 렌더링이 가능함을 확인하였다.

표 2. 단일 위도우에서 렌더링 결과 비교

	경우 1		경우 2		경우 3	
	고려되는 면 개수	fps	고려되는 면 개수	fps	고려되는 면 개수	fps
은면 미고려	144,078	295	144,078	300	144,078	295
제안된 방법	13,508	486	16,434	432	26,308	389

표 3. 해상도에 따른 렌더링 성능 비교

	320×240 윈도우의 fps			640×480 윈도우의 fps		
	경우 1	경우 2	경우 3	경우 1	경우 2	경우 3
은면 미고려	80	84	80	63	65	63
제안된 방법	376	421	202	153	174	136

표 4. 윈도우 수에 따른 렌더링 성능 비교

	4개 윈도우에서의 fps			8개 윈도우에서의 fps		
	경우 1	경우 2	경우 3	경우 1	경우 2	경우 3
은면 미고려	70	70	71	21	21	21
제안된 방법	206	286	161	95	32	75

제안된 방법에 의해 렌더링된 앞면 영상과 은면이 제거된 뒷면 영상을 그림 8에 나타내었다. 카메라의 시점에 따라 경계구 영역을 단계별로 샘플링하며, 각 영역을 구 좌표 맵 상에 표현하면 그림 5와 같이 등고선 형태로 분포한다. 시선 벡터와 경계구의 샘플링에 의해 은면 영역을 판단하며 어느 영역까지를 고려하는가에 따라 렌더링 성능이 영향받는다. 따라서 장면 및 물체를 BVH 등의 가속화 구조로 구성하면, 가려지는 영역을 보다 효과적으로 검색할 수 있을 것으로 예상된다.

카메라의 시각 절두체에 대상 물체가 완전히 포함되면, 시선벡터에 대해 경계구의 샘플링과 수직축에 대한 삼각형 무게중심의 사영성분 크기를 비교하여 은면을 결정한다. 정점배열과 그래픽스 하드웨어에서 정점 계산의 효용성을 높이는 기술인 VBO 방법을 적용하고, Bunny 모델을 640×480 크기의 윈도우에서 제안된 방법으로 렌더링한 결과를 표 5에 나타내었다. 가려지는 면을 판단하는 과정에서 물체의 구조에 따라 고려되는 정점의 수가 가변적이기 때문에 10 개 이상의 시점에서 렌더링 성능을 조사하여 최대, 최소 및 평균 fps를 나타내었다.

표 5. 정점배열과 VBO 방법에 따른 성능 비교

	정점 배열 방법 (fps)			VBO 방법 (fps)		
	최대	최소	평균	최대	최소	평균
가려짐 미고려	134	134	134	700	700	700
제안된 방법	199	137	170	828	738	780

4. 결 론

본 논문에서는 직교 좌표의 물체 정점 데이터를 구 좌표로 변환하는 새로운 실시간 렌더링 알고리즘이 제안되었다. 모델 데이터에 대한 효율적인 처리를 위해 각 삼각형의 무게중심 위치와 색인, 그리고 해당 정점의 메모리 접근 자료구조를 각각 구성한다. 그래픽 하드웨어에 기반한 렌더링 파이프라인 상에서 물체의 모든 정점을 고려하지 않고, 카메라의 가시 영역에 해당하는 표면과 정점만을 고려하는 알고리즘을 제안함으로써 렌더링 퀄리티의 저하 없이 전체 렌더링 소요시간이 크게 단축됨을 실험결과로부터 확인하였다. 이후 연구에서는 구 좌표계 표현을 BVH 등의 가속화 방법과 결합하여 장면 및 물체의 자료구조를 구성함으로써 보다 효율적인 렌더링 알고리즘을 개발하고, 복잡한 장면을 대상으로 제안된 방법의 활용 범위를 분석할 예정이다.

참고문헌

[1] J. W. Ratcliff, “Sphere trees for visibility culling, ray tracing, and range searching,” Game Programming Gems 2, Charles River Media, pp. 384-387, 2001.

[2] H. Samet, “The design and analysis of spatial data structures,” Addison-Wesley, Reading, Massachusetts, 1989.

[3] T. Akenine-Möller, E. Haines, and N. Hoffman, “Real-time rendering,” A.K. Peters Ltd., 2002.

[4] D. Cohen-Or, Y. Chrysanthou, C. T. Silva, and F. Durand, “A survey of visibility for walkthrough applications,” IEEE Trans on Visualization and Computer Graphics, Vol. 9, No. 3, pp. 412-431, 2003.

[5] J. H. Clark, “Hierarchical geometric models for visible surface algorithms,” Communications of the ACM, Vol. 19, No. 10, pp. 547-554, 1976.

[6] J. F. Blinn and M. E. Newell, “Texture and reflection in computer generated images,” Communications of the ACM, Vol. 19, No. 10, pp. 542-547, 1976.

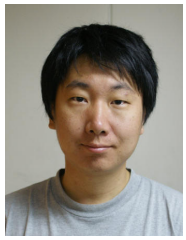
[7] R. Fernando and M. J. Kilgard, “The Cg tutorial: the definitive guide to programmable real-time graphics, Addison-Wesley Professional, 2003.

[8] V. Havran, M. Smyk, G. Myszkowski, and H. Seidel, “Interactive system for dynamic scene lighting using captured video environment maps,” Proc. of Eurographics Symposium on Rendering, pp. 31-42, 2005.

[9] D. Shreiner, M. Woo, J. Neider, and T. Davis, “OpenGL programming guide,” 5th Ed., pp. 56-57, Addison-Wesley Ltd., 2005.

[10] L. Szirmay-Kalos, B. Aszódi, I. Lazányi, and

M. Premecz, “Approximate ray-tracing on the GPU with distance impostors,” Computer Graphics Forum, Vol. 24, No. 3, pp. 695-704, 2005.



한은호 (Eun-Ho Han)

2006년 3월～현재 중앙대학교 첨단영상대학원 첨단영상학과 석사과정

관심분야 : 컴퓨터그래픽스

홍현기 (Hyun-Ki Hong)

1998년 8월 중앙대학교 전자공학과 박사
1998년 9월～1999년 8월 서울대학교 자동제어특화 연구센터 연구원
1999년 9월～2000년 2월 중앙대학교 정보통신연구소 연구교수
2002년 2월～2003년 1월 Univ. of Colorado at Denver post-Doc.
2000년 3월～현재 중앙대학교 첨단영상대학원 첨단영상학과 부교수 재직 중.

관심분야 : 컴퓨터그래픽스, 컴퓨터비전, 증강현실 등

부록 1. 제안된 알고리즘의 슈도 코드

```
for each object
{
    intersect_check = intersect frustum 6 planes with bounding sphere
    if( intersect_check )
    {
        calculate frustum 4 ray;
        for each frustum ray
        {
            intersect point = intersect bounding box;
            intersect triangle = Triangle_Search(INDEX_MAP, intersect point);
        }
        minmax, state = Visibility_Area_Test(intersect triangle);
        Render_State(MEM_MAP, state, minmax);
    }
    else if( intersect_check is include )
    {
        intersect_point = intersect camera position with bounding sphere;
        sampling_position = Area_sampling(intersect_point);
        hidden_position = Hidden_Surface_Test(INDEX_MAP, sampling_position);
        Render_Selected(MEM_MAP, hidden_position);
    }
}
```