

간단한 조명 샘플 (DX12)

*이 샘플은 Windows 10 기념일 업데이트 SDK (14393)*

# 설명

이 샘플은 정적 및 동적 Lambertian 조명을 사용하여 인덱싱 된 형상을 그리는 정적 Direct3D 12 버텍스, 인덱스 및 상수 버퍼를 만드는 방법을 보여줍니다.

이 샘플에서는 두 개의 조명 (흰색과 빨간색 중 하나)에 의해 큐브로 표현되는 대형 큐브를 렌더링합니다. 빨간색 조명이 중앙 큐브 주위를 도는 동안 흰색 조명은 고정되어 있습니다. 중앙 큐브도 회전합니다. 모션을 사용하면 각기 다른 각도에서 컬러 조명의 효과를 관찰 할 수 있습니다.



# 샘플 사용하기

|  |  |  |
| --- | --- | --- |
| 액션 | 게임 패드 | 키보드 |
| 종료 | 보기 버튼 | Esc |

# 구현 정보

## 셰이더

## 이 샘플은 정점 셰이더 ("TriangleVS")와 두 개의 픽셀 셰이더 ("LambertPS", "SolidColorPS")를 사용하여 장면을 렌더링하는 세 개의 셰이더를 사용합니다. 컴파일 된 셰이더 블롭을 CreateDeviceDependentResources에 로드한 다음 각 셰이더 조합에 대한 파이프 라인 상태 개체를 만들 때 그것을 참조합니다. 모든 셰이더는 동일한 HLSL 포함 파일 "SimpleLighting.hlsli"에 정의되어 있으며, 세 개의 스텁 셰이더에는 이 파일이 포함되어 있습니다. 각 스텁 쉐이더는 세 개의 쉐이더 블롭을 생성하기 위해 다른 진입점에 대해 컴파일됩니다.

## 파이프 라인 상태 객체 (PSO)

Simple Lighting 샘플에는 두 가지 고유한 셰이더 조합이 있습니다. 첫 번째는 TriangleVS를 LambertPS와 결합한 것이고 두 번째는 TriangleVS이며 SolidColorPS입니다. DirectX 12에서는 모든 고유한 셰이더 조합에 대해 파이프 라인 상태 개체 (PSO)를 만들어야 합니다. 이름에서 알 수 있듯이 PSO는 특정 셰이더 집합을 사용하여 여러 번의 그리기 호출에 필요한 모든 파이프 라인 상태를 캡슐화합니다. PSO는 루트 서명, 파이프 라인의 다양한 스테이지에 대한 셰이더, 래스터 라이저 상태, 깊이 스텐실 상태, 블렌드 상태 등과 같은 상태 설정을 결합합니다. 자세한 내용은 MSDN의 설명서를 참조하십시오.

## 루트 서명

루트 서명은 그래픽 파이프 라인에 바인딩되는 리소스 유형과 리소스 배치 방법을 정의합니다. 루트 서명은 API 함수 서명과 유사하며 매개 변수의 유형, 매개 변수 순서 및 레이아웃을 설명하지만 실제 매개 변수 인스턴스는 정의하지 않습니다. 루트 매개 변수는 루트 서명의 요소에 해당하는 실제 데이터 인스턴스입니다. 샘플의 버텍스 셰이더는 셰이더 상수에 대한 단일 구조만 있으면 되므로 루트 서명이 매우 간단합니다. 루트 서명은 ConstantBufferView 유형의 단일 루트 매개 변수를 포함합니다.

## 기하학

## 장면의 지오메트리는 정육면체의 6 개 쿼드를 나타내는 24 개의 정점에 대한 데이터를 포함하는 정적 정점 및 인덱스 배열로 구성됩니다. 2 개의 배열은, Sample::CreateDeviceDependentResources의 내부에서 선언되고 있어, ID3D12Device::CreateCommittedResource에 의해 즉시 사용되어 버퍼의 ID3D12Resources를 작성합니다. 간단하게 하기 위해 샘플에서는 D3D12\_HEAP\_TYPE\_UPLOAD를 사용합니다. 이렇게 하면 각 리소스를 한 번에 데이터로 초기화하면서 만들 수 있습니다. 그러나 \_UPLOAD 힙은 기하학 데이터에 적합하지 않은 위치입니다. 보다 효율적인 구현은, 기하학 데이터에 D3D12\_HEAP\_TYPE\_DEFAULT를 사용합니다. \_DEFAULT 힙을 초기화하려면 \_UPLOAD 힙을 사용해야 하기 위해 두 개의 힙을 사용하게 되므로 구현이 복잡해집니다.

지오메트리의 버퍼가 창조되면, 샘플은 정점에 대한 D3D12\_VERTEX\_BUFFER\_VIEW 및 인덱스에 대한 D3D12\_INDEX\_BUFFER\_VIEW를 생성할 수 있습니다. 뷰는 ID3D12GraphicsCommandList::IASetVertextBuffer 및 ID3D12GraphicsCommandList::IASetVertextBuffer를 호출하여 입력 어셈블러를 설정할 때 Sample::Render에서 사용됩니다.

## 셰이더 상수 관리하기

이 매우 간단한 장면에서 모든 셰이더 상수는 다음을 포함하는 하나의 상수 버퍼로 함께 정리됩니다.

* 월드, 뷰 및 투영 행렬
* 가벼운 방향과 색상
* 단색

보다 복잡한 장면의 경우 상수가 얼마나 자주 업데이트 되는 지에 따라 상수를 여러 버퍼로 분할합니다.

대형 큐브와 빨간색 조명이 애니메이션 되므로 쉐이더 상수는 그리기 호출 사이에서 프레임 당 여러 번 업데이트해야 합니다. 루트 서명에는 각 그리기 호출에 사용할 셰이더 상수의 복사본을 참조해야 하는 ConstantBufferView 유형의 단일 루트 매개 변수가 포함되어 있습니다. CPU와 GPU가 병렬로 작동하기 때문에 CPU는 GPU가 이를 사용하여 완료될 때까지 상수 버퍼를 업데이트하려고 시도하면 안됩니다. 하나의 상수 버퍼만 있는 경우 CPU는 GPU가 그리기를 마칠 때까지 블록해야 합니다. 이는 몇 가지 그리기 호출을 위해 상수를 업데이트해야 하므로 비현실적입니다. 따라서이 샘플에서는 여러 상수 버퍼를 사용하여 GPU가 그리는 동안 CPU가 GPU에 상수를 계속 보낼 수 있습니다.

샘플은 간단하며 프레임 당 고정된 수의 그리기 호출을 가지며, 그리기 호출 횟수는 컴파일 타임에 알려져 있습니다. 이 샘플에서는 각 그리기 호출에 대해 하나의 버퍼와 스왑 체인의 백 버퍼 수를 곱합니다. 이 수는 CPU가 쓸 수있는 여유 버퍼가 항상 있음을 보장합니다. 모든 상수 버퍼는 Sample::CreateDeviceDependentResources에서 생성된 단일 연속 업로드 버퍼에 저장됩니다. 업로드 버퍼는 CPU 주소 공간과 GPU 주소 공간에 대한 기본 메모리 주소를 얻기 위해 즉시 매핑됩니다.

Sample::Render 메서드에서 상수는 상수 업로드 힙의 CPU 기본 주소에서 인덱싱된 위치에 기록됩니다. 동일한 인덱스가 GPU 주소와 결합되어 ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView에 대한 호출을 통해 버퍼를 파이프 라인에 바인딩합니다. 인덱스는 백 버퍼 생성과 그리기 횟수를 고려해야 합니다. 자세한 내용은 Sample::Render 구현을 참조하십시오.

## CPU/GPU 동기화

CPU가 GPU가 명령리스트를 처리할 수 있는 것보다 빨리 GPU에 명령리스트를 발행할 수 있다면 결국 CPU는 GPU가 처리를 완료할 때까지 기다려야 합니다. 이 샘플에서는 고정 버퍼 메모리를 라운드 로빈 방식으로 사용합니다. 이는 고정된 기간 후에 버퍼 슬롯을 다시 사용한다는 것을 의미합니다. 일반적으로 공유 리소스를 재사용하기 전에 리소스가 사용 중이 아닌지 확인하기 위해 일종의 동기화 전략을 사용하는 것이 중요합니다. 이 샘플에서는 ID3D12Fence를 사용하여 CPU와 GPU를 동기화합니다. CPU는 명령 대기열에 명령을 삽입하여 프레임 색인 값으로 차단 개체에 "신호"를 보냅니다. 제공된 프레임 인덱스 값은 신호 명령이 GPU에서 처리되 자마자 CPU에 표시됩니다. 이렇게 하면 CPU가 현재 프레임 인덱스와 GPU가 신호한 마지막 프레임 인덱스를 비교하여 GPU가 CPU와 얼마나 뒤져 있는지 확인할 수 있습니다. GPU 프레임 수와 CPU 프레임 수의 차이가 백 버퍼 수를 초과하면 CPU가 대기해야 합니다.

# 업데이트 기록

초기 출시 2018 년 4 월.