

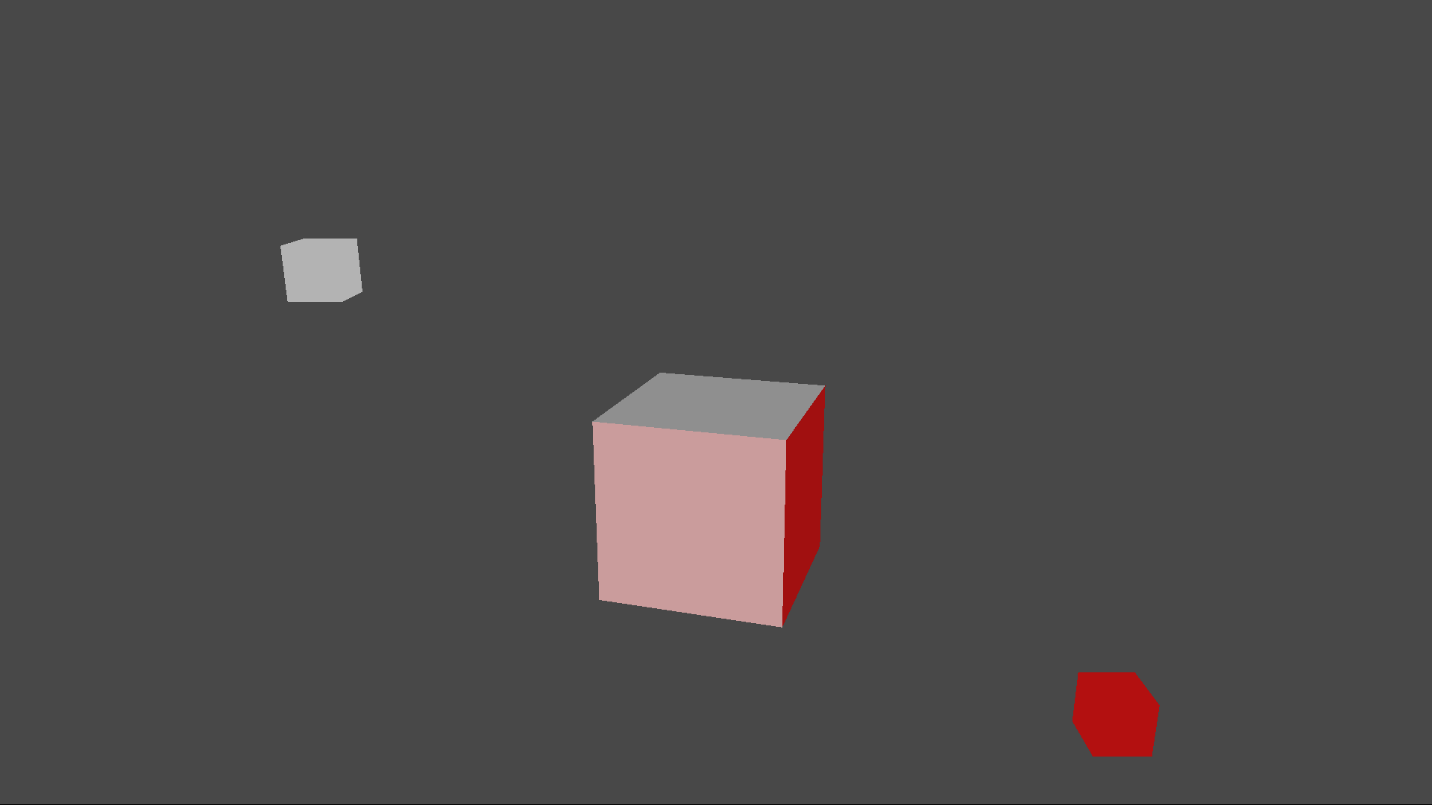
간단한 조명 샘플

*이 샘플은 Microsoft 게임 개발 키트 미리 보기와 호환됩니다(2019년 11월).*

# 설명

이 샘플에서는 정적 및 동적 Lambertian 조명을 사용하여 비추는 인덱싱된 기하 도형을 그리는 정적 Direct3D 12 꼭짓점, 인덱스 및 상수 버퍼를 만드는 방법을 보여 줍니다.

이 샘플에서는 두 개의 조명(흰색 1개, 빨간색 1개)으로 비추고 큐브로 표현되는 대형 큐브를 렌더링합니다. 흰색 조명은 고정된 상태로 유지되고, 빨간색 조명은 중앙 큐브 주위를 선회합니다. 중앙 큐브도 회전합니다. 모션을 활용하여 다양한 각도에서 색이 지정된 조명 효과를 관찰할 수 있습니다.



# 샘플 빌드

Xbox One 개발 키트를 사용하는 경우 활성 솔루션 플랫폼을 Gaming.Xbox.XboxOne.x64로 설정하세요.

Project Scarlett을 사용하는 경우 활성 솔루션 플랫폼을 Gaming.Xbox.Scarlett.x64로 설정하세요.

*자세한 내용은 GDK 문서에서* 샘플 실행하기*를 참조하세요.*

# 샘플 사용

|  |  |
| --- | --- |
| 작업 | 게임패드 |
| 종료 | 보기 단추 |

# 구현 참고 사항

## 셰이더

## 샘플에서는 세 개의 셰이더(꼭짓점 셰이더 1개("TriangleVS"), 픽셀 셰이더 2개("LambertPS", "SolidColorPS"))를 사용하여 장면을 렌더링합니다. 컴파일된 셰이더 BLOB은 CreateDeviceDependentResources에 로드된 다음 각 셰이더 조합에 대한 파이프라인 상태 개체를 만들 때 참조됩니다. 동일한 HLSL에 정의되는 모든 셰이더에는 "SimpleLighting.hlsli" 파일이 포함되며 세 스텁 셰이더에 이 파일이 포함되어 있습니다. 세 셰이더 BLOB을 만들기 위해 진입점마다 다른 스텁 셰이더를 컴파일합니다.

## 파이프라인 상태 개체(PSO)

간단한 조명 샘플에는 두 개의 고유한 셰이더 조합이 있습니다. 첫 번째는 TriangleVS와 LambertPS가 결합된 조합이고, 두 번째는 TriangleVS와 SolidColorPS가 결합된 조합입니다. DirectX 12에서 모든 고유한 셰이더 조합을 위한 파이프라인 상태 개체(PSO)를 만들어야 합니다. 이름에서 알 수 있듯이 PSO는 특정 셰이더 집합을 사용하여 가능한 일부 그리기 호출에 필요한 모든 파이프라인 상태를 캡슐화합니다. PSO는 루트 서명, 다양한 파이프라인 단계를 위한 셰이더, 상태 설정(예: 래스터라이저 상태, 깊이 스텐실 상태, 혼합 상태 등)을 결합합니다. 자세한 내용은 MSDN 관련 문서를 참조하세요.

## 루트 서명

루트 서명은 그래픽 파이프라인에 바인딩되는 리소스의 유형과 리소스를 배치하는 방법을 정의합니다. 루트 서명은 API 함수 서명과 유사하며, 매개 변수 유형, 매개 변수 순서 및 레이아웃을 설명하지만, 실제 매개 변수 인스턴스를 정의하지 않습니다. 루트 매개 변수는 루트 서명의 요소에 해당하는 실제 데이터 인스턴스입니다. 샘플의 꼭짓점 셰이더에는 셰이더 상수를 위한 단일 구조만 필요하므로 루트 서명이 매우 단순합니다. 루트 서명에는 ConstantBufferView 형식의 단일 루트 매개 변수가 포함되어 있습니다.

## 기하 도형

## 장면의 기하 도형은 큐브의 6개 쿼드를 나타내는 24개 꼭짓점에 대한 데이터를 포함하는 인덱스 배열과 정적 꼭짓점으로 구성됩니다. 두 개의 배열이 Sample::CreateDeviceDependentResources 내에 선언됩니다. ID3D12Device::CreateCommittedResource에서 두 배열을 직접 사용하여 버퍼에 대한 ID3D12Resources를 만듭니다. 간단히 하기 위해 샘플에서는 D3D12\_HEAP\_TYPE\_UPLOAD를 사용합니다. 그러면 각 리소스를 만드는 단계와 데이터로 리소스를 초기화하는 단계를 한 번에 수행할 수 있습니다. 하지만 \_UPLOAD 힙은 기하 도형 데이터에 최적화된 위치가 아닙니다. 더 효과적인 구현에서는 기하 도형 데이터에 D3D12\_HEAP\_TYPE\_DEFAULT를 사용합니다. \_DEFAULT 힙을 초기화하려면 \_UPLOAD 힙을 사용해야 합니다. 결국 두 개의 힙을 사용하게 되어 구현이 복잡해집니다.

기하 도형의 버퍼가 생성되면 샘플에서는 꼭짓점을 위한 D3D12\_VERTEX\_BUFFER\_VIEW와 인덱스를 위한 D3D12\_INDEX\_BUFFER\_VIEW를 만들 수 있습니다. 이러한 보기는 ID3D12GraphicsCommandList::IASetVertextBuffer 및 ID3D12GraphicsCommandList::IASetVertextBuffer를 호출하여 입력 어셈블러를 설정할 때 Sample::Render에서 사용됩니다.

## 셰이더 상수 관리

매우 간단한 이 장면에서 모든 셰이더 상수는 다음을 포함하는 단일 상수 버퍼로 함께 결합됩니다.

* 세계, 보기 및 투영 행렬
* 조명 방향 및 색
* 단색

더 복잡한 장면에서는 일반적으로 상수가 업데이트되는 빈도에 따라 상수를 여러 버퍼로 분할합니다.

대형 큐브와 빨간색 조명이 애니메이션되므로 그리기 호출 간에 셰이더 상수를 프레임당 여러 번 업데이트해야 합니다. 루트 서명에는 각 그리기 호출에서 사용하기 위해 셰이더 상수의 복사본을 참조해야 하는 ConstantBufferView 형식의 단일 루트 매개 변수가 포함되어 있습니다. CPU와 GPU가 병렬로 작동하므로 GPU에서 상수 버퍼 사용을 마칠 때까지 CPU에서 상수 버퍼를 업데이트하려고 시도해서는 안 됩니다. 상수 버퍼가 하나뿐인 경우 GPU에서 그리그를 마칠 때까지 CPU를 차단해야 합니다. 여러 그리기 호출에 대해 상수를 업데이트해야 하므로 이는 비현실적입니다. 따라서 샘플에서는 GPU에서 그리기를 수행하는 동안 CPU에서 CPU에 계속해서 상수를 전송할 수 있도록 상수 버퍼를 여러 개 사용합니다.

이 샘플은 간단하고 프레임당 그리기 호출 수가 고정되어 있으므로 컴파일 시간에 알 수 있습니다. 이 샘플에서는 그리기 호출당 하나의 버퍼를 만들고 스왑 체인의 백 버퍼 수를 곱합니다. 이 숫자를 사용하면 CPU에서 쓸 수 있는 빈 버퍼가 항상 존재합니다. 모든 상수 버퍼는 Sample::CreateDeviceDependentResources에서 생성된 단일 연속 업로드 버퍼에 저장됩니다. 업로드 버퍼를 즉시 매핑하여 CPU 주소 공간과 GPU 주소 공간 모두에 대한 기본 메모리 주소를 얻습니다.

Sample::Render 메서드에서 상수는 상수 업로드 힙의 CPU 기본 주소에서 인덱싱된 위치에 기록됩니다. ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView를 호출하여 파이프라인에 버퍼를 바인딩하기 위해 동일한 인덱스를 GPU 주소와 결합합니다. 인덱스에서는 백 버퍼 주기와 그리기 횟수를 고려해야 합니다. 자세한 내용은 Sample::Render 구현을 참조하세요.

## CPU/GPU 동기화

GPU에서 처리할 수 있는 것보다 더 빠른 속도로 CPU에서 GPU에 명령 목록을 발행할 수 있을 경우 GPU에서 확인할 때까지 CPU가 대기해야 합니다. 이 샘플에서는 라운드 로빈 방식으로 상수 버퍼 메모리를 사용합니다. 즉, 정해진 기간이 경과한 후 버퍼 슬롯을 다시 사용합니다. 일반적으로 공유 리소스를 다시 사용하기 전에 몇 가지 동기화 전략을 사용하여 리소스가 사용 중이 아닌지 확인해야 합니다. 샘플에서는 ID3D12Fence를 사용하여 CPU와 GPU를 동기화합니다. CPU는 명령을 명령 큐에 삽입하여 프레임 인덱스 값으로 울타리 개체에 “신호”를 보냅니다. 이제 신호 명령이 CPU에서 처리되는 즉시 제공된 프레임 인덱스 값이 CPU에 표시됩니다. 그러면 CPU에서 현재 프레임 인덱스를 GPU에서 신호로 보낸 마지막 프레임 인덱스와 비교하여 GPU와 CPU의 간격을 확인할 수 있습니다. GPU 프레임 수와 CPU 프레임 수의 차이가 백 버퍼 수를 초과할 경우 CPU가 대기해야 합니다.

# 개인정보처리방침

샘플을 컴파일하고 실행할 때 샘플의 사용을 추적하는 데 도움이 되도록 샘플 실행 파일의 파일 이름이 Microsoft에 전송됩니다. 이 데이터 수집을 옵트아웃하려면 Main.cpp에서 "샘플 사용 원격 분석"이라고 레이블이 지정된 코드 블록을 제거할 수 있습니다.

Microsoft의 일반 개인정보취급방침에 대한 자세한 내용은 [Microsoft 개인정보처리방침](https://privacy.microsoft.com/en-us/privacystatement/)을 참조하세요.