한국산학기술학회논문지
Vol. 10, No. 8, pp. 1843-1849, 2009

실시간 3D 그래픽을 위한 렌더링 상태 변경 비용 감소 기법

김석현[1*]
[1]남서울대학교 멀티미디어학과

Rendering States Changing Costs Reducing Technique for Real-time 3D Graphics

Seok-Hyun Kim[1*]
[1]**Deparment of Multimedia, Namseoul University**

요 약 실시간 3D 그래픽에서 렌더링 성능을 향상 시키는 기법 중 하나로 파이프라인 최적화가 있다. 파이프라인 최적화는 버퍼 재정렬 문제로 볼 수 있다. 그러나 이는 NP-hard이다. 따라서 최적의 해를 근사하며 실시간 3D 그래픽에 적합한 알고리즘 개발이 필요하다. 본 논문은 이를 위해 실시간 3D 그래픽의 렌더링 상태 변화 비용의 패턴을 분석하였다. 그리고 분석된 패턴을 기반으로 렌더링 상태 변화 비용이 큰 것에 대하여 우선적으로 정렬하는 알고리즘을 제시한다. 제안 기법의 우수함을 보이기 위해, 상태비용을 정렬하지 않는 알고리즘과 성능을 평가한다. 제안 방법은 상태비용을 정렬하지 않는 알고리즘에 비해 약 2.5배-4배 정도 비용이 감소되며. 특정 렌더링 상태의 변화 비용이 크게 증가할수록 우수함을 보인다.

Abstract In real-time 3D Graphics, pipeline optimization is one of techniques enhancing rendering performance. Pipeline optimization is kind of buffer reordering problem, but it is NP-hard. Therefore techniques that is approximating optimal solution and suitable for real-time 3D graphics are needed. This paper analyze pattern of rendering states changing costs for real-time 3D graphics, and based on this, the algorithm that brings rendering states into line by changing costs is proposed. The proposed technique shows good performance enhancement when costs of some rendering states are much higher than others. Proposed technique shows 2.5 to 4 times better performance than non-ordering algorithm and becomes more faster when rendering costs of a state gets higher.

Key Words : Real-time 3D graphics, Rendering States

1. 서론

컴퓨터 그래픽은 다양한 영역에서 널리 사용되고 있다. 특히 실감나는 3D 그래픽은 게임, 애니메이션 등의 고부가가치 콘텐츠 산업은 물론 의료 영상, 지리정보시스템, CAD 등 광범위한 분야에 널리 활용되고 있다. 3D 그래픽은 사실적이고 화려한 이미지를 제공할 수 있으나 많은 수치 계산을 요구하므로 다양한 가속화 기법이 필요하다.

알고리즘 차원에서 렌더링 속도를 향상시키기 위한 다양한 가속화 알고리즘(Acceleration algorithms)이 연구되었다[1]. 공간 자료 구조 (Spatial data structures)를 기반으로 적용되는 다양한 컬링(Culling)기법은 화면에 렌더링 하는 폴리곤 중 카메라에 보이지 않는 것을 빠르게 찾아내어 그리지 않음으로써 렌더링을 가속화 하는 알고리즘이다[2-6]. LOD(Level of detail)와 같은 기법은 실시간에 카메라에서 멀리 떨어져서 작게 보이는 물체의 폴리곤 수를 줄임으로써 렌더링 속도를 향상시키는 방법이다[7].

또 다른 3D 그래픽 가속화 기법은 파이프라인 최적화

이 논문은 2008학년도 남서울대학교 학술 연구비 지원에 의하여 연구되었음.
*교신저자 : 김석현(shkim@nsu.ac.kr)
접수일 09년 04월 05일 수정일 (1차 09년 07월 27일 2차 09년 08월 17일) 게재확정일 09년 08월 19일

(Pipeline optimization)이다. 3D 그래픽 파이프라인은 어플리케이션에서 제공한 3차원 장면(Scene) 데이터를 단계적으로 처리하여 최종적으로 이차원 모니터 화면에 표시되는 영상을 만들어 내는 과정이다. 어플리케이션은 그래픽 파이프라인에 3차원 장면 데이터를 넘겨주게 된다. 파이프라인 최적화는 어플리케이션이 제공한 3차원 영상 데이터의 순서를 파이프라인이 최대한 효율적으로 렌더링 가능하도록 변환하는 최적화 기법이다.

어플리케이션은 3D API를 통해 그래픽 파이프라인에 다양한 데이터를 보내게 된다. 어플리케이션의 필요에 따라 서로 다른 텍스처(Texture), 머티리얼(Material), 버텍스 버퍼(Vertex buffer) 등의 다양한 데이터가 함께 렌더링 되도록 설정 된다. 이와 같이 다양한 렌더링 상태(Rendering state)를 설정한 다음 렌더링 함수를 호출 하면 호출 시점까지 그래픽 파이프라인에 설정된 렌더링 상태가 화면에 렌더링 된다. 표 1은 다양한 렌더링 상태가 설정되고 이것이 렌더링 함수 호출을 통해 렌더링 되는 과정을 추상적으로 보여준다.

[표 1] 3D 장면 렌더링을 위한 API 사용

```
...
set various rendering states
call rendering function
...
set various rendering states
call rendering function
...
```

렌더링 상태를 설정하고 렌더링 함수를 호출하는 과정을 반복할 때 성능을 위해 함수 호출 순서가 중요해 진다. 표 2는 렌더링 API 호출 순서에 따른 렌더링 상태를 보여준다. 똑같은 내용을 렌더링 하지만 API 호출 순서에 따라 렌더링 상태의 변화가 다르다. 호출2의 경우 상태 (A, B)가 연달아 나오므로 실제로 상태를 바꿀 필요가 없어져서 상태 변경 오버헤드가 적어진다.

렌더링 상태 변화 최소화는 버퍼 재정렬 문제(reordering buffer problem)의 관점에서 설명할 수 있다[8]. 버퍼 재정렬 문제는 서비스를 원하는 요청(request)이 버퍼에 쌓여 있을 때, 이 요청들의 처리 순서를 결정하여 전체 비용을 최소화하는 문제이다. 전체 요청을 처리하는 비용은 각 요청들을 처리하는 비용에 요청들의 처리 순서에 따른 상태 변화 비용의 총합으로 구할 수 있다.

본 논문은 파이프라인 내의 렌더링 요청의 순서를 정렬을 통해 변경함으로써 실시간 3D 그래픽의 성능을 향상 시킬 수 있는 알고리즘을 제시한다. 렌더링 요청을 그래픽 하드웨어의 파이프라인으로 보내기 전에 렌더링 상태 변경 비용이 큰 상태에 대하여 우선적으로 정렬한다. 3장에서 서술한 바와 같이 렌더링 상태의 종류에 따라 비용의 차이가 크기 때문에 이렇게 변경 비용이 큰 상태에 대하여 우선적으로 정렬하면 전체 상태 변경 비용을 감소시킬 수 있다.

논문의 구성은 다음과 같다. 2장에서 관련 연구를 기술하고, 3장에서 렌더링 상태 변화 비용 감소를 위한 알고리즘을 제안하고, 4장에서 시뮬레이션 결과를 통해 알고리즘의 성능을 평가한다. 마지막으로 5장에서 결론을 맺는다.

[표 2] API 호출 순서에 따른 렌더링 상태 변화

API 호출1	렌더링 함수 호출 시 렌더링 상태
set state A, B	
call rendering function	states (A, B)
set state C	
call rendering function	states (C)
set state A, B	
call rendering function	states (A, B)
API 호출2	**렌더링 함수 호출 시 렌더링 상태**
set state A, B	
call rendering function	states (A, B)
set state A, B	
call rendering function	states (A, B)
set state C	
call rendering function	states (C)

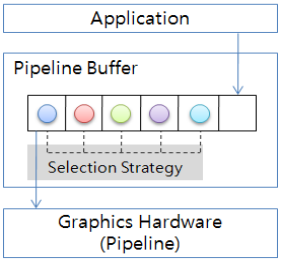
2. 관련 연구

실시간 3D 그래픽의 성능에 있어서 중요한 변수는 현재 화면에 렌더링 되고 있는 폴리곤의 수이다. 화면상의 어떤 폴리곤들은 렌더링 되지 않아도 최종 결과에 영향을 주지 않는다. 예를 들어 다른 물체에 가려져 있는 물체의 폴리곤은 그래픽 파이프라인에 보내지지 않아도 시각적으로 아무런 문제가 없다. 이렇게 카메라에 보이지 않는 폴리곤을 빠르게 찾아서 그래픽 파이프라인에 보내지 않는 알고리즘은 오랫동안 연구되어온 분야이다[2,3,5,6].

그래픽 파이프라인 최적화 문제의 중요성은 많이 알려져 있지만[9-12] 상대적으로 연구가 덜 되어 있는 분야이

다. 파이프라인 최적화 문제에서 각 요청을 그래프의 정점(Vertex)으로, 상태 변화를 간선(Edge)으로 생각하면 모든 정점을 방문하면서 간선에 할당되어 있는 가중치의 총합을 최소화 하는 문제가 된다. 이는 해밀턴 경로 문제(Hamiltonian path problem)에서 최소의 가중치(weight)를 구하는 문제가 되며 NP-hard이다[8]. 그래서 버퍼 재정렬 문제는 N개의 전체 입력에 대해 버퍼의 크기를 k로 놓고 현재 버퍼에 들어있는 요구들만을 고려하는 온라인 알고리즘에 대해 논한다[8,13].

[14]는 3D 그래픽 파이프라인에 렌더링 요청들을 보내기 전에 이 요청들을 버퍼에 넣고 정렬함으로써 성능을 향상시키는 개념을 제안 하였다. 그림1은 파이프라인 버퍼의 개념을 보여준다. 어플리케이션이 파이프라인에 보낸 렌더링 요청은 선택 전략(selection strategy)에 따라 순서가 변경되어 그래픽 하드웨어(graphics hardware) 내부 파이프라인으로 보내진다.



[그림 1] 파이프라인 버퍼

[13]은 버퍼에 쌓여있는 요청 p_i, p_j에 대해 두 요청 사이의 변환 비용 또는 거리가 |i-j|로 정의되는 경우, 요청의 수가 N이라 할 때 $i, j \in \mathbb{N}$이면 O(logN), $i, j \in \mathbb{R}$이면 O(logN loglogN)에 최적 순서를 찾는 알고리즘을 제시하였다. 이 알고리즘을 잘 적용할 수 있는 대표적인 경우는 디스크에서 seek time을 최소화하는 문제이다. 디스크 암(disk arm)이 찾고자 하는 실린더 번호가 주어지면 디스크 암의 이동 시간은 실린더 번호우, 요에 비례하므로[13]을 적용하면 최적 시간을 얻을N)에 최적 그러나 본 논N이서 다루는 상태 변화의 경우 비용 함수를 이용 또는 단순하게 산출할)에 없으므로 다른 방식을 적용해야 한다.

[13]이 제한된 거리공간(metric space) 상에서의 알고리즘을 다룬 것에 비하여 [8]은 일반적인 거리공간을 그 대상으로 하였다. 이를 위해 먼저 트리(tree) 형태로 요청이 들어오는 경우 O(Dlogk)의 성능이 가능함을 보였다. D는 가중치 없는 트리의 지름(diameter)이고 k는 버퍼의 크기 이다. 다음으로 균형이 잘 맞는 트리에 대하여 $O(\log^2 k)$임을 보이고 트리 상의 거리에서 확률적 근사로 일반적 거리를 구하는 방법을 사용하여 최종적으로 일반적인 거리 공간에 대해 $O(\log^2 k \log n)$ 성능의 알고리즘을 유도 하였다. 실시간 그래픽은 초당 30프레임 정도의 빠른 렌더링 연산을 필요로 한다. [8]의 방법은 요청 사이의 거리를 구하기 위해 트리를 만들고 다시 일반적인 거리 공간으로 변환하는 과정을 거치는 다단계의 방식이기 때문에 빠른 연산을 요구하는 실시간 그래픽에 바로 사용하기에는 전처리 과정이 무겁다고 할 수 있다.

실제 렌더링 환경에서 렌더링 상태 변화의 비용을 단순하게 고정된 값으로 산출할 수 없다. Direct3D에서 정확하게 API 호출의 비용을 산출하는 것은 어려운 일이다. 실시간 렌더링은 CPU와 GPU의 병렬 수행으로 이루어지는데 API 호출 시간을 측정하는 방법은 단지 CPU의 작업만을 측정한다는 문제가 있다. 게다가 API 호출에 대한 CPU의 작업 시간은 Direct3D 런타임(Runtime)과 비디오카드 드라이버에서 사용된 시간의 합인데, 같은 API 함수 호출이라도 드라이버의 종류에 따라 측정 시간이 달라진다[9].

API 함수의 종류에 따라 절대적인 호출 시간을 명확히 측정하기는 어렵지만, API 함수의 종류에 따라 상대적으로 소요되는 시간은 큰 차이가 있다. 표3은 [9]에서 측정한 API 함수에 따라 소요되는 CPU 사이클(cycle) 중 일부를 보여준다. 따라서 이 비용들을 될수록 감소시키는 알고리즘 개발이 필요하다.

[표 3] API 호출에 따른 평균 사이클 수

API 호출	평균 사이클 수
SetVertexDeclaration	6500 - 11250
SetFVF	6400 - 11200
SetTransform	3200 - 3750
SetScissorRect	150-340

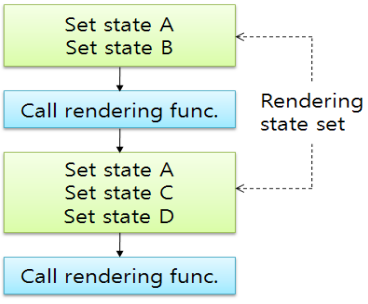
3. 제안된 렌더링 상태 변경 비용 감소 기법

본 논문에서 제안하는 기법은 렌더링 상태 변경 비용이 렌더링 상태의 종류에 따라 큰 차이를 보인다는 사실에 기반 한다. 표 3과 같이 렌더링 상태에 따라 최대 20배 정도의 사이클 차이가 발생한다. 또한 4.1절에서 수행한 실험을 보면 렌더링에 이용되는 데이터의 위치에 따라 큰 비용 차이가 생김을 알 수 있다.

렌더링 상태에 따른 변화 비용이 크게 다르다면 비용

이 큰 렌더링 상태의 변경을 최대한 줄이는 것이 유리하다. 이를 위해 본 논문에서 제시하는 기법은 실시간에 렌더링 상태의 변화 비용이 큰 렌더링 상태를 결정하고, 이러한 상태의 변화가 가능한 적도록 한다.

3D API는 다양한 함수들로 이루어져 있다. 이 함수들 중 실제 렌더링을 시작시키는 그리기(Draw) 함수를 제외한 다른 함수들은 상태 변화를 유발한다. 그림 3과 같이 그리기 함수들 사이에 끼어있는 일련의 함수들을 렌더링 상태 집합(RSS, Rendering state set)이라 하자. 어떤 RSS 다음에 나오는 그리기 함수는 바로 앞의 RSS에서 결정된 상태를 화면에 렌더링 하게 된다.



[그림 3] RSS(Rendering state set)

렌더링 상태 변경 비용 감소 기법은 렌더링 상태 집합의 순서를 변경하여 변경 비용이 큰 상태에 대하여 변화가 적게 일어나도록 한다. 렌더링 상태 변경 비용 감소 알고리즘은 전체적으로 표 4와 같은 단계로 이루어진다.

[표 4] 렌더링 상태 변경 비용 감소 알고리즘

1. 렌더링 상태 변화 비용 추정
2. 고비용 렌더링 상태 결정
3. 렌더링 상태 집합에 고비용 렌더링 상태들을 기준으로 이진수 아이디 부여
4. 3에서 부여된 이진수 아이디를 기준으로 전체 렌더링 상태 집합 정렬

렌더링 상태 변화 비용은 동적으로 변화할 수 있기 때문에 실시간에 동적으로 렌더링 상태에 따른 비용을 측정할 필요가 있다. 이 비용은 표 3에서의 API 호출 비용과 데이터가 저장된 위치에 의해 크게 영향을 받는다. 즉 데이터가 메모리에 있는 경우와 디스크에 있는 경우의 로딩 비용은 크게 차이가 발생한다. 식 (1)은 렌더링 상태 s의 변화 비용을 나타낸다. s는 상태 비용을 표현하고 F(s)와 L(s)는 각각 API 함수 호출 비용과 데이터 로딩 비용을 의미한다.

$$C(s) = F(s) + L(s) \quad (1)$$

F(s)는 s의 종류에 따라 제한된 범위 내에서 변화하므로 상수라 가정한다. 그러면 전체 렌더링 상태 변화 비용의 변화량은 데이터 로딩 비용의 변화와 같다.

고비용 렌더링 상태 집합 HCS(High Cost rendering States)는 다음과 같이 정의된다.

정의 4.1. HCS는 렌더링 상태들을 교체 비용에 따라 내림차순으로 정리했을 때 비용이 가장 큰 것에서부터 비용을 누적하여 처음으로 미리 정해놓은 한계값 t보다 커지게 하는 렌더링 상태까지로 한다.

|HCS|=n이라면 HCS 내에서 교체 비용이 가장 큰 렌더링 상태는 s_0, 가장 작은 상태는 s_{n-1}이 된다. 이렇게 정의된 HCS를 기준으로 각 렌더링 상태 집합에 이진수 아이디를 부여한다. 만약 s_0에서 s_{n-1}을 모두 가지고 있다면 n개의 연속된 1을 아이디로 가지게 된다. 표 5는 HCS를 기준으로 각 렌더링 상태에 아이디를 부여하는 알고리즘을 보여준다.

이렇게 각 RSS에 아이디를 부여하고 아이디를 기준으로 정렬하면 렌더링 상태 교체 비용이 큰 렌더링 상태에 대하여 우선적으로 정렬이 된다.

[표 5] RSS의 이진수 아이디 부여 알고리즘

```
for each set_i in RSS
  id = 0
  for each s_j in HCS (j=0, 1, ... n-1)
    if s_j ∈ set_i then
      id = id | 1
      id << 1
  (id of set_i) = id
```

예를 들어 RSS의 아이디가 각각 111, 100, 101, 110, 001, 010인 경우를 살펴보자. 내림차순으로 정렬하면 111, 110, 101, 100, 010, 001이 된다. 이 순서에서 s_0는 한번, s_1은 세 번, s_2는 네 번 바뀌게 된다.

RSS가 2^n개이고 |HCS| = n이며 모든 RSS가 서로 다른 경우에 대해 렌더링 상태 변경 비용을 계산해 보자. RSS가 모두 다르므로 아이디는 00...0에서 11..1까지 2^n개가 존재한다. 이를 크기순으로 정렬하면 s_0는 한 번, s_1은 3번, s_2는 7번, ..., s_{n-1}은 2^n-1번 바뀐다. 따라서 이 경우 최종적인 렌더링 상태 변경 비용 C_{total}은 식 (2)와 같은 방법

으로 구할 수 있다.

$$C_{total}=\sum_{i=0}^{n-1}(2^{i+1}-1)C(s_i) \quad (2)$$

일반적인 경우 s_i의 변경 수를 정확히 도출하기 어렵기 때문에 C_{total}을 구하는 일반식을 결정하기 어렵다. 본 논문에서 제시하는 알고리즘은 렌더링 상태 변경 비용이 가장 큰 상태 s_0에 대하여 단 한 번의 상태 변경이 발생하는 것을 보장한다. RSS의 아이디로 정렬하면 MSB가 1인 RSS가 한 쪽으로 모두 모이기 때문이다. i번째로 비용이 큰 상태 s_i의 변경 회수를 정확히 결정할 수는 없지만 정렬에 의해 렌더링 상태 변경 비용이 클수록 변경 횟수가 적어짐을 보장한다.

만약 전체 렌더링 상태 변경 비용이 일부 고비용 상태에 의하여 크게 좌우된다면 본 논문에서 제시하는 기법은 큰 성능 향상을 보이게 된다. 그러나 각 렌더링 상태의 변경 비용이 큰 차이가 없는 경우 본 논문의 기법은 성능 향상을 기대하기 어렵다.

4장의 실험은 특정 렌더링 상태의 변경 비용이 커지는 경우가 일반적으로 발생함을 보여준다. 또한 4장의 시뮬레이션은 특정 렌더링 상태의 변경 비용이 큰 경우 큰 성능 향상이 가능함을 보인다.

4. 시뮬레이션 결과

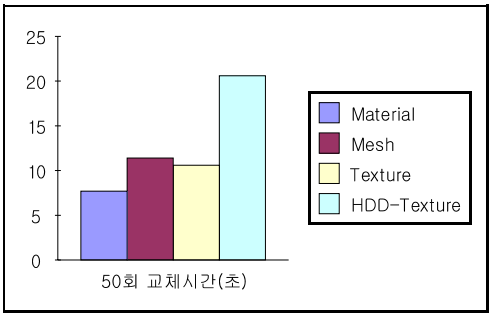
본장에서는 렌더링 상태 비용의 특징을 관찰하고 제안 방법의 우수함을 보이기 위해 성능을 평가한다.

4.1 렌더링 상태 비용 패턴 분석

렌더링 상태 변화 비용의 특성을 관찰하기 위한 실험을 실시하였다. 렌더링 데이터로 5000개의 랜덤 폴리곤 집합, 텍스처, 머티리얼을 각각 두 개씩 생성한다. 이를 각각 폴리곤-A/B, 텍스처-A/B, 머티리얼-A,B라 하자. 이상의 세 가지 렌더링 상태 각각에 대한 교체 비용을 측정하기 위하여 한 가지 상태만 바꾸면서 렌더링 시간을 측정 한다. 예를 들어 텍스처 교체 비용 측정을 위해 폴리곤-A, 머티리얼-A를 계속 사용하면서 텍스처만 텍스처-A, 텍스처-B를 교대로 사용한다. 실험에 사용한 환경은 다음과 같다. Intel Pentium Dual CPU 2.20GHz, RAM 2.0GB, Windows Vista, Visual C++ 2008 Express, DirectX SDK August 2007을 사용하였다.

각 실험은 렌더링 50회에 대한 시간 측정이며 이 실험을 3회 반복하여 평균을 구한다. 실험 결과는 그림 2와 같다. 머티리얼(Material), 메쉬(Mesh), 텍스처(Texture) 항목은 앞에서 기술한 방식에 따라 측정 하였으며 HDD-텍스처는 텍스처를 설정할 때 항상 디스크에서 읽어오도록 만든 결과이다.

실험 결과 머티리얼은 7.7초, 텍스처는 11.4초, 메쉬는 10.6초, HDD-텍스처는 20.6초가 소요 되었다. 이처럼 렌더링 상태가 변화할 때 데이터를 읽어오는 위치에 따라 상태 변경 비용이 달라진다. 비디오 메모리에 모든 데이터가 상주할 수 없기 때문에 메인 메모리나 디스크에서 읽어 오는 경우가 생기게 된다. [9]에서 언급하고 있는 API 함수 종류에 따른 비용 차이와 함께 데이터를 읽어오는 위치에 따른 차이를 고려하면 렌더링 상태에 따른 교체 비용은 종류에 따라 그 차이가 더욱 커진다고 할 수 있다. 이 결과를 통해 상태 변화 최적화 알고리즘을 설계할 때 모든 상태 변화를 똑같은 비중으로 다루지 않고 상태 변화의 비용이 큰 것에 집중하는 것이 상태 교체 비용 감소에 도움을 줄 수 있다.



[그림 2] 렌더링 상태에 따른 비용 측정

결과적으로 본 실험 결과와 표 3을 통해 렌더링 상태 변화 비용의 특징은 다음의 두 가지로 요약할 수 있다.

렌더링 상태의 종류에 따른 상대적인 비용 차이가 존재한다.

같은 렌더링 상태 변화라도 데이터를 읽어오는 위치에 따라 비용이 달라질 수 있다.

4.2 성능 평가 시뮬레이션

제안 알고리즘의 성능을 평가하기 위하여 랜덤하게 생성된 RSS의 집합과 알고리즘을 적용하여 정렬한 RSS에 대하여 전체 렌더링 상태 변경 비용의 차이를 측정한다. 앞에서 언급한 바와 같이 렌더링 상태들의 비용 차이가 크지 않으면 제안 알고리즘으로는 성능 향상을 기대할 수 없다. 시뮬레이션은 렌더링 상태의 비용 차이가 큰 경우 전체 비용의 향상 정도를 측정한다.

HCS의 수는 두 실험 모두 3으로 설정하여 시뮬레이션을 수행 한다. 상태 변경 비용이 낮은 상태의 비용은 1에서 10사이의 값으로, 변경 비용이 큰 세 가지 상태의 비용은 1에서 10사이의 랜덤 값을 설정한 다음 20을 더하여 21에서 30사이가 되도록 하였다. 이는 4.1절에서 관찰한 패턴과 같이 렌더링 상태에 따라 교체 비용이 큰 차이가 발생하는 것을 반영하기 위함이다.

첫 번째 실험은 다른 모든 변수를 고정시키고 RSS의 개수를 증가시키면서 논문에서 제시한 알고리즘에 의한 성능을 평가 한다. 두 번째 실험은 RSS의 수를 고정시키고 각 RSS의 렌더링 상태 수를 증가 시키면서 성능 개선을 평가한다.

[표 6] RSS 개수의 변화에 따른 비용변화

RSS 개수	40	80	120	160	200
제안 알고리즘	334	553	643	979	1213
not-sorted	921	1801	2694	3701	4588

[표 7] 상태 개수에 따른 비용변화

스테이트 개수	5	10	15	20	25
제안 알고리즘	334	672	1097	1352	1685
not-sorted	1179	1384	1776	2114	2447
기법 간 비용 차이	845	712	679	762	762

표 6은 첫 번째 실험의 결과이다. 렌더링 상태의 수는 5이고, HCS는 3개이다. 이런 상황에서 RSS의 수를 각각 40 ~ 200으로 하면서 전체 상태변화 비용을 비교 한다. not-sorted는 랜덤하게 생성된 상태 그대로 측정한 방법이다. 제안 알고리즘의 상태 비용은 not-sorted에 비해 약 3배에서 4배 감소되었다. 이는 상태 변경 비용이 큰 HCS의 변화 수를 정렬에 의하여 줄였기 때문이다.

표 7은 두 번째 실험 결과이다. 첫 번째와 반대로 RSS의 수는 50으로 고정하고 RS의 수를 5~25로 증가시키면서 비용 변화를 비교한다. 이 때 HCS의 수는 3으로 고정한다. 제안 알고리즘의 상태 비용은 not-sorted에 비해 약 2.5배에서 4배 감소하였다. 주목할 만한 사실은 두 가지 방법의 렌더링 상태 비용의 차이가 전체적으로 거의 일정하다는 것이다. 이는 제안 알고리즘의 성능 향상이 HCS를 기준으로 한 정렬에 의해 이루어지기 때문이다. 제안 알고리즘에 의한 성능 향상은 HCS의 상대적인 렌더링 상태 변경 비용이 클 때 이루어진다. 따라서 스테이트의 수가 증가하면 HCS에 의한 성능 향상이 전체 렌더링 상태 변경 비용에서 차지하는 비율이 점점 줄어들게 된다. 스테이트의 수가 계속 증가하면 결국 HCS 정렬에 의한 성능 향상은 무시할 수 있을 정도가 된다.

본 논문에서 제시한 방식은 전체 렌더링 상태 중에서 비용이 큰 일부 상태에 집중하여 비용을 감소시키는 방식이다. 따라서 정렬의 기준이 되는 상태의 수가 중요하다. 정렬의 기준이 되는 상태의 수가 전체 렌더링 상태의 수에 비해 높은 비율을 차지하는 첫 번째 실험의 경우 본 논문이 제시한 기법은 상당히 좋은 결과를 보여준다. 그러나 정렬의 기준이 되는 상태의 수가 한정적인 두 번째 실험과 같은 경우 본 논문이 제시한 최적화 기법은 렌더링 상태 변경 비용의 감소에 한계를 보여준다. 그러나 특정 상태 변화의 비용이 다른 상태 변화의 비용보다 큰 경우가 실시간 3D 그래픽에서 많이 있으므로 본 논문에서 제시한 기법을 통해 렌더링 성능의 향상을 꾀할 수 있다.

5. 결론 및 추후 연구방향

실시간 3D 그래픽에서 렌더링 상태 변경 비용을 줄이는 것은 전체 렌더링 성능에 많은 영향을 준다. 본 논문에서는 렌더링 상태 변경 비용이 특정한 상태 변화에서 매우 크게 나타날 수 있기 때문에 높은 변경 비용을 나타내는 렌더링 상태를 기준으로 정렬함으로써 전체적인 렌더링 성능을 향상하는 알고리즘을 제안하고 성능을 평가하였다. 제안 알고리즘은 특정 렌더링 상태 변경 비용이 전체 비용에서 차지하는 비중이 높은 경우 큰 성능 향상이 가능하고 실험 결과 약 2.5배~4배의 성능 향상을 볼 수 있었다.

추후에는 실시간에 특정 렌더링 상태의 비용을 정확하게 측정할 수 있는 방법을 연구하여 본 논문에서 제안한 알고리즘을 보완할 예정이다.

참고문헌

[1] T.Akenine-Möller, E.Haines and N.Hoffman, “Real-time rendering 3rd edition”, ISBN 987-1-56881-424-7, A.K. Peters Ltd., 2008.

[2] A.S. Winter, “An Investigation into Real-Time 3D Polygon Rendering Using BSP Trees”, 1999, http://citeseer.ist.psu.edu/winter99investigation.html.

[3] H.Zhang. “Effective Occlusion Culling for the Interactive Display of Arbitrary Models”. Ph.D. thesis, Department of Computer Science,

UNC-Chapel Hill, 1998.

[4] Wikipedia, "Hidden surface determination", http://en.wikipedia.org/wiki/Frustum_culling#Viewing_frustum_culling.

[5] W.Jane and V.G.Allen, "Octrees for faster isosurface generation", ACM Transactions on Graphics, pp. 201-227, 1992.

[6] J.L. Bentley, "Multidimensional binary search trees used for associative searching", Communications of the ACM 18 9, pp. 509-517, September 1975.

[7] P.Lindstrom, D.Koller, W.Ribarsky, L.Hodges, N.Faust and G.A.Turner, "Real-time, continuous level of detail rendering of height fields", In Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques, ACM, New York, NY, pp. 109-118, 1996.

[8] M. Englert, H. R□acke, and M. Westermann, "Reordering buffers for general metric spaces", In Proceedings of 39th Symposium on Theory of Computing (STOC), pp. 556-564, 2007.

[9] MSDN, "Accurately profiling Direct3D API calls", http://msdn.microsoft.com/en-us/library/bb172234(VS.85).aspx.

[10] Panda3D team, "Performance monitoring: features", http://panda3d.org/features.php.

[11] Direct3D Help, "Performance Optimizations (Direct3D 9)", DirectX Documentation for c++, DirectX SDK Agust 2007.

[12] ATI Help, "Rendering States - Designing for ATI Rage 128 and Rage 128 Pro", http://ati.amd.com/developer/sdk/rage128sdk/Design.html.

[13] I. Gamzu and D. Segev, "Improved online algorithms for the sorting buffer problem", In Proceedings of the 24th Symposium on Theoretical Aspects of Computer Science (STACS), pp. 658–669, 2007.

[14] J.Krokowski, H.Räcke, C.Sohler and M.Westermann, "Reducing state changes with a pipeline buffer", In Proceedings of the Vision Modeling and Visualization 2004 (VMV), Stanford, USA, pp. 217-224, November 2004.

김 석 현(Seok-Hyun Kim) **[정회원]**

- 2001년 2월 : 서울대학교 재료공학부 (공학학사)
- 2008년 2월 : 서울대학교 컴퓨터공학부 (공학석사)
- 2008년 3월 ~ 현재 : 서울대학교 컴퓨터공학부 박사과정
- 2008년 3월 ~ 현재 : 남서울대학교 멀티미디어학과 전임강사

<관심분야>
운영체제, 시스템소프트웨어, 보안, 컴퓨터그래픽