NVPerfHUD 3
요약 설명서

NVPerfHUD 를 구성하면 그래픽 파이프라인 실험, 성능 수치 그래픽 표시를 수행할 수 있으며, 여러 가지 성능 시각화 모드를 통해 잠재적인 문제를 발견할 수 있습니다. 그 뿐 아니라, 고급 디버깅을 위한 NVPerfHUD 디버그 콘솔 또는 프레임 분석 모드를 사용할 수도 있습니다. 모드를 전환하려면 단축키를 사용하십시오.

F5 **성능 분석 모드 –** 타이밍 그래프와 직접 실험으로 병목을 확인합니다.

F6 **디버그 콘솔 모드 –** DirectX Debug 런타임의 메시지, NVPerfHUD 경고, 애플리케이션의 커스텀 메시지를 검토합니다.

F7 **프레임 분석 모드 –** 고급 상태 검사기로 그래픽 파이프라인의 각 단계를 조사하여, 현재 프레임을 중단하고 한 번에 장면 하나에 대한 그리기 요청을 단계별로 처리합니다.

성능 분석 모드

화면에 표시되는 정보를 구성하고 여러 가지 그래픽 파이프라인 실험을 수행합니다.

F1 주기적인 유용한 정보 표시

B 배치 크기 히스토그램 표시

F 흐릿한 배경

H 그래프 숨김

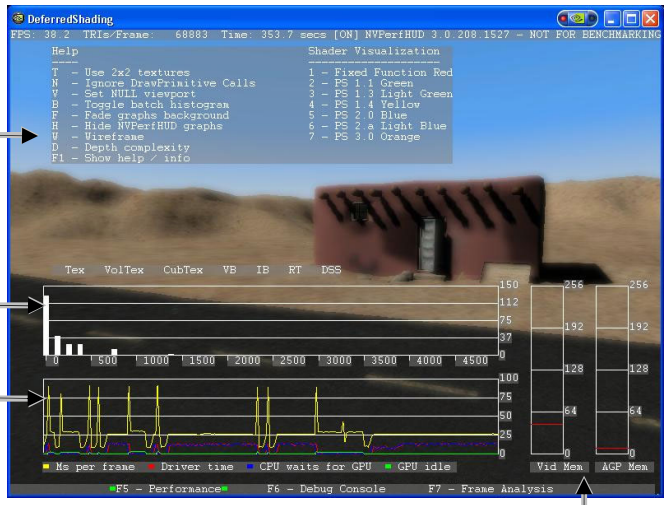
W 곡면 표시

D 깊이 복잡성 표시

T 2x2 텍스처를 사용하도록 강제하여 텍스처 유닛 격리

V 버텍스 쉐이더가 1x1 scissor rectangle 을 사용하여 모든 래스터와 및 쉐이딩 작업을 클립 처리한 후 유닛의 작업부하를 0 으로 만듦

N 모든 그리기 요청을 무시하여 GPU(및 상태 변경 오버헤드)를 제거



도움 정보 (**F1** 토글)

배치 그래프(**B**)

타이밍 그래프(**H**)

메모리 그래프

프레임 분석 모드

슬라이더를 사용하거나 왼쪽/오른쪽 화살표 키를 사용한 후 아래 옵션으로 장면을 수정합니다.

A 고급 / 단순 디스플레이 전환

S 경고 표시

W 곡면 표시

D 깊이 복잡성 표시

상태 검사기

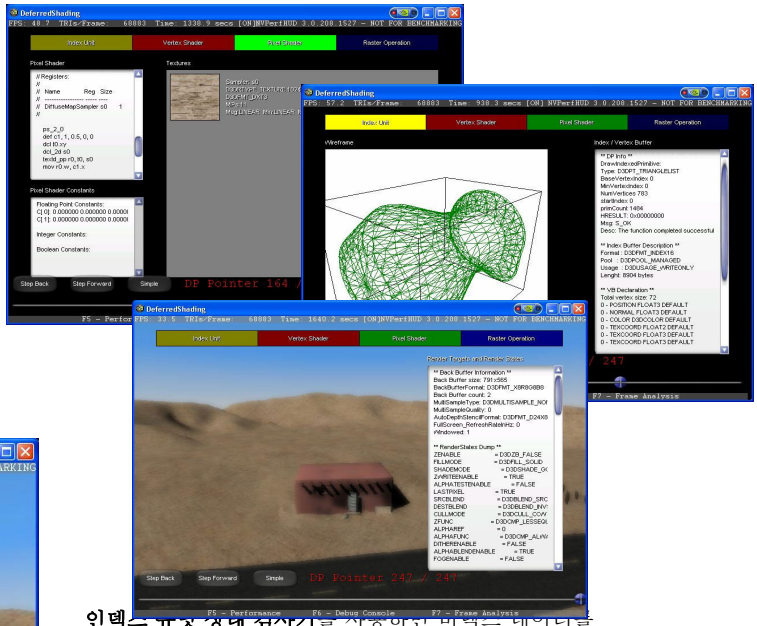
고급... 버튼을 클릭하면 고급 상태 검사기가 활성화됩니다. 컬러 막대를 클릭하거나 아래 단축키를 사용하여, 각 상태 검사기를 전환할 수 있습니다.

1 **인덱스 장치** – 버텍스 데이터 페치

2 **버텍스 쉐이더** – 버텍스 쉐이더 실행

3 **픽셀 쉐이더** – 픽셀 쉐이더 실행

4 **래스터 연산 –** 포스트 쉐이딩 연산



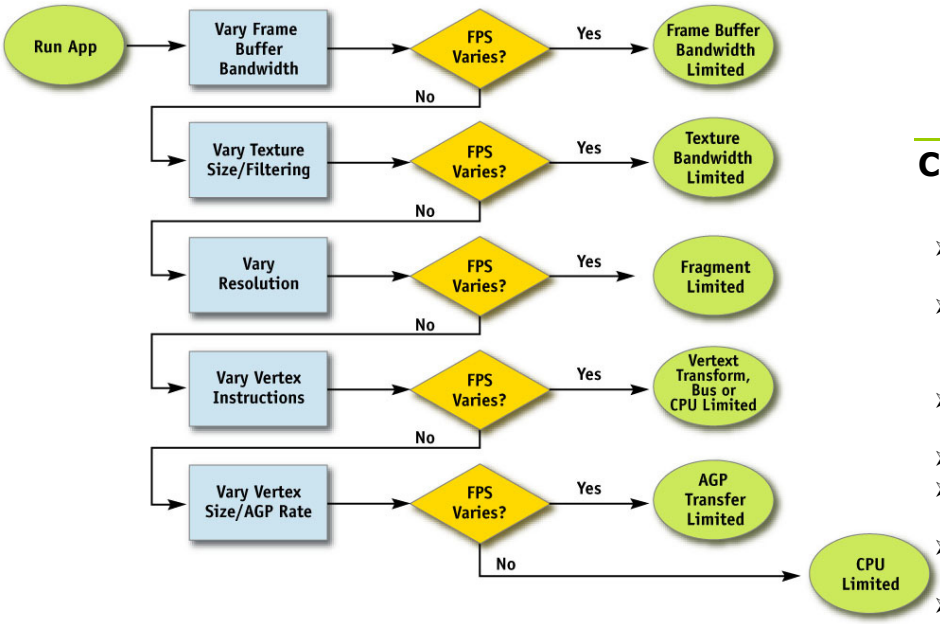
인덱스 유닛 상태 검사기를 사용하면 버텍스 데이터를 페치하는 데 사용한 모든 정보를 검사하고, 현재 그리기 요청에 관한 기하가 정확한지 확인할 수 있습니다.

버텍스 및 **픽셀 쉐이더 상태 검사기**를 사용하면, 현재 그리기 요청에 대한 쉐이더 프로그램 상수 및 텍스처를 검사할 수 있습니다. 상수가 **#NAN** 또는 **#INF** 가 아닌지 확인하십시오.

혼합이 제대로 작동하지 않을 때 **래스터 연산 상태 검사기**를 사용하여, 백 버퍼 형식에 알파 구성요소가 있는지 확인하고 **blendEnable** 로 불투명 객체를 그리지 않았는지 확인하십시오.

NVPerfHUD 3 요약 설명서

병목점 찾기



참고: CPU 제한이 의심되는 경우에는 언제든지 N 을 누르십시오. 애플리케이션의 프레임 속도가 변하지 않으면 CPU 에 제한이 있는 것입니다.

GPU 최적화

❑ 픽셀 쉐이딩 속도 향상

- 먼저 깊이를 렌더링합니다.
- 이른 z 값 최적화로 픽셀 처리를 줄입니다.
- 복잡한 함수를 텍스처에 저장합니다.
- 픽셀 단위 작업을 버텍스 쉐이더로 옮깁니다.
- 가능한 한 낮은 정밀도를 사용합니다.
- 불필요한 정규화를 피합니다.
- 가능하면 50% 정밀도의 정규화를 사용합니다.
- 픽셀 셰이더에 LOD 를 사용하는 것을 고려합니다.
- 꼭 필요한 것이 아니라면 삼중 선형 필터링은 해제합니다.

❑ 텍스처 대역폭 줄이기

- 텍스처의 크기를 줄입니다.
- 최소로 할 수 있는 모든 표면에 항상 밉매핑을 사용합니다.
- 모든 컬러 텍스처를 압축합니다.
- 꼭 필요한 것이 아니라면 비용이 많이 드는 텍스처 포맷은 피합니다.

❑ 프레임버퍼 대역폭 최적화

- 먼저 깊이를 렌더링합니다.
- 알파 블렌딩을 줄입니다.
- 가능하면 깊이 쓰기를 사용하지 않습니다.
- 불필요한 컬러 버퍼 지우기를 피합니다.
- 앞에서 뒤로 렌더링합니다.
- 스카이박스 렌더링을 최적화합니다.
- 필요할 때에만 부동소수점 프레임 버퍼를 사용합니다.
- 가능하면 16 비트 깊이 버퍼를 사용합니다.
- 가능하면 16 비트 컬러를 사용합니다.

최적화에 대한 자세한 내용은 *NVPerfHUD 사용 설명서*를 참조하십시오.

방법론

1. **병목점을 찾습니다**
2. **병목 스테이지를 최적화합니다**
3. **원하는 수준에 도달할 때까지 1 단계와 2 단계를 반복합니다.**

CPU 최적화

❑ 리소스 잠금 회수 줄이기

- 이전에 렌더링하고 있었던 표면을 잠그거나 그 표면에서 데이터를 읽지 않습니다
- 텍스처 또는 버텍스 버퍼와 같이, GPU 가 데이터를 읽어들이는 표면에 쓰기를 하지 않습니다

❑ 그리기 호출 최소화

- ‘triangle strip’을 사용하는 경우에는 분해된 스트립(strip)을 붙이기 위해 ‘퇴화된’ 삼각형을 사용합니다.
- 텍스처 페이지를 사용합니다.
- 버텍스 쉐이더의 상수 메모리를 매트릭스에 대한 룩업 테이블로 사용합니다.
- 화면 내에 동일한 메시가 여러 개 존재한다면 geometry instancing 을 사용합니다.
- 배치 크기를 늘리기 위해 GPU 쉐이더 분기를 사용합니다.
- 결정은 파이프라인에서 되도록 나중으로 미룹니다.

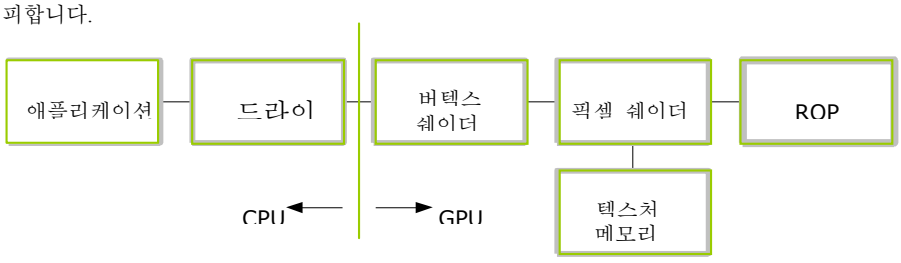
❑ 버텍스 전송 비용 줄이기

- 버텍스 포맷에서 가능한 한 작은 바이트를 사용합니다.
- 버텍스 프로그램 내에서 유도가 가능한 버텍스 속성을 입력 버텍스 포맷 속에 저장하지 않고 생성합니다.
- 32 비트 인덱스 대신 16 비트 인덱스를 사용합니다.
- 비교적 순차적으로 버텍스 데이터에 접근합니다.

❑ 버텍스 처리 최적화

- 객체별 연산을 CPU 쪽으로 뺍니다.
- post-TnL 버텍스 캐시를 위해 최적화합니다.
- 처리할 버텍스 수를 줄입니다.
- 버텍스를 처리할 때 LOD 를 사용합니다.
- 정확한 좌표 공간을 사용합니다.
- 버텍스 분기를 사용하여 연산을 빨리 끝냅니다.

그래픽 파이프라인



developer.nvidia.com
GPU 프로그래밍을 위한 소스