Vulkan 을 이용한 메시 기반의 광선추적기

김지온*, 신병석
*인하대학교 컴퓨터공학과
e-mail : 3161508@gmail.com

Mesh-based ray tracing system using Vulkan

Ji-On Kim*, Byeong-Seok Shin
*Dept of Computer Science, Inha University

요 약

이 논문에서는 Graphics 시스템이 점차 광선추적법(Ray-tracing) 기반으로 전환되고 있는 점과, OpenGL, Direct3D 등의 3 차원 그래픽스 API 가 삼각형 메시를 주로 사용하는 점에서 착안하여 크로노스 그룹의 차세대 그래픽 표준인 Vulkan API 를 이용하여 광선추적기를 개발하였다. 여기에 삼각형 메시를 적용하여 성능평가를 수행하였다.

1. 서론

최근 광선추적법(ray-tracing) 기반 시스템의 지속적인 개발과 하드웨어 성능의 진보에 힘입어서, 이전에 비해서 훨씬 복잡한 장면을 처리할 수 있는 광선추적 기법이 지속적으로 개발되고 있다. NVIDIA 등의 여러 기업들이 광선추적법 기반의 소프트웨어, 하드웨어를 지속적으로 개발하고 있다. 하드웨어로는 Geforce RTX 그래픽 카드와 소프트웨어로는 Optix API 를 예로 들 수 있다. 본 연구에서는 광선추적법을 기존의 그래픽스 API 인 OpenGL, Direct3D 와 같이 현재 범용적인 파이프라인 구조에서 사용되는 삼각형 메시 형태로 데이터를 받아옴으로써, 이후에 파이프라인 구조에 맞춘 최적화가 가능하도록 했다. 시스템의 개발을 위해서 오버헤드가 적고 최적화에 유리한 Vulkan API 를 사용하였다.

2. 관련연구

광선추적법은 각 픽셀들의 색상 값을 3 차원 장면 상에서 이동하는 광자(photon)의 움직임에 기반하여 계산하는 기법이다. 광자는 광원을 기점으로 출발하여 해당 3-차원장면 상의 여러 물체들과 교차한 뒤에 최종적으로 관측평면(image plane)을 거쳐서 눈으로 들어오는데 이 궤적을 광선(ray)이라고 한다. 특정 광선과 물체 간의 교차점들의 정보와 해당 교차점의 물체 정보들을 분석하고 결합하여 최종적으로 해당 광선과와 관측평면이 교차하는 픽셀의 색상 값을 계산하게 된다. Vulkan API 는 Graphics, Compute 장치를 위한 프로그래밍 인터페이스이다. Vulkan API 는 OpenGL 등 기존의 API 와는 다르게 디버깅, 메모리 관리 등 기존에 시스템상에서 자동적으로 관리해주는 부분들을 전부 사용자가 관리 해야 한다는 특징을 가지고 있다. 따라서 시스템의 매우 자세한 부분까지 사용자가 제어할 수 있지만 디폴트 설정이 존재하지 않기 때문에 모든 설정을 사용자가 정해주어야 하기 때문에 코드가 장황해지며, 시스템 부분도 API 상에서 관리해 주지 않기 때문에 작은 변화에도 프로그램이나 시스템 상에 큰 에러로 이어지는 프로그램이 만들어 질 수도 있다.

Möller–Trumbore 교차 알고리즘은 광선이 삼각형과 교차했을 때, 해당 교차점을 전처리 과정 없이 적은 저장 공간만 활용해서 계산할 수 있게 해주는 알고리즘이다. 삼각형과 광선의 교차식은 다음과 같다.

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2 \quad (1)$$

여기서 u,v 는 무게중심 좌표를 의미하며 $u \geq 0$, $v \geq 0$, $u+v \leq 1$ 를 만족해야 한다. 그리고 이식을 Creamer's rule 과 선형대수를 기반으로 정리하여 교차점을 계산하는 식은 아래와 같다.

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix} \quad (2)$$

$$P = (D \times E_2)$$
$$Q = (T \times E_1)$$

여기서 위 식에서 나온 인자들을 재사용함으로써 계산 속도를 향상시킨다.

3. 광선 추적기 구현

3. 1. 파일 파싱

.obj 파일 포맷은 Wavefront Technologies 에서 개발된 공개된 모델 파일 포맷이며, 모델의 정점 좌표, 법선 벡터등의 정보들이 저장되어 있다. 우선 파일을 파싱하여 이를 시스템 내에서 사용할 수 있는 삼각형 메시 데이터로 변환시킨다. 여기서 파일에 저장되어 있

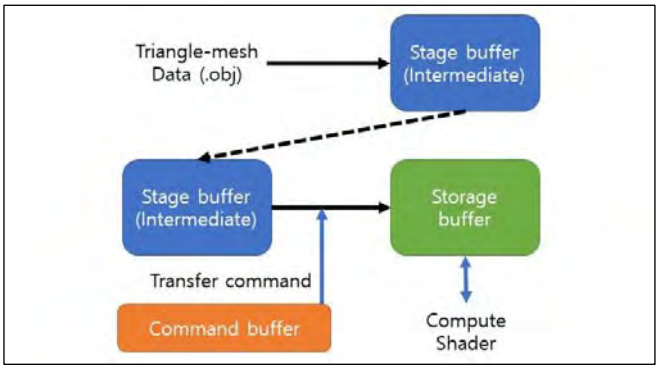
는 데이터 중 일부는 다음과 같이 표시된다.

1. v %f %f %f
2. vn %f %f %f
3. f %d/%d/%d %d/%d/%d %d/%d/%d (%d/%d/%d)
(%f 는 실수 값, %d 는 정수 값을 의미한다.)

여기서 첫번째 줄은 정점의 로컬 좌표를 의미하며 3 개의 실수 값을 저장하고 있다. 그리고 두번째 줄은 정점의 법선 벡터를 의미하며, 3 개의 실수 값을 저장하고 있다. 그리고 세번째 줄은 세 정점의 인덱스를 기반으로 구성된 삼각형의 인덱스 정보를 저장하고 있다. 여기서 법선 벡터 인덱스가 생략될 수 있으며, 생략될 경우 정점 정보를 기반으로 법선 벡터를 계산한다. 여기서 %d%d%d 묶음이 3 개 혹은 4 개 존재할 수 있는데, 3 개 존재할 경우, 3 개의 정점 좌표, 법선 벡터의 인덱스 정보로 1 개의 삼각형을 표현하게 되고, 묶음이 4 개 존재할 경우, 왼쪽에서 1, 2, 4 번째, 그리고 2, 3, 4 번째 정점, 법선 벡터의 인덱스 정보로 2 개의 삼각형을 표현한다.

3. 2. 셰이더로의 데이터 전달

데이터를 파싱해서 삼각형 메시 기반의 데이터를 생성한 뒤에 이를 compute 셰이더로 보내주게 되는데 이 과정은 다음과 같다. 이 과정에서 Vulkan API 가 사용된다.

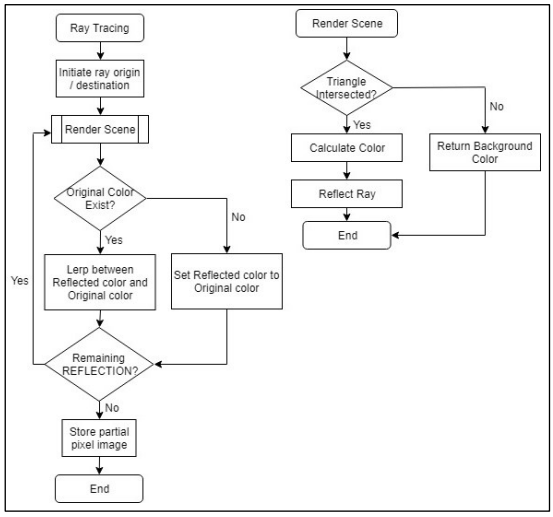


(그림 1) 버퍼간 데이터 전달 과정

이 과정에서 최종적으로 삼각형 메시 정보를 저장하는 저장 공간 버퍼(storage buffer)를 만든다. 이 버퍼는 삼각형 데이터가 모두 전달된 이후에 이를 셰이더로 보내 연산을 수행하게 된다. 우선 파일로부터 삼각형 메시 데이터를 받아온 이후에, 중간 버퍼(stage buffer)를 생성하여 삼각형 데이터를 임시로 저장한다. 그리고 이를 최종적으로 셰이더와 연결되어 있는 결과 버퍼(destination buffer)를 생성하고, 중간 버퍼의 데이터를 커맨드 버퍼(command buffer) 내의 커맨드를 이용하여 최종적으로 옮긴다. 이로써 셰이더에서 삼각형 관련 정보를 가져와서 사용할 수 있게 된다.

3. 3. Compute 셰이더 내에서의 광선 추적기 구현

이제 compute 셰이더 내에 삼각형 메시 정보를 이용할 수 있게 되었으므로, 이를 이용해서 광선추적법을 셰이더에서 실행하게 되는데, 과정은 아래와 같다.



(그림 2) 광선추적법 순서도

우선 광선의 시작점과 끝점을 각각 카메라의 위치, 각 관측 평면의 픽셀 좌표의 중심점으로 잡는다. 그리고 해당 광선과와 삼각형 메시의 각 삼각형과 교차하는 지 확인한다. 그리고 교차할 경우에, 해당 교차점의 색상 값을 계산한다. 그리고 교차점 계산 시에 Möller–Trumbore 교차 알고리즘을 이용하였다. 색상 값을 계산하기 위해, 해당 삼각형 메시의 디퓨즈(diffuse), 스패큘러(specular) 값을 기반으로 Blinn-Phong 모델을 이용하였다. 그리고 해당 삼각형과의 교차점을 새로운 광선의 원점으로 설정하고, 삼각형의 법선 벡터를 기준으로 반사된 방향을 광선의 방향으로 새로 정한다. 어느 삼각형과도 교차하지 않을 경우 기본 배경색인 검은색을 반환한다. 교차점의 색상 값을 계산한 이후에는 이전 광선에서 계산된 기존의 색상값(Original Color)이 존재할 경우, 해당 기존의 색상값과 현재 광선에서 계산된 반사된 색상값(reflected color)을 일정 비율로 보간해서 최종 색상을 결정한다. 그리고 남은 반사 수만큼 교차, 색상값 계산을 반복한다.

4. 결과

CPU 는 Intel(R) Core™ i7-4770, GPU 는 NVIDIA GeForce GTX 1080 Ti 를 사용하였다. 운영체제는 Windows 10 Pro 64 비트이고 GPU 의 동작 속도는 3.4GHz 이며, 메모리는 16GB DRAM 이다.
외부에는 6 개의 평면이 정사면체 형태로 둘러싸여 있으며, 해당 평면들은 광선을 반사 시켜 특정 물체를 비치게 하는 스패큘러 특성을 가지고 있다. 그리고 중앙에 각자 다른 정점 수를 가진 모델을 배치시키고, 각각 다른 해상도에 따라 초당 프레임 수(fps)를 측정한다. 그리고 동일한 모델의 수를 늘려가면서 초당 프레임 수를 측정한다. 단일 모델의 경우에는 모두 일정한 크기, 위치를 가지며, 다중 모델의 경우 각 모델간의 거리가 결과에 영향을 줄

수 있으므로, 원점의 물체와 x, y, z 축을 중심으로 일정한 거리만큼 떨어진 위치에 일정한 크기의 모델을 추가하는 방식을 이용한다. 그리고 위와 같은 장면의 경우에는 광선 반사가 2 회를 넘어가도 품질상에 큰 차이가 없기 때문에 편의상 2 회로 일정하게 설정하였다. 그리고 광원은 관측 평면이 만든 공간 내에 특정 궤도를 돌고 있는 형태로 구현되어 있다.

기본적으로 해상도가 각 축마다 2 배씩 총 4 배 증가시켰을 경우 거의 모든 모델의 초당 프레임 수는 선형적으로 3~4 배의 비율로 감소하는 것을 확인할 수 있다. 그리고 각 모델들의 정점, 삼각형 면의 수와 초당 프레임 수를 서로 비교하면 정점, 삼각형 면 수가 증가할수록 초당 프레임 수의 감소폭이 점점 줄어드는 것을 확인 할 수 있었다. 그리고 최종적으로 이 시스템에서 30 fps 의 실시간으로 불러올 수 있는 최대 삼각형 면 수는 512x512 의 해상도의 경우에는 약 450 개 전후, 1024x1024 의 경우에는 200 개 전후임을 확인할 수 있었다. 동일 모델에서 수를 증가시키는 경우에도 해상도가 4 배 증가함에 따라 초당 프레임 수도 3~4 배 정도 감소하는 것을 확인할 수 있다. 그리고 모델을 처음 추가시킬 때엔 초당 프레임 수가 2 배 가까이 차이가 나고 그 뒤에는 모델을 증가시킴에 따라 그 차이가 줄어드는 것을 확인할 수 있다.

5. 결론

본 논문에서는 Vulkan 기반의 광선추적법 시스템을 삼각형 메시 기반으로 구현하였다. 파일을 파싱하여 가져온 삼각형 메시 데이터들을 Compute 셰이더에 보내고, 색상 값을 가져오고 이를 이전의 색상 값과 보간하여 관측 평면의 픽셀의 색상 값을 계산하여 최종적으로 하나의 장면을 그려낼 수 있었다. 이를 셰이더 내에서 Möller–Trumbore 교차 알고리즘을 기반으로 한 광선과 삼각형 간 교차를 통해 더 많은 정점과 삼각형 면을 가진 모델들을 최소 30 fps 에 실시간으로 그려 내기 위해서 KD, BVH 등의 구조체를 이용하여 더 빠르고 효율적으로 삼각형들을 횡단하는 것. 그리 고 삼각형이 집중된 픽셀에 광선을 집중시켜서 해당 장면에서 중요한 부분을 자세하게 그려낼 수 있는 것. 그리고 스무딩, 안티에일리어싱을 통해서 거친 테두리 표면을 부드럽게 만들어 주는 것. 그리고 텍스쳐 매핑, 환경 매핑을 이용해서 좀 더 현실적인 장면을 표현할 수 있는 시스템으로 만드는 것이 앞으로 개선해야 할 내용이다.

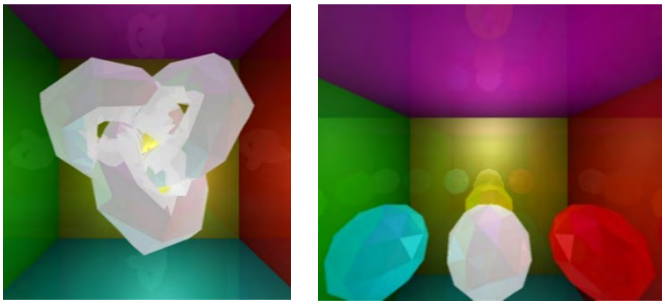
참고문헌

[1] AS Glassner. “An introduction to ray tracing”, pp 5~8, 1989.

[2] Graham Sellers, and John Kessenich “Vulkan Programming Guide: The Official Guide to Learning Vulkan” pp 2~3, 2016

[3] T Möller, B Trumbore “Fast, minimum storage ray/triangle intersection” ACM SIGGRAPH 2005 Courses, pp 1~7, 2005

[4] K McHenry, P Bajcsy “An overview of 3d data content, file formats and viewers” National Center for Supercomputing Applications, pp 6, pp 12~13, 2008



(그림 5) 단일 모델, 다수 모델의 광선 추적 결과

<표 1> 다른 정점, 삼각형 면 수를 가진 모델들의 해상도에 따른 성능 측정 결과

			Resolution	FPS	ms/frame
geosphere	vertices	42	512x512	250	4.52
	facet	80	1024x1024	72	14.58
torusknot	vertices	256	512x512	50	20.91
	facet	512	1024x1024	14	77.42
sphere	vertices	482	512x512	27	38.96
	facet	960	1024x1024	8	143.36
cylinder	vertices	600	512x512	22	48.12
	facet	1200	1024x1024	6	200.5

<표 2> 동일 모델의 수와 해상도에 따른 성능 측정 결과

geosphere	num	Resolution	FPS	ms/frame
model	1	512x512	250	4.52
vertices	2		140	7.69
42	3		98	10.81
facet	4		76	13.83
80	1	1024x1024	72	14.58
	2		40	26.14
	3		27	38.96
	4		21	50.5