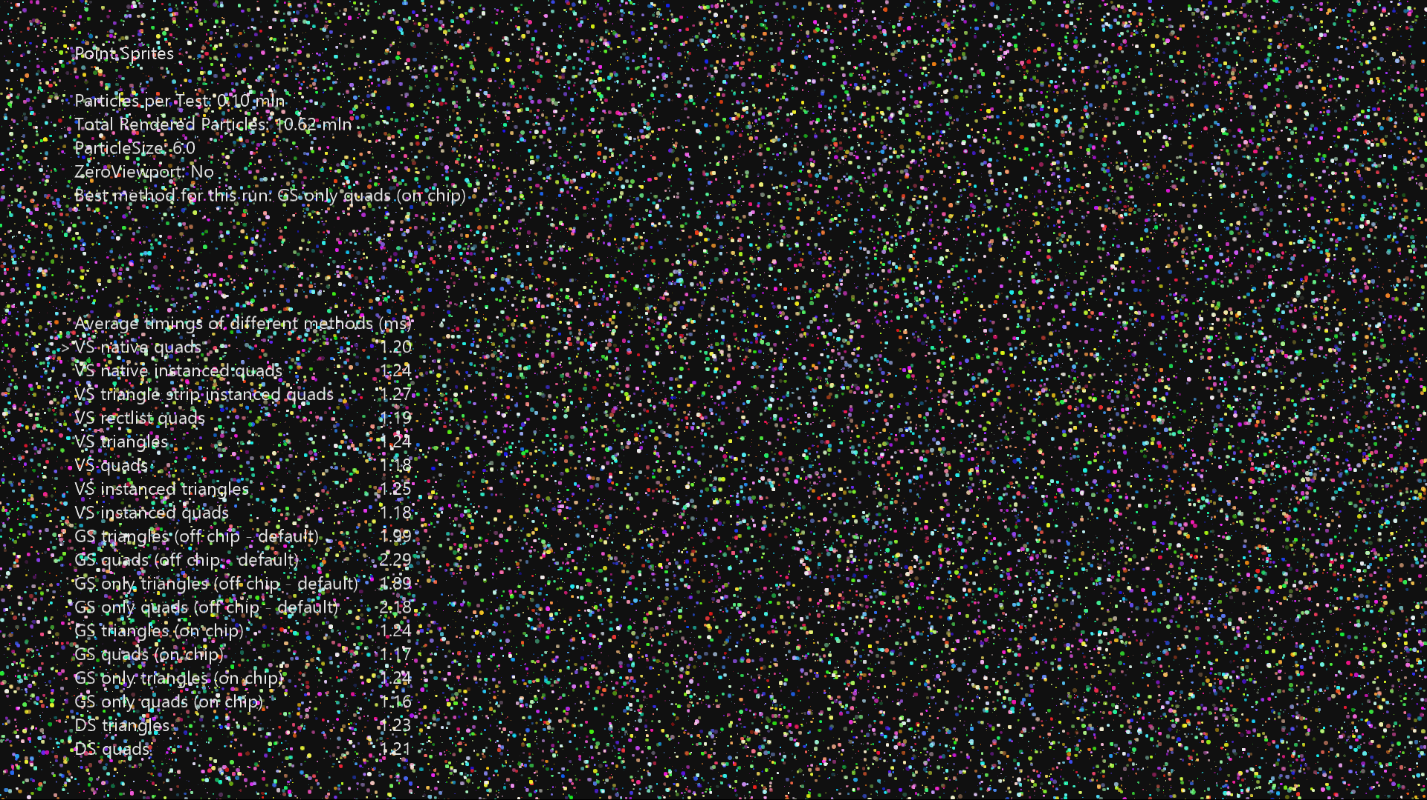


점 스프라이트

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

# 설명

이 샘플은 DirectX 12에서 점 스프라이트를 렌더링하는 10가지 방법을 보여 줍니다.



# 샘플 빌드

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

*자세한 내용은 GDK 문서에서* 샘플 실행하기*를 참조하세요.*

# 샘플 사용하기

이 샘플은 다음 컨트롤을 사용합니다.

|  |  |
| --- | --- |
| 작업 | 게임패드 |
| 샘플을 종료합니다. | 보기 단추 |
| 스프라이트 크기 늘리기 | A 단추 |
| 스프라이트 크기 줄이기 | B 단추 |
| Null 뷰포트 토글 | Y 단추 |
| 선택한 테스트 변경 | D-패드 위/아래로 |
| 강조 표시된 테스트 토글 | X 단추 |

# 구현 참고 사항

그래픽 파이프라인이 진화하면서, 점 스프라이트에 대한 첫 번째 클래스 지원은 형상 셰이더와 같이 완전히 유연하고 사용자 지정 가능한 스테이지를 사용할 수 있게 제거되었습니다. 사실 GPU의 모든 기하 도형 관련 스테이지인 정점, 기하 도형 및 공간 분할 셰이더에서 점 스프라이트를 구현할 수 있습니다. 사례별로 측정한 성능에 따라 적절한 방법을 선택하는 것이 이상적이지만, 점 스프라이트를 확장하기 위해 기하 도형 셰이더를 사용하는 것이 단순한 점 스프라이트를 렌더링하는 가장 효율적인 방법은 아닌 것으로 나타납니다.

## 삼각형 및 4분면

렌더링 파이프라인에서 병목 현상이 발생하는 위치에 따라, 삼각형이나 4분면은 다른 성능을 나타낼 수 있습니다. 채우기 속도와 관련된 상황에서 4분면은 투명한 픽셀에서 픽셀 셰이더를 실행할 필요가 없으므로 삼각형보다 성능이 뛰어납니다. 렌더링이 기하 도형 스테이지로 제한되면 4분면 대신 삼각형을 출력하는 것이 더 적합합니다. 여러 복잡한 렌더링 상황에서는 삼각형이 4분면보다 더 빠를 수 있습니다.

이 샘플에서는 삼각형과 4분면을 대조하여 다른 크기의 점 스프라이트에서의 차이를 보여 주는 각 방법을 설명합니다.

## 방법 1 - 정점 + 기하 도형 셰이더

정점 셰이더가 정점을 정상적으로 읽도록 파이프라인을 초기화한 후, 기하 도형 셰이더가 입력 정점별로 4분면 또는 삼각형을 출력합니다. 이러한 테스트에서는 온칩 및 오프칩 기하 도형 셰이더 할당도 비교합니다. 컴파일러는 기본적으로 정점 및 기하 도형 스테이지 사이에 스테이지 간 데이터를 저장하는 데 사용되는 오프칩 메모리를 사용합니다.

장점

* 잘 알려져 있고 간단합니다.

단점

* 대부분의 설정 작업이 필요합니다.
* 항상 가장 빠른 것은 아닙니다.

## 방법 2 – 기하 도형 셰이더만

이 방법은 빈 정점 셰이더를 사용합니다. 정점 데이터를 로드하기 위해 기하 도형 셰이더는 SV\_PrimitiveID 인덱스(점 목록의 경우 기하 도형 셰이더의 SV\_PrimitiveID는 정점 셰이더의 SV\_VertexID와 같음)를 사용하여 정점 버퍼의 원시 바이트 보기에서 버퍼 로드를 수행합니다. 정점이 로드된 후에는 방법 1과 같이 점 스프라이트 확장이 수행됩니다. 여기서는 온칩 및 오프칩 GS 성능이 유사하게 테스트됩니다.

장점

* VS는 no-op이므로 입력 레이아웃이 필요하지 않습니다.
* VS와 GS 사이에 내부 트래픽이 발생하지 않으므로 메서드 1보다 속도가 빠릅니다.

단점

* 수동 정점 로드를 위해 정점 레이아웃마다 다른 셰이더 변형을 사용해야 합니다. 최적화된 코드에서는 문제가 되지 않습니다.
* 이 방법은 이해하기가 좀 더 어렵습니다.

## 방법 3 – 정점 셰이더만

DX11에서는 단지 4분면 또는 삼각형으로 점을 확장하기 위해 반드시 기하 도형 셰이더를 사용할 필요는 없으며, 정점 셰이더를 대신 사용할 수 있습니다. DX11에서 정점 셰이더 스테이지는 원시 바이트 UAV를 읽을 수 있으므로 SV\_VertexID와 정점 버퍼의 원시 바이트 보기가 있는 경우 정점을 수동으로 읽을 수 있습니다.

따라서 정점을 삼각형 또는 4분면으로 확장하려면 그리기 호출에서 3배 또는 6배 더 많은 정점을 렌더링하고, 셰이더에서 3 또는 6으로 나누어 정점 인덱스를 가져오고, 이 인덱스를 사용하여 정점을 로드하고, 나누기의 나머지 결과에 따라 확장하여 스프라이트의 모서리를 구해야 합니다.

장점

* 지금까지 방법 중에서 점 스프라이트를 렌더링하는 가장 빠른 방법입니다. 좀 더 작은 스프라이트의 경우는 기하 도형 셰이더 방법보다 3배 이상 더 빠르며(기 하 도형으로 제한된 경우), 좀 더 큰 스프라이트의 경우는 약 50% 더 빠릅니다.
* 시스템 생성 값 SV\_VertexID를 사용하는 경우에만 VS에 대해 입력 레이아웃이 필요하지 않습니다.

단점

* 정점 로드는 수동으로 진행되므로, 다른 정점 레이아웃을 로드하는 데 다른 셰이더가 필요할 수 있습니다. 가장 빠른 방법을 원할 경우에는 이것이 문제가 되지는 않을 것입니다.

## 방법 4 - 정점 셰이더 인스턴스화

인스턴스화를 사용하여 GPU에서 점 스프라이트 정점을 로드하도록 하고 SV\_VertexID를 사용하여 확장을 위한 스프라이트 모서리를 결정할 수 있습니다. 이 방법은 이전 방법보다 약간 느리지만 다른 방법보다는 더 빠릅니다.

장점

* 두 번째로 빠른 방법이며, 방법 3과 성능은 매우 유사합니다.
* 나누기가 필요하지 않으므로 셰이더의 ALU가 더 작습니다.
* 정점 로드는 입력 레이아웃을 사용하여 수행되므로 셰이더를 수정하지 않고 다른 정점 레이아웃을 로드할 수 있습니다.
* 매우 큰 점 스프라이트의 경우에는 적절한 방법으로 보입니다.

단점

* 없음

## 방법 5 – 공간 분할 스테이지

공간 분할 스테이지를 사용하여 단일 입력 정점에서 삼각형 및 4분면을 생성할 수 있습니다. 이 방법의 성능은 기하 도형 셰이더 기반 방법과 메서드와 유사하지만 더 많은 스프라이트 셰이프를 사용할 수 있으므로 좀 더 유연합니다. 예를 들어, 4분면 영역에서 공간 분할을 수행하여 4분면 대신 원을 출력할 수 있습니다. 픽셀 셰이더가 더 느리고 스프라이트가 원이므로 이 방법이 더 효율적일 수 있습니다.

장점

* 거의 임의의 기하 도형 셰이프를 출력할 수 있습니다.

단점

* 기하 도형 셰이더 접근법과 동일합니다.

## 결론:

위의 모든 방법은 다른 GPU 로드를 수행하게 되므로 본인의 사례에 가장 적합한 방법을 선택합니다.

# 알려진 문제

이 샘플은 현재 Gaming.Xbox.Scarlett.x64 플랫폼에서는 빌드되지 않습니다.

# 업데이트 기록

2019년 4월 12일 - Xbox 샘플 프레임워크에서 DX12로 이식

# 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.