차세대 쉐이딩과 렌더링

Bryan Dudash
NVIDIA

개요

- 3.0 쉐이더 모델 개요
  - ps.3.0 대 ps.2.0
  - vs.3.0 대 vs.2.0
- 차세대 렌더링 예제
  - 유동적 물의 움직임
    - 버텍스 텍스쳐 페치 (버텍스 텍스쳐 Fetch)
    - 부동점 필터링/블렌딩
    - GPU 기반의 물리 시뮬레이션
  - 입체적 안개 (Volumetric Fog)
    - 가속을 위한 MRT와 브랜칭
  - 자연 렌더링 (Deferred 렌더링)
    - 가속을 위한 MRT와 브랜칭
- 지오메트리 인스턴스화 (Geometry Instancing)
  - 시각적 복잡도 추가
  - 성능 최적화

픽셀 쉐이더 3.0 특징

픽셀 쉐이더 기능	쉐이더 2.0	쉐이더 3.0	설명
쉐이더 길이	96	65535+	보다 복잡한 쉐이더, 빛, 질감의 표현을 가능하게 한다.
Dynamic branching	아니오	예	관계없는 픽셀에서는 복잡한 쉐이딩을 거치지 않으므로 성능면에서 절약하게 됨
쉐이더 앤티앨리어싱	지원되지 않음	내장된 파생 명령어 (Built-in derivative 명령어)	개발자들은 어떤 함수에서 파생된 화면 공간을 계산할 수 있어서 artifact을 제거하기 위한 쉐이딩 프리퀀시 혹은 오버샘플링을 조절할 수 있도록 함
최소 정밀도	fp24	fp32	더 적은 artifacts과 더 많은 dynamic range
Back-face 레지스터	아니오	예	한 번의 pass로 2면의 라이팅을 허용함

픽셀 쉐이더 3.0 기능비교

픽셀 쉐이더 기능	쉐이더 2.0	쉐이더 3.0	설명
Back-face 레지스터	아니오	예	한 번의 pass로 양면의 라이팅(lighting)이 가능함
Interpolated color format	8-비트 정수 최소	32-비트 부동소수점 최소	더 넓은 범위와 정밀 색상은 버텍스 단계에서 high-dynamic range lighting을 가능하게 함
다중 렌더 타겟 (Multiple render targets)	선택사항	4 필수	필터링(filtering) 과 버텍스 작업을 줄이기 위한 향상된 라이팅 알고리즘이 가능함– 최소 단가로 더 많은 광원이 가능
Fog 와 specular	8-비트 고정 함수 최소	커스텀 fp16-fp32 쉐이더 프로그램	쉐이더 모델 3.0 개발자들에게 기존의 고정 함수였던 스펙큘러(specular) 및 포그(fog) 연산에 완전하고 정확한 제어를 부여한다
텍스쳐 coordinate count	8	10	다 많은 픽셀 당 입력은 좀 더 현실적인 렌더링 (특히 skin에서)을 가능하게 한다

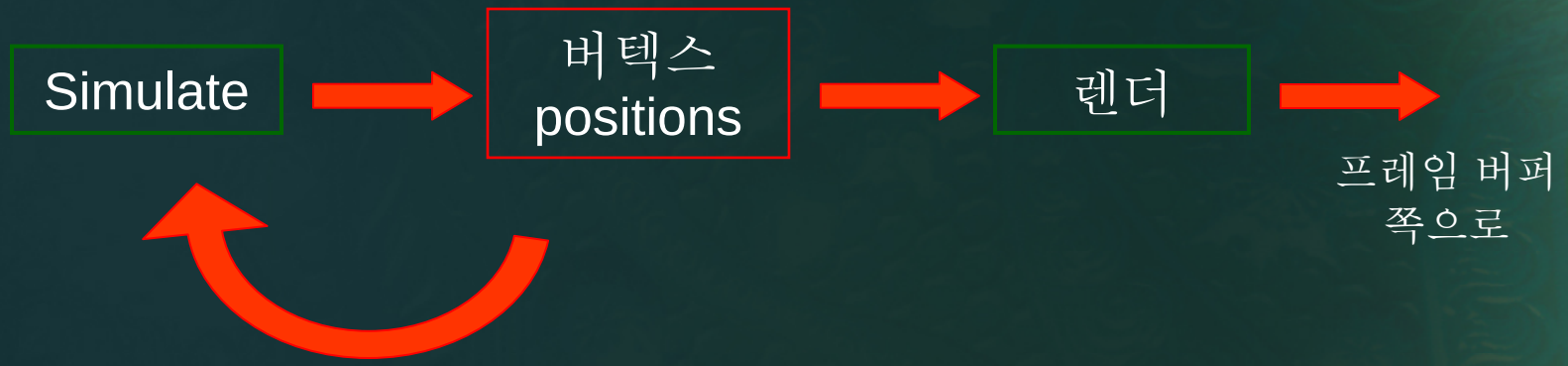
버텍스 쉐이더 3.0 기능비교

버텍스 쉐이더 기능	쉐이더 2.0	쉐이더 3.0	설명
쉐이더 길이	256 명령어	65535 명령어	더 많은 명령어로 더 상세한 캐릭터 라이팅 및 애니메이션이 가능
Dynamic branching	아니오	예	관계없는 버텍스에서는 애니메이션과 연산을 거치지 않으므로 성능면에서 절약하게 됨
버텍스 텍스쳐	아니오	4개의 텍스쳐까지 어떠한 숫자의 룩업(lookup) 가능	디스플레이스먼트 매핑(displacement mapping), 파티클 효과 가능
Instancing 지원	아니오	필수	단 하나의 명령어만으로 수 많은 다양한 개체들이 그려지는 것이 가능함

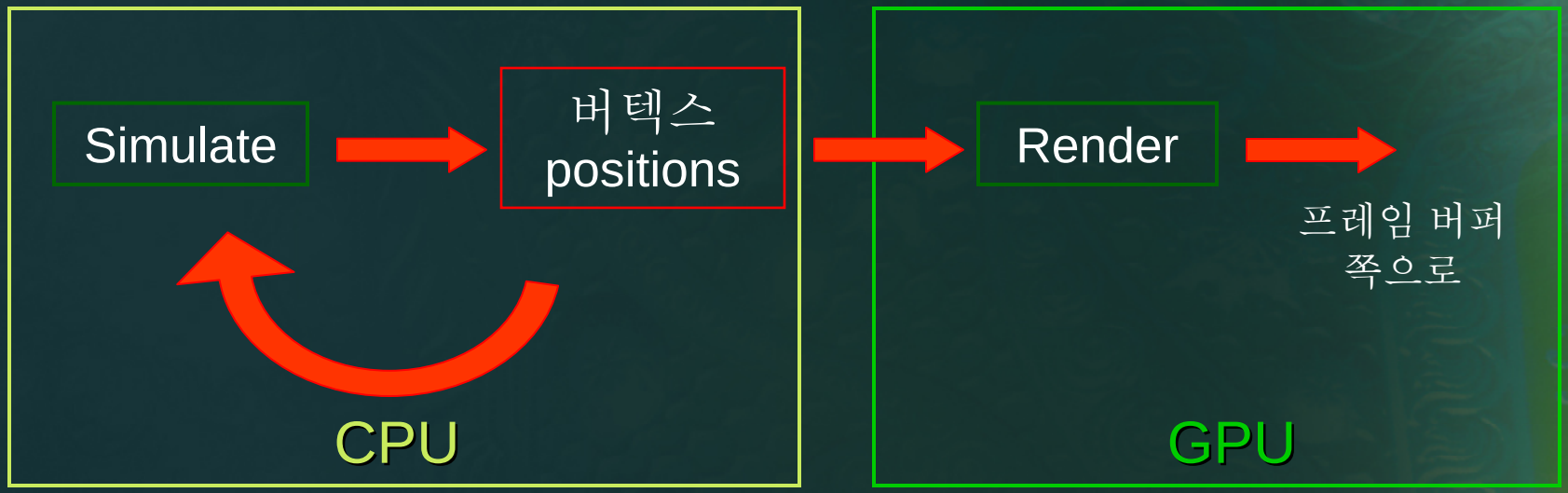
이러한 기능으로 가능한 작업

- 물의 움직임 렌더링
  - VS3.0 버텍스 텍스쳐 페치
  - 부동점 필터링/블렌딩
  - GPU 기반의 물리 시뮬레이션
- 입체적 안개 애니메이션
  - 다각형 기본을 사용한 안개 바운딩
  - 가속을 위한 MRT와 브랜칭
- 지오메트리의 인스턴스화
  - Draw call 1회로 한 메쉬의 여러 “인스턴스”를 한번에 드로잉

일반적인 워크플로우

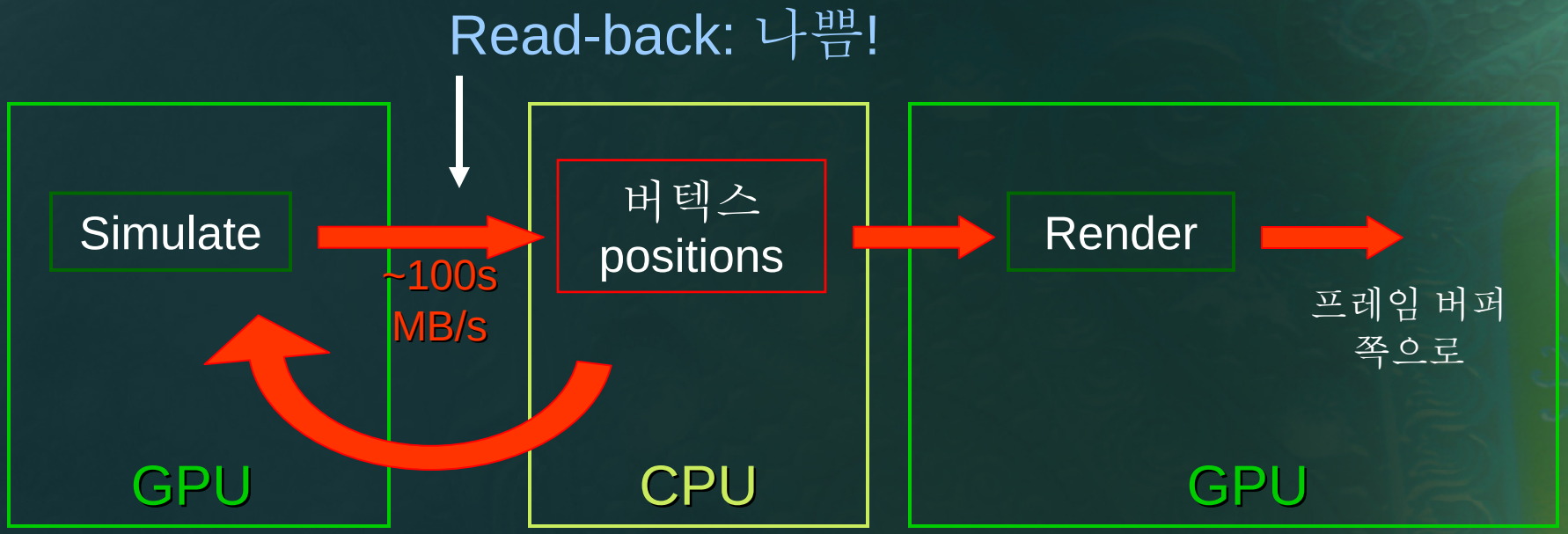


일반적인 프로세스 할당



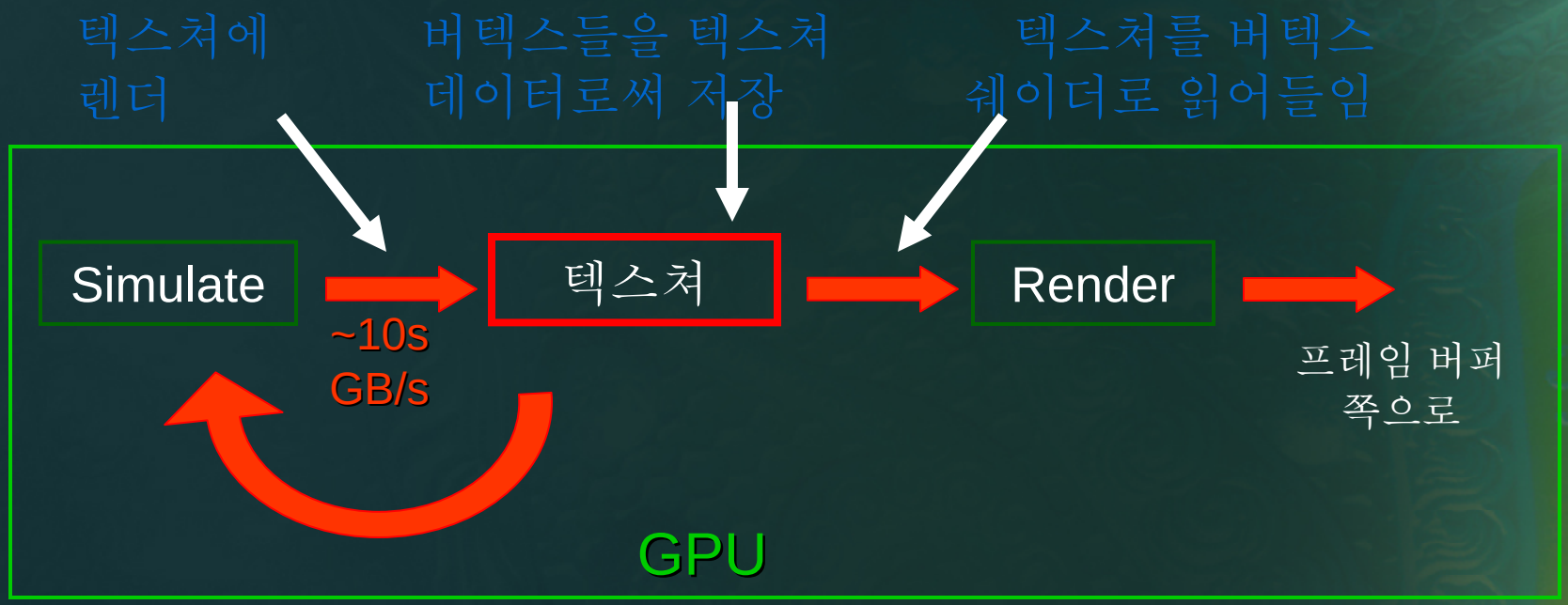
GPU에서 시뮬레이션

- 프로그램이 가능한 쉐이더를 사용하십시오
  - 리드백(read-back)이 어려울 수 있음
  - 이 제품은 PCI용이며, PCI Express가 더 적합



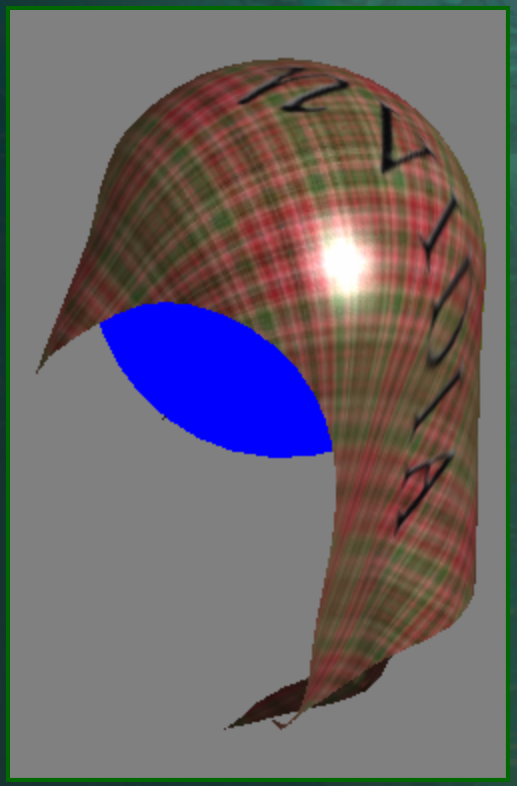
“버텍스 버퍼로 렌더링”

- GPU의 리드백(read-back)을 CPU로 이동



예제

- 직물 (cloth)
  - 직물 충돌의 장면
  - 직물 물리 실행:
    젖은 직물의 탄력성
- 디스플레이스먼트 매핑
  - 버텍스 변위

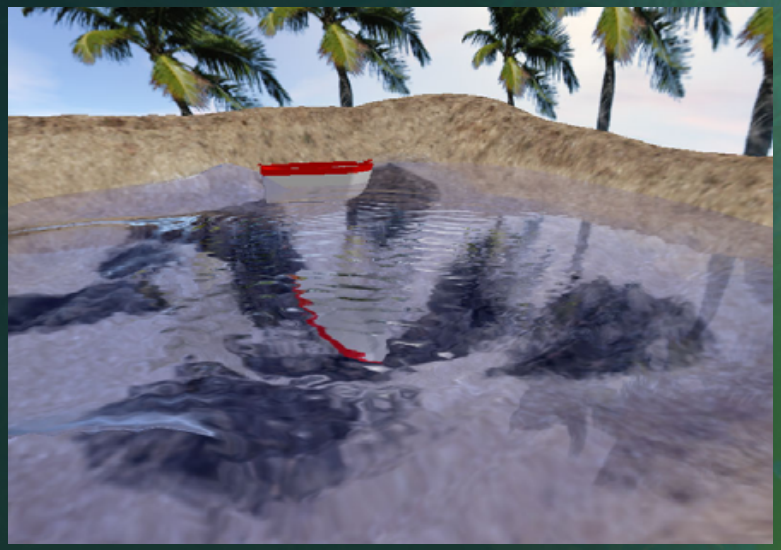


추가 예제

- 눈/모래의 누적
  - 마찰/활주 시뮬레이션
- 바람 (시뮬레이션)에 휘어지는 풀
- 입자 시스템
- 물결/흔적

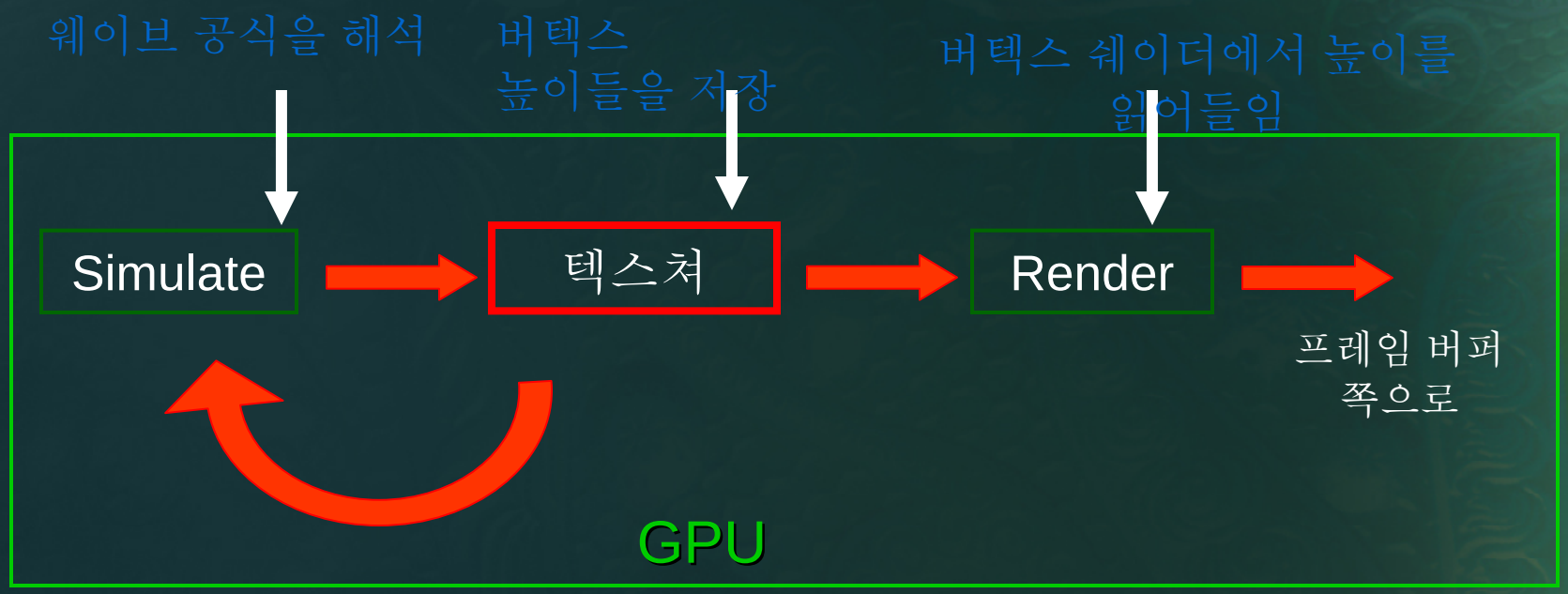
물 렌더링 – 알고리즘 개요

- 픽셀 쉐이더에서 물 시뮬레이션
  - 텍스쳐로 렌더링 (D3DFMT_A16B16G16R16F)
- 굴절 및 반사 맵 렌더링
- 물 표면 렌더링
  - VS3.0 버텍스 텍스쳐 페치를 통한 시뮬레이션 결과 사용
  - 교란 후 텍스쳐 좌표 계산
  - 프레넬(Fresnel) 함수를 이용한 굴절과 반사의 결합

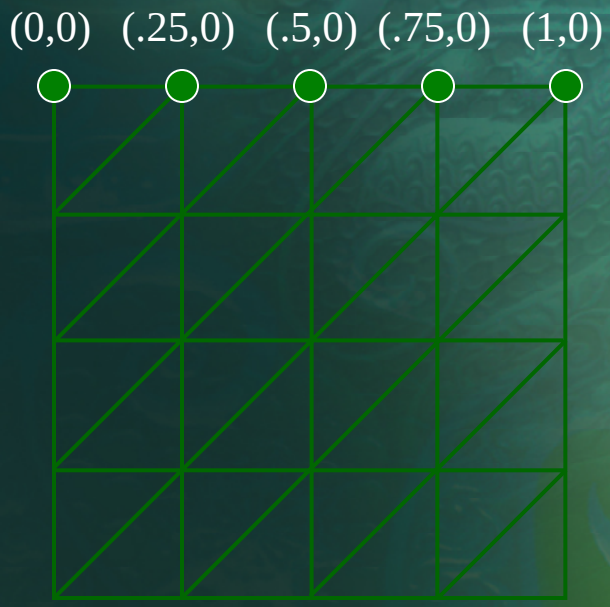


물 렌더링

- 물의 모자이크 평면



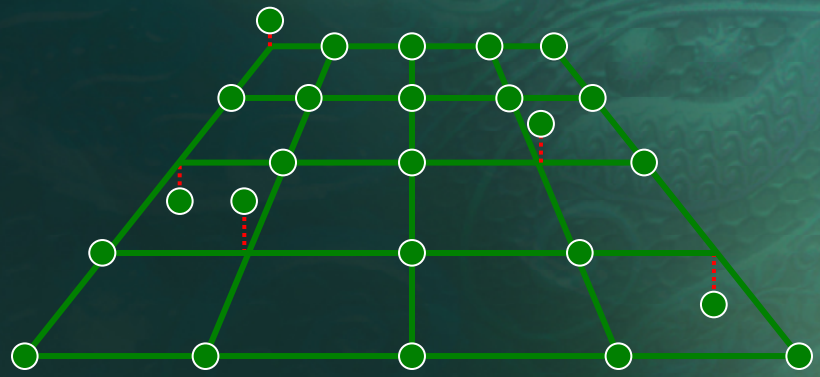
작동 방식



- 물 표면의 버텍스-메쉬 생성
  - 예: 128 X 128 버텍스
  - 버텍스의 메쉬 위치를 UV 좌표로 인코딩

버텍스 쉐이더 작업

- ‘높이 맵’ 읽기
  - 부동점 텍스쳐
  - 버텍스의 UV에서 텍스쳐 읽기
- 버텍스의 y에 결과 추가
- 버텍스 변형/투사

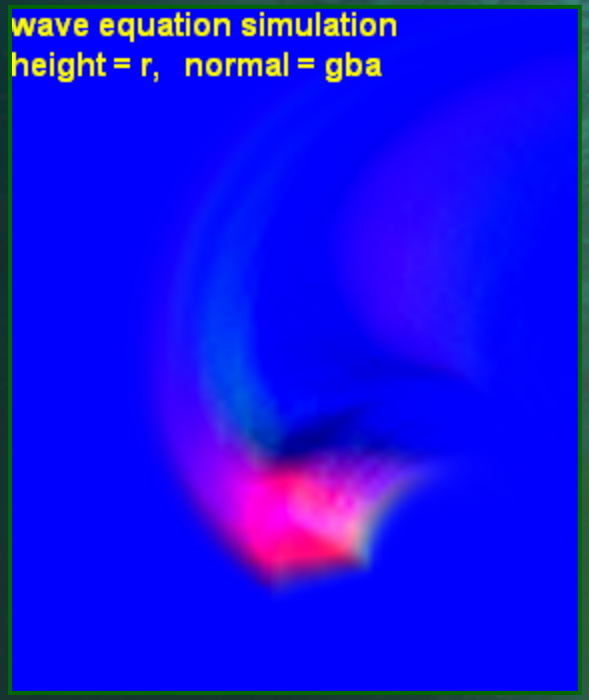


vertex.y += tex(u, v)

Out.pos = WorldViewProj * vertex

높이 맵(height map)은 역동적이다

- 모든 프레임을 업데이트
  - 텍스쳐로 렌더링 후 GPU 사용
- 벌레트(Verlet) 통합

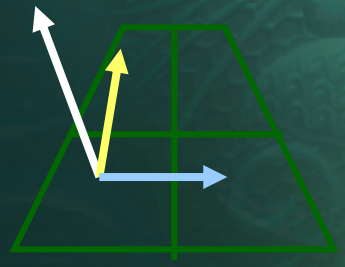


벌레트란?

$A = \sum (\text{neighbors}) - 4\,H_n$

$H_{n+1} = H_n + (H_n - H_{n-1}) + A$

$H_{n+1} = (2*H_n - H_{n-1}) + A$



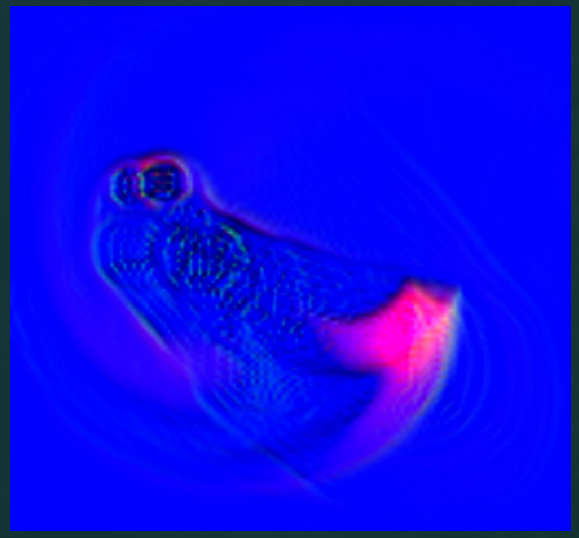
- 위치에서만 작동
  - 속도나 가속의 저장 불필요
- 위치에서 표준(normal)을 계산:

  N = 표준화(S x T)

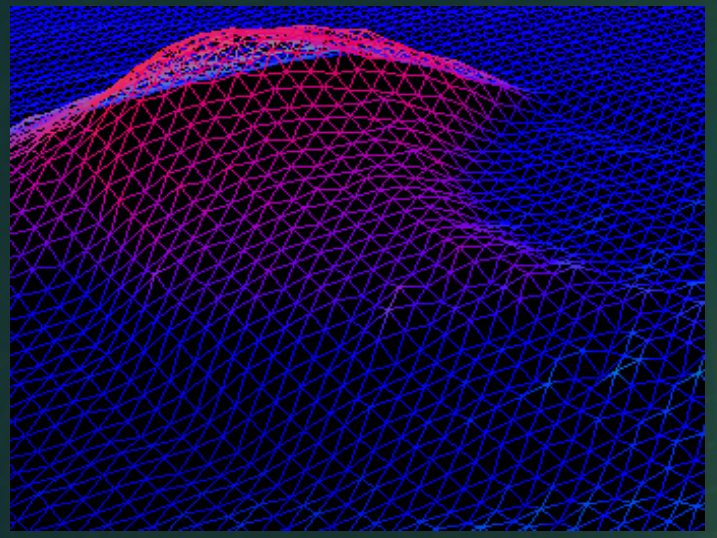
버텍스 텍스쳐 페치 (VS3.0)

- 버텍스 쉐이더는 버텍스 텍스쳐 페치를 사용하여 시뮬레이션 결과를 읽음

시뮬레이션 텍스쳐



Height Map 적용



높이 맵에 교란추가

- 디스플레이스먼트를 물에 블렌딩
  - 예: 보트, 바위, 해안
- 다음 프레임에서 벌레트(Verlet) 통합

- 부동점 렌더링 대상 블렌딩

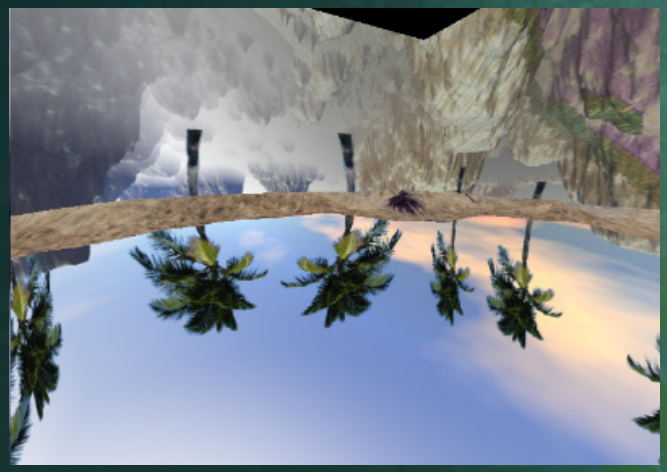
굴절 맵

- 카메라 시점에서 장면 렌더링

- 물 너머에 있는 대상을 렌더링
  - 물 평면을 사용한 지오메트리 클리핑
  - 수면위의 카메라
    - 수면 아래 지오메트리를 렌더링
  - 수면 아래의 카메라
    - 수면 위 지오메트리를 렌더링

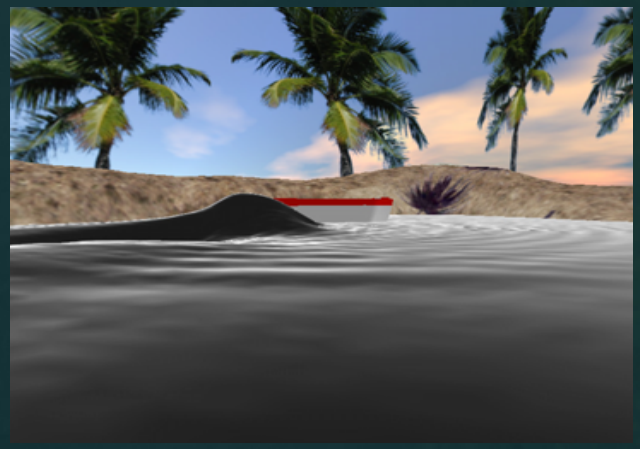
반사 맵

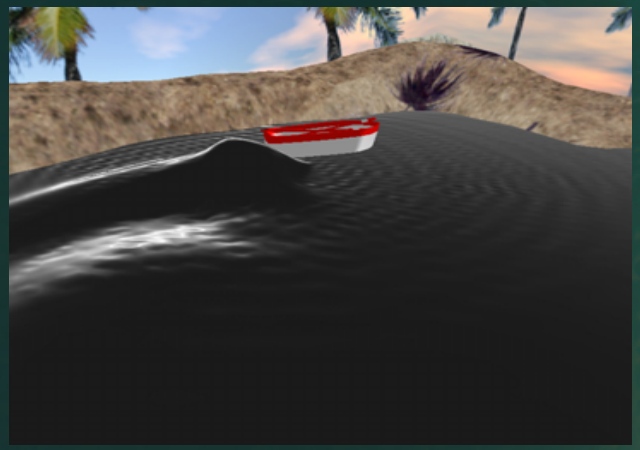
- 반사된 카메라 시점에서 장면 렌더링
  - 물 평면에 대한 보기 변형의 반사
- 물 평면을 사용하여 다시 클리핑
- 물의 이쪽 편에 있는 대상을 렌더링
  - 수면 아래의 카메라
    - 수면 아래 지오메트리를 렌더링
  - 수면 위의 카메라
    - 수면 위의 지오메트리를 렌더링



프레넬 반사 조건

- 반사/굴절의 양을 결정
- 대략 pow((1 - dot(eye, normal)), p)
  - 프레넬 조건 = 0 => 모두 굴절
  - 프레넬 조건 = 1 => 모두 반사

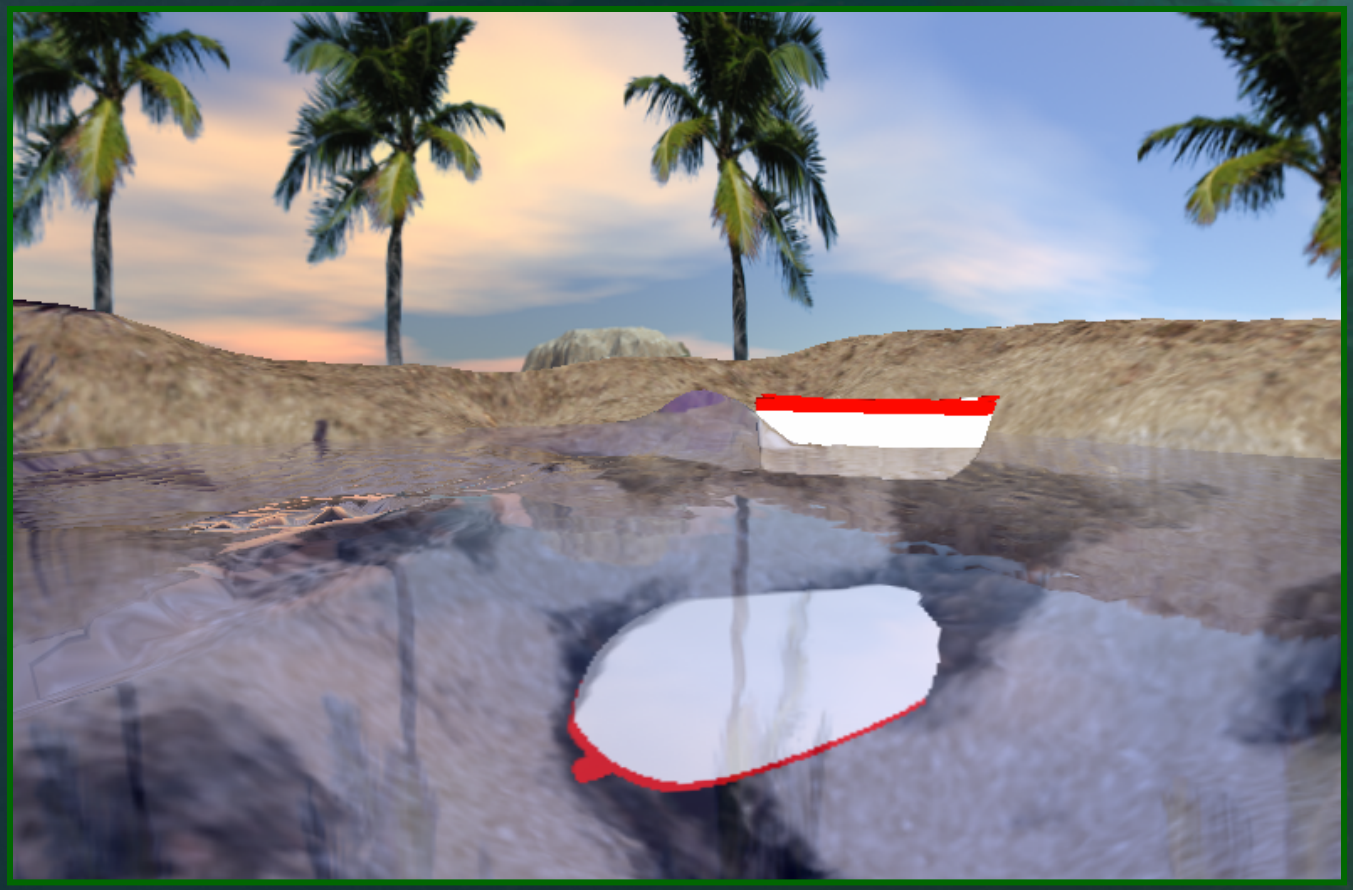




장점

- 빠른 속도!
  - 128x128 텍스쳐에서 시뮬레이션
  - GPU 표준에 따른 작은 규모
  - 시뮬레이션의 영향을 받지 않는 프레임 속도
- 적당한 수준의 기하학적 복잡성
  - 128x128 은 16k 버텍스

VTF 데모



반투명 및 산란

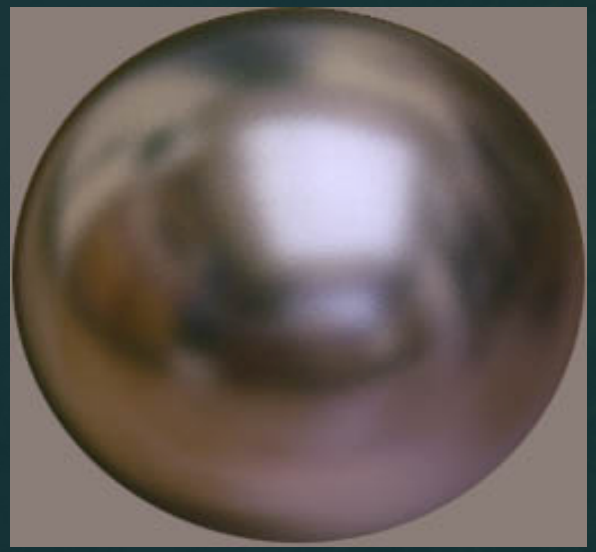
- 모든 materials은 반투명함(translucent)
  - 빛 파장길이(light wavelength)에 의존
- 빛은 모든 표면을 어느 정도는 관통한다
  - 다른 파장길이는
    - 다른 관통 깊이(penetration depths)를 가진다
    - 다른 감소(falloff) 대 깊이(depth)
- 흡수되거나 반사되지 않은 빛은 흩어지고(scatter) 다른 어떤 곳으로 빠져나갈 수 있다(exit)

반투명 기본

- 광학적 특성들(Optical Properties)
  - 흡수(Absorption) (확률 대 거리)
  - 산란(Scattering) (확률 대 거리 & 각도)
  - 장애물 변화(Impedance changes) (반사와 굴절)
    - 광학적 장애는 굴절의 인덱스를 결정함
- 모든 것은 흡수하고 산란한다(absorbs and scatters)
  - 유체, 고체, 기체, 심지어는 순수한 깨끗한 공기까지도
- 불투명도(Opacity), 투명도(transparency), 및 반투명도(translucency)
  - 흡수와 산란의 확률에 의해 다양함

불투명

- 흡수 및 산란 확률 높음
- 빛은 최단 경로로 이동
- 빛은 내부가 아니라 표면에서 나옴



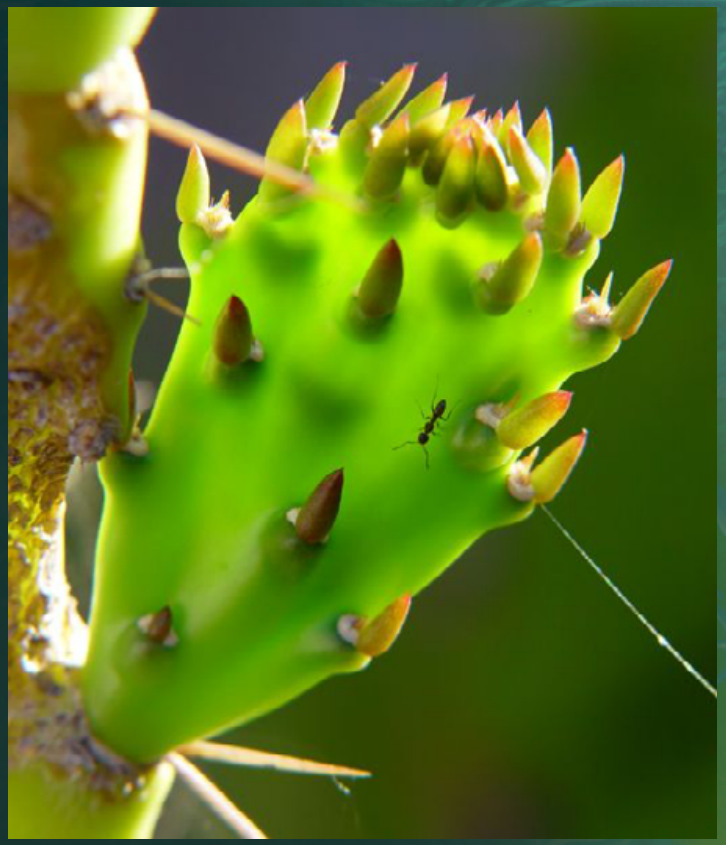
투명

- 흡수 및 산란 확률 낮음

Images courtesy of Leigh Van Der Byl

반투명

- 흡수 확률 낮음
- 산란 확률 높음



Leigh Van Der Byl

“실시간”적인 사고방식

- 외향을 구하라. 수학은 잊어버려라
  - 산란 관련 수학(scatter math)는 Hoffman & Preetham 을 참조
- 다양한 기법
  - 두께와 산란(두께 & scattering)을 위해서는 Depth-map 렌더링
  - 텍스쳐-공간 방산(texture-space diffusion)
- 필수사항들
  - 아티스트들이 친밀감을 느낄 수 있고 컨텐츠에 적합함
  - 정말 빨라야 함
  - 만일을 위한 대비책
  - 애니메이션 가능한 라이팅과 자체 음영화(self-shadowing)

깊이 맵

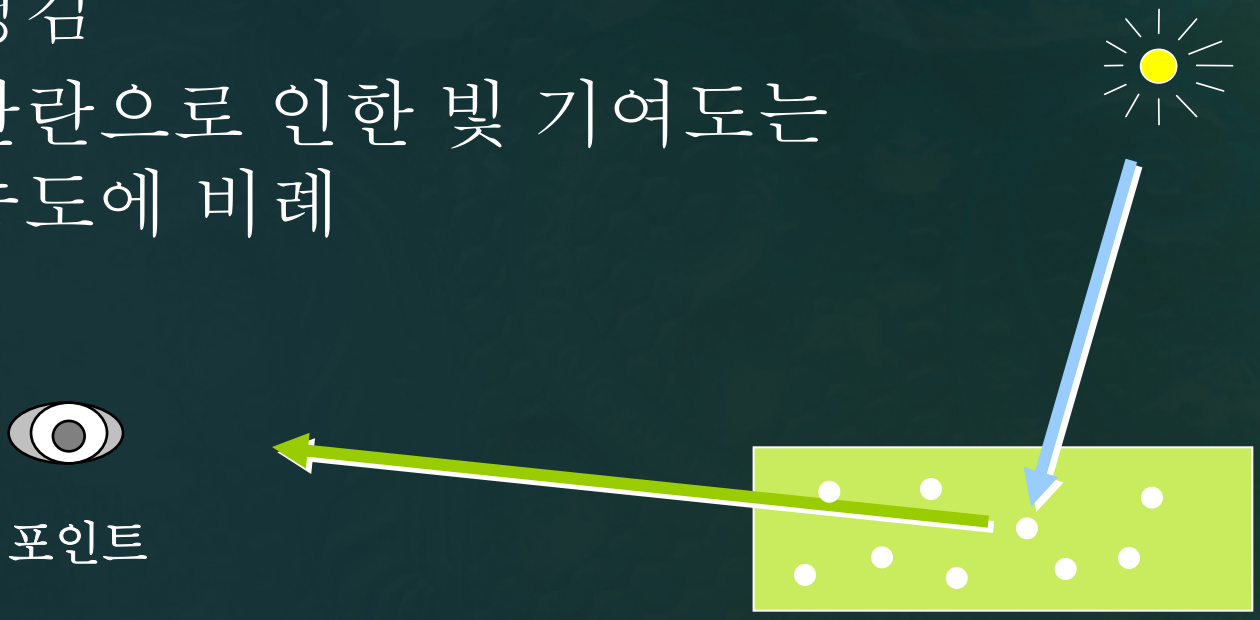
- 포그(Fog)는 평범한 폴리곤 모델이다
- 텍스쳐에 렌더링 (Render-to-Texture) 하는 패스들(passes) 은 포그 객체를 통하는 길이를 계산하기 위하여 사용됨
- ps.1.3
- ps.2.0 가 더 빠름
- ps.3.0 거 더 더 빠름

입체적 안개(Volume Fog) 기법

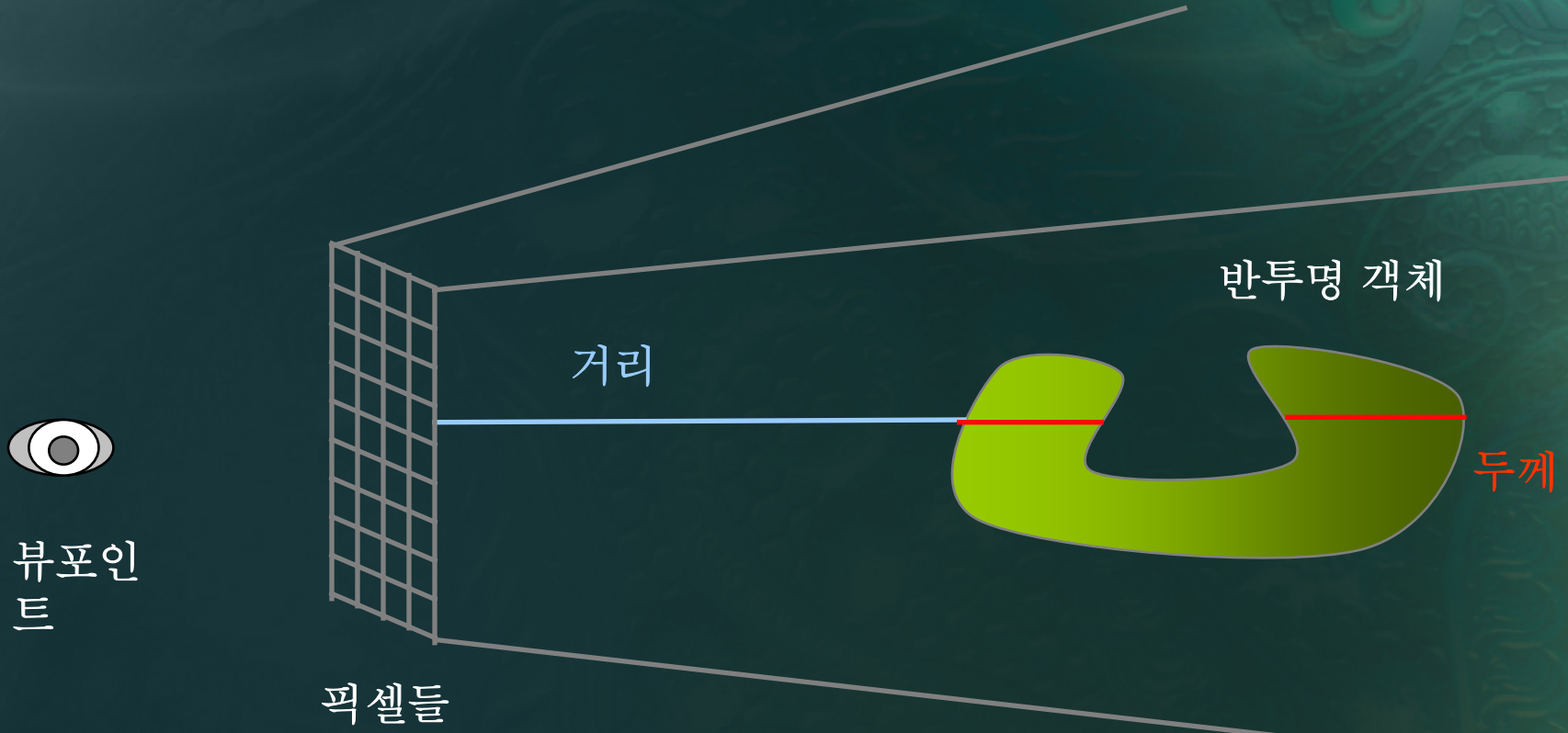
- 마이크로소프트 “볼륨 포그”(Volume Fog) DXSDK 데모 (Dan Baker)에서 착안
- [Mech01]에서 착안
- 카메라 P.O.V.의 일반 다각형 개체를 통해 농도 계산
  - 개체 앞면 및 뒷면의 깊이 렌더링
- 농도에서 색상 도출
- 단일 산란이 탁월

단일 산란

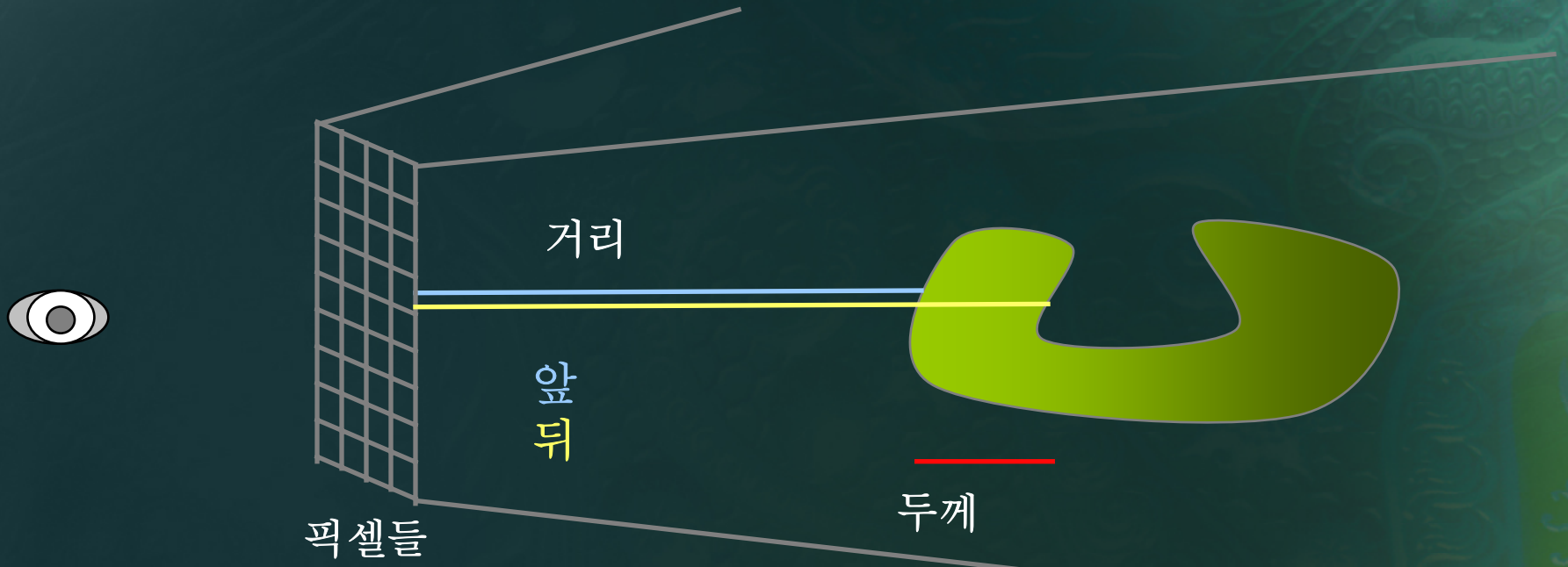
- 빛이 광원에서 눈으로 한번 튕김
- 산란으로 인한 빛 기여도는 농도에 비례



픽셀당 렌더링 농도

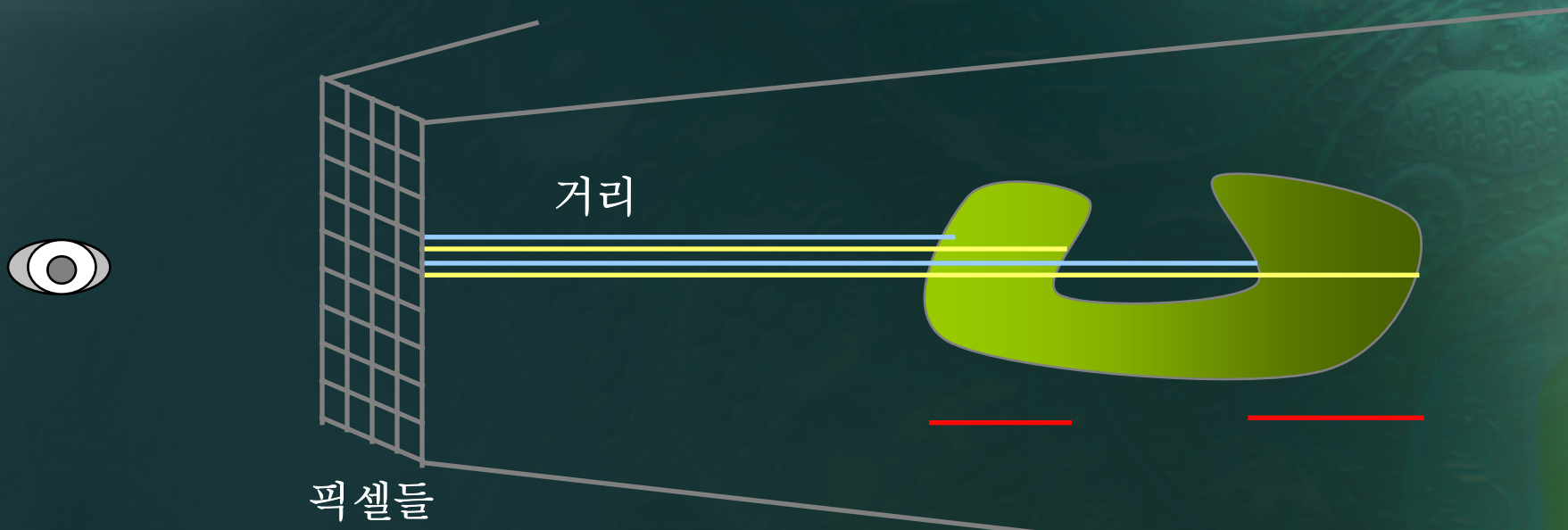


원거리 농도



두께 = 뒤 - 앞

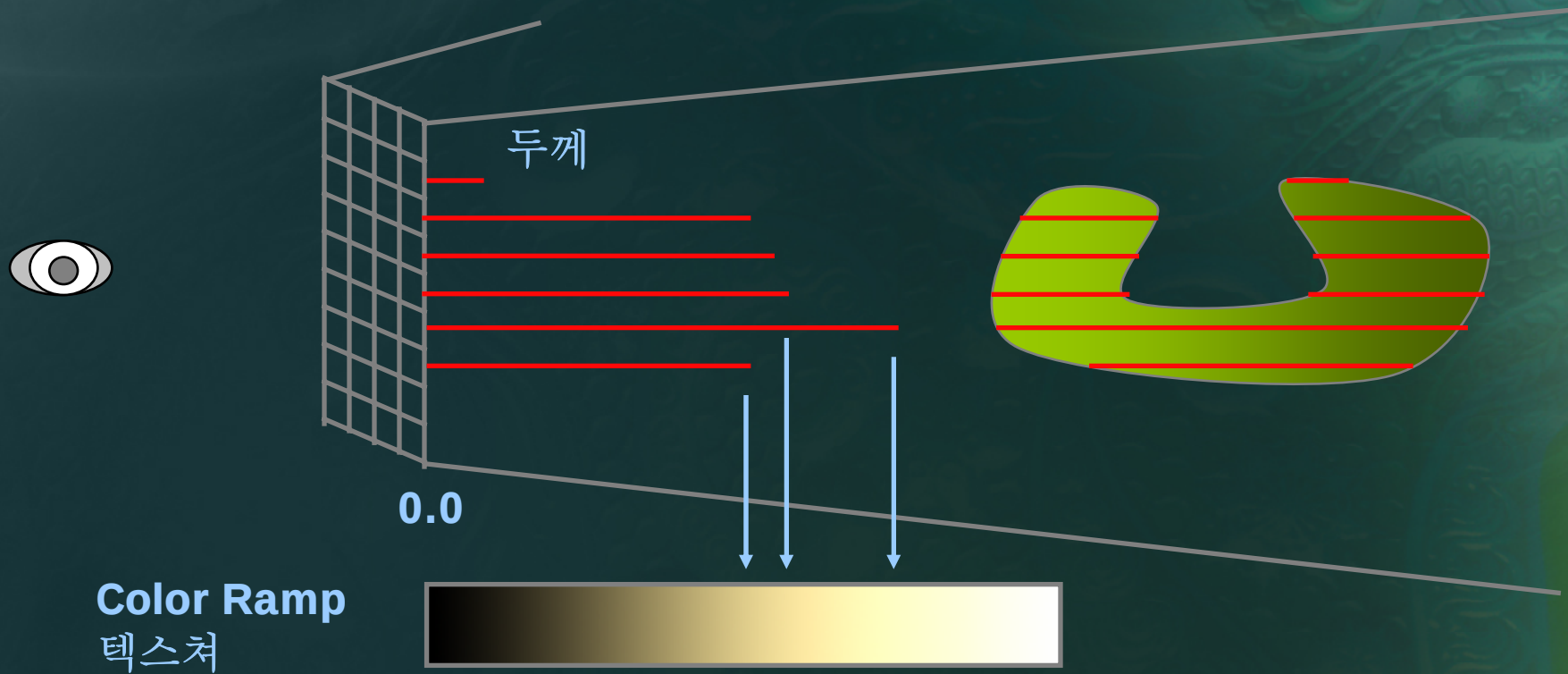
픽셀당 렌더링 농도



$$\text{두께}(Thickness) = \sum \text{뒤}(Back) - \sum \text{앞}(Front)$$

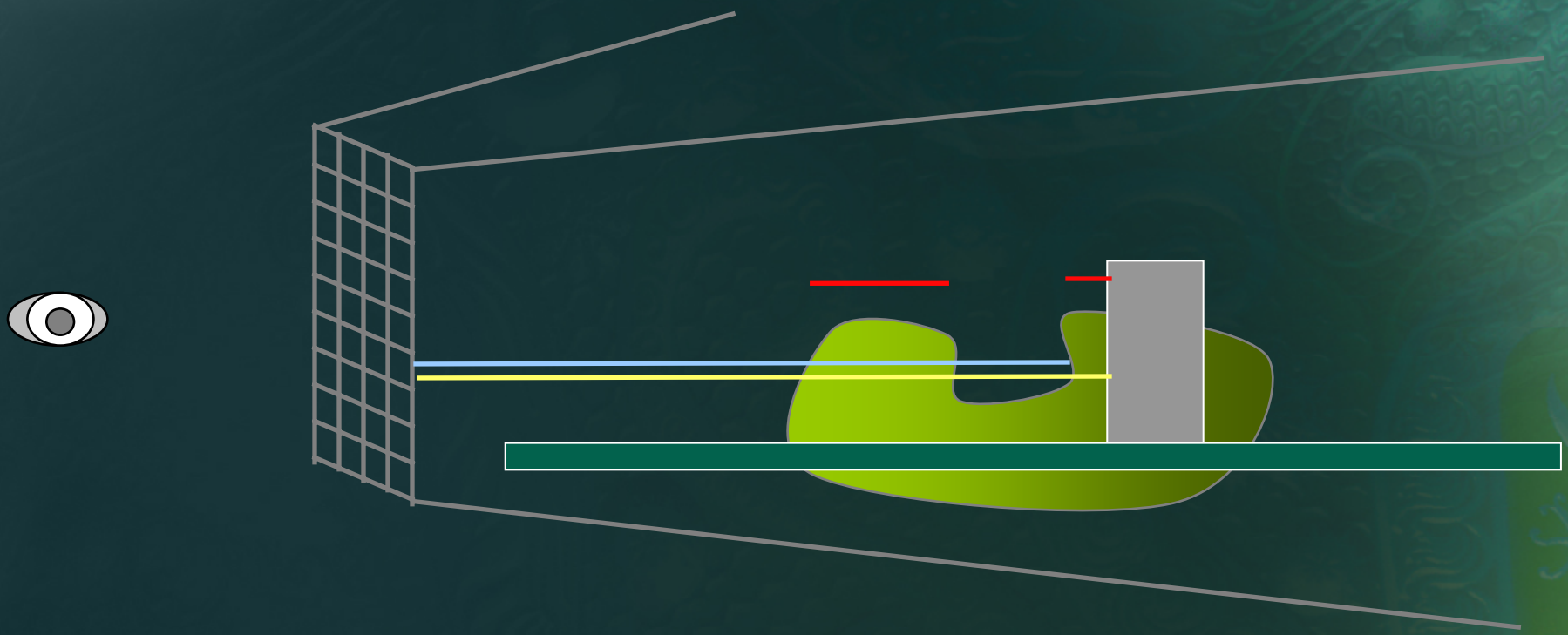
- 밀도가 균일한 개체의 농도는 쉽게 결정
- Z 버퍼 없음. 가색 블렌딩(Additive Blending) 사용

농도를 색상으로 변환



- 농도 스케일 ➔ TexCoord.x
- 컬러 램프 텍스쳐: 예술적 또는 수학적
- 외형제어가 용이

교차점

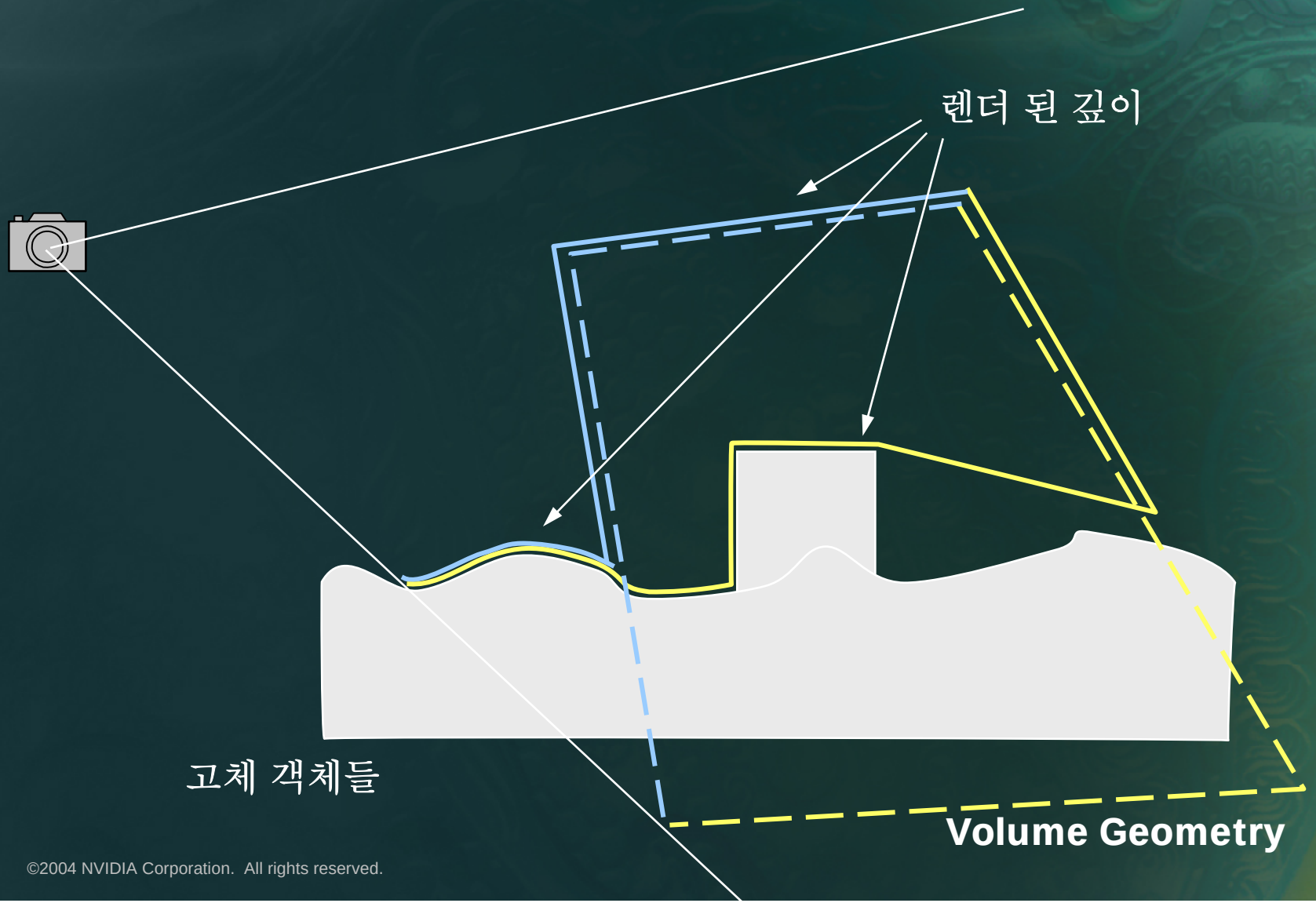


- 고체의 깊이 필요
- 입체 표면의 깊이가 아님

교차점 해결 방법

- 가장 가까운 고체의 깊이 필요
  - 텍스쳐로 렌더링
  - 픽셀 쉐이더에서 텍스쳐 읽기
- 입체의 각 표면을 렌더링할 때
  - 픽셀 쉐이더는 다음 중 적은 쪽을 출력
    - 그리고 있는 입체 삼각형의 깊이
    - 그리고 있는 픽셀에서 고체 깊이 (텍스쳐)
  - 깊이 테스트 비활성화
  - 출력 깊이를 프레임 버퍼로 가색 블렌딩

교차점 해결 방법



교차점 방법의 장점

장점

- 스텐실 불필요
- 멀티패스 불필요

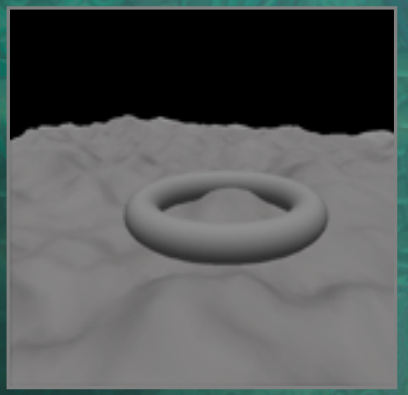
단점

- 깊이 렌더링 필요
  - 그 입체와 교차하는 모든 것의 깊이
  - 입체와 맞물릴 수 있는 모든 것의 깊이
    - 장면에 따라서는 무시

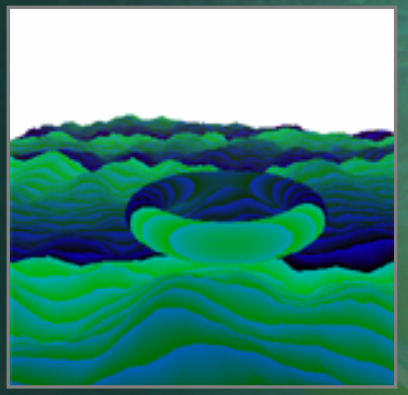
단계: 픽셀 쉐이더 2.0

1. 고체 객체들을 백버퍼(backbuffer)에 렌더링 함
   - 일반적 렌더링
2. 포그 볼륨(fog volumes)을 교차할 가능성이 있는 고체 객체들의 깊이를 렌더링 함
   - ARGB8 텍스쳐에, “S”
   - RGB로 인코드 된 깊이. 고 정밀도!
3. 포그 볼륨을 백페이스(backfaces)에 렌더링
   - ARGB8 텍스쳐에, “B”
   - 깊이(depth)를 합하기 위해서 Additive blend
   - 교차(intersection)에는 단순한 “S”

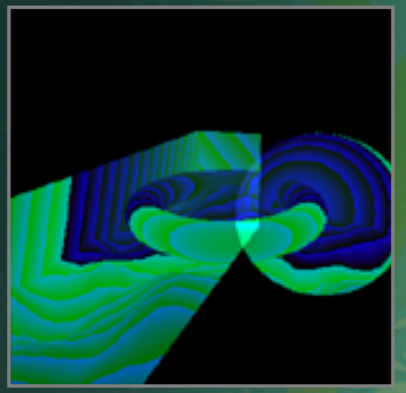
O



S



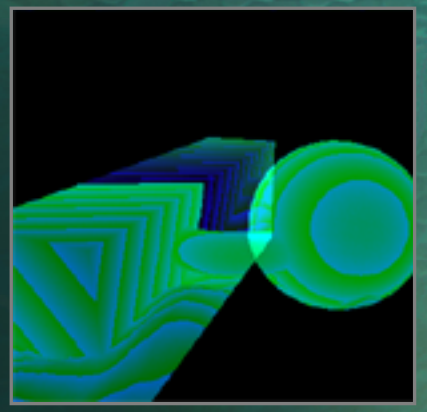
B

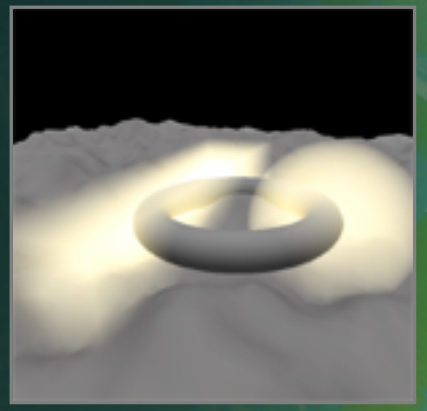


단계: PS.2.0 계속

4. 안개 용적의 전면 렌더링
   - ARGB8 텍스쳐에는 “F”
   - 가색 블렌딩을 통해 깊이 합계
   - 교차점을 통한 샘플 텍스쳐 “S”
5. 백 버퍼 위로 쿼드 렌더링
   - 샘플 “B” 및 “F”
   - 픽셀별로 농도 계산
   - 안개 색상 램프 텍스쳐를 사용하여 농도를 색상으로 변환
   - 색상을 장면에 블렌딩
   - 5 지침 ps.2.0 쉐이더

F





Final

PS.3.0 HW 개선

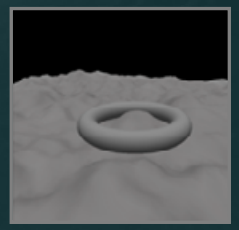
- 전방/후방 레지스터
- 다중 렌더링 대상(MRT)
- 부동점 프레임 버퍼 블렌딩
- 경로 수의 감소
- 렌더 대상 텍스쳐의 감소

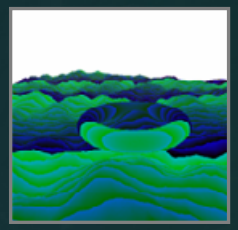
PS.3.0 대 PS.2.0

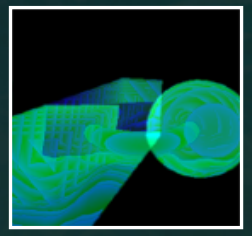
MRT

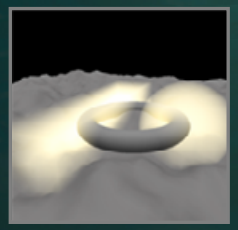
F/B 레지스터

ps.3.0 HW


RT-Tex

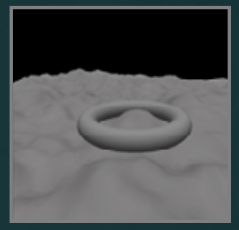

RT-Tex ‘O’

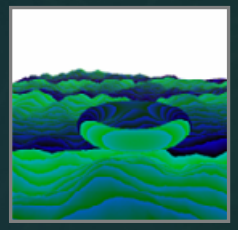

Floating point RT-tex

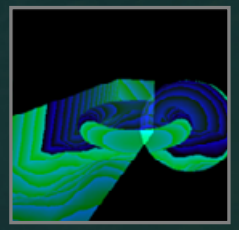

Backbuffer

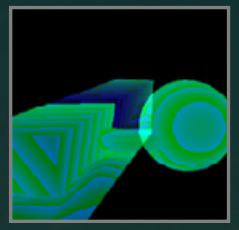
3 Passes

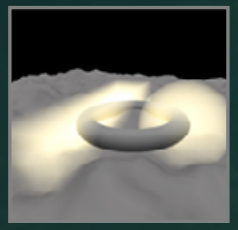
ps.2.0 HW


Backbuffer


RT-Tex ‘O’

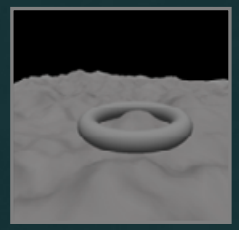

RT-Tex ‘B’

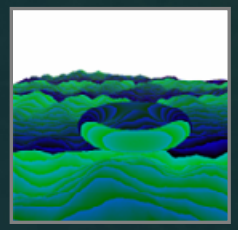

RT-Tex ‘F’

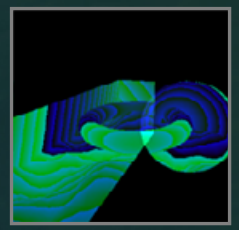

Backbuffer

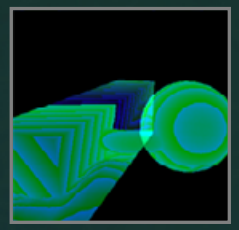
5 Passes

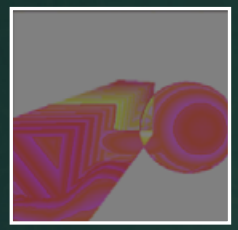
ps.1.3 HW

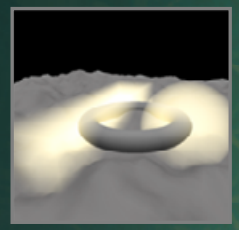






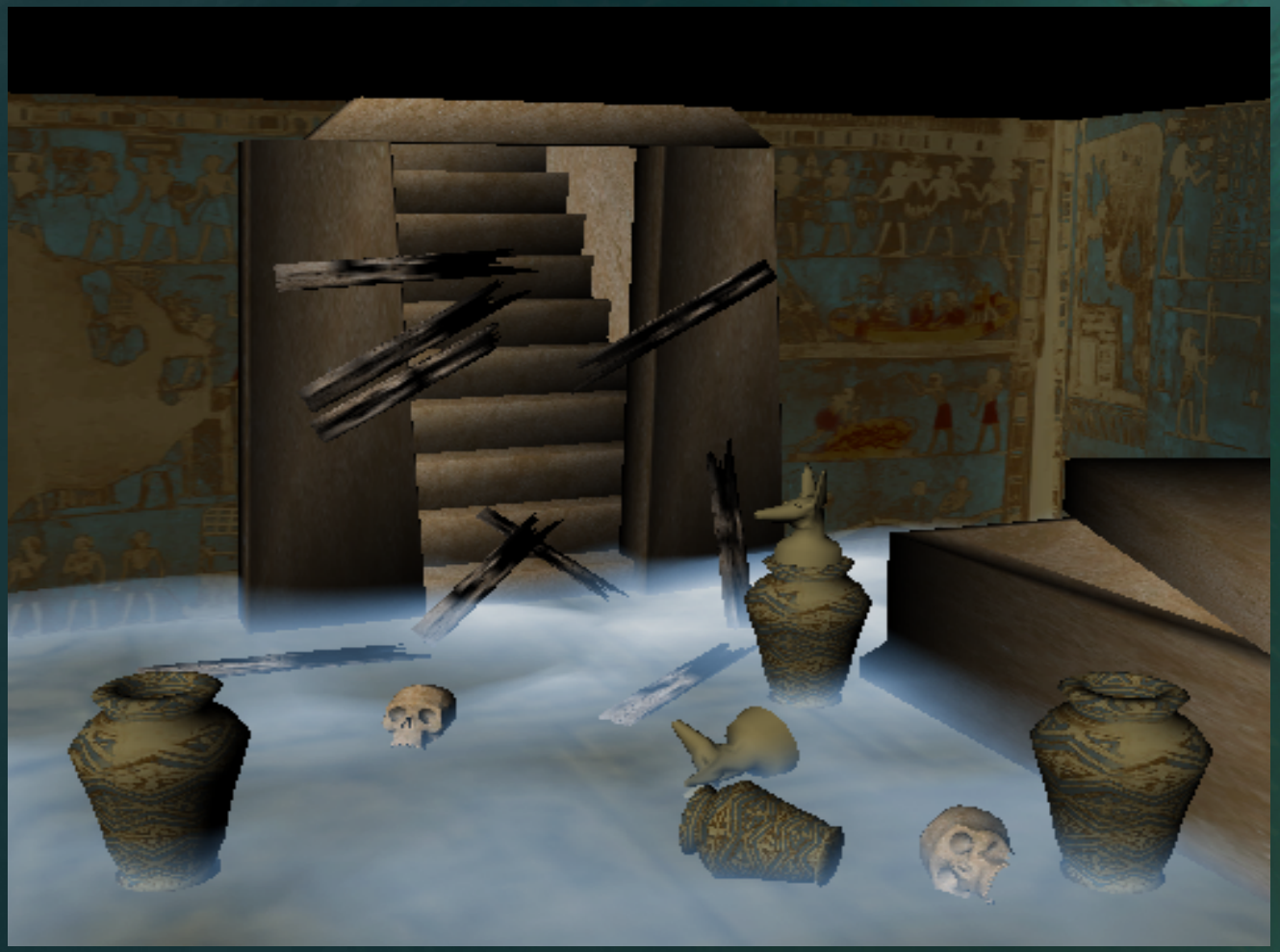






6

입체적 안개(Volume Fog) 데모



지연 렌더링(Deferred Rendering)

- 소개
  - 단순 계산으로 매개 변수를 오프스크린 버퍼로 렌더링
    - 보통 눈, 빛 벡터 등
  - 최종 조명 경로에 통합
- 렌더링 지연 목적
  - 복잡한 쉐이더를 사용한 겹쳐 그리기는 성능에 악영향을 줌
  - 최종 경로의 깊이 복잡성은 1
    - 성능 향상을 위한 PS3.0 브랜칭
    - 모든 조명 데이터를 한 경호에서출력하는 MRT

DX9 인스턴스화 API

- 소개
  - Draw call 1회로 같은 모델의 여러 인스턴를 드로잉
  - DIP 호출을 피하고 브랜칭 오버헤드를 최소화
- 사용 요건
  - DX 9.0c
  - VS/PS 3.0 기능의 그래픽 장치

사용하는 이유

- 속도.
  - 현재 대부분의 게임에서 병목 현상을 일으키는 가장 일반적인 원인은 그리기 호출 (draw call).
- 그렇다. Draw call이 나쁜 것은 누구나 알고 있다.
  - 그러나 월드 매트릭스 때문에 종종 draw call을 분리해야만 한다.
- 인스턴스화 API는 인스턴스별 draw 로직을 드라이버/하드웨어 수준으로 낮춤
  - D3D 와 드라이버 양쪽에서 draw call의 오버헤드 감소
  - 드라이버를 통해 인스턴스 간 상태 변경을 최소화

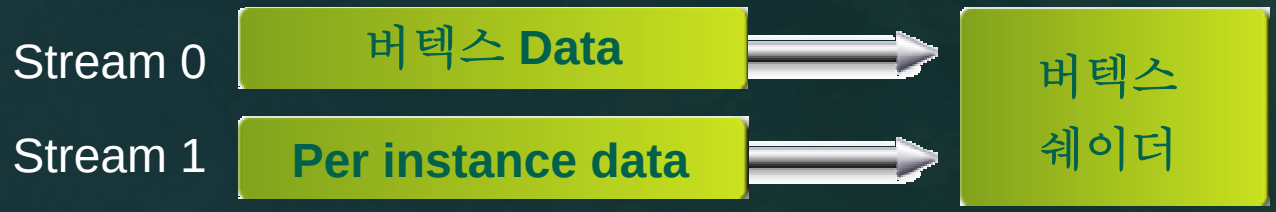
인스턴스화 예제

인스턴스화 사용사례

- 같은 모델의 많은 인스턴스가 있는 장면
  - 숲, 입자, 영혼, 군인
- 이동 또는 변경하는 인스턴스
  - 따라서 정적 버퍼는 실행불가
- 배치 크기가 큰 경우 유용성이 낮아짐
  - 드로우 1회당 poly 가 1K 이상이면 더 적은 draw call 바운드
  - 인스턴트화 작업의 고정 오버헤드
    - 별도의 버텍스 속성
    - 별도의 버텍스 쉐이더 작업

작동방식

- DX 인스턴싱 API 는 버텍스 스트림 주파수 분할자(VSF) API를 사용



- 1차 스트림은 모델 데이터의 단일 사본
- 2차 스트림에는 인스턴스별 데이터가 포함되며 1차 스트임을 렌더링할 때마다 스트림 포인터가 전진
  - IDirect 3DDevice9::SetStreamSourceFreq

작동 방식 (2)

- #0의 모듈러스, #1의 분할자
  - 스트림 0에서 루프
    - 각 루프는 한 “인스턴스”를 나타냄
  - 스트림 1을 분할
    - 따라서 각 “인스턴스” 이후에만 증분함

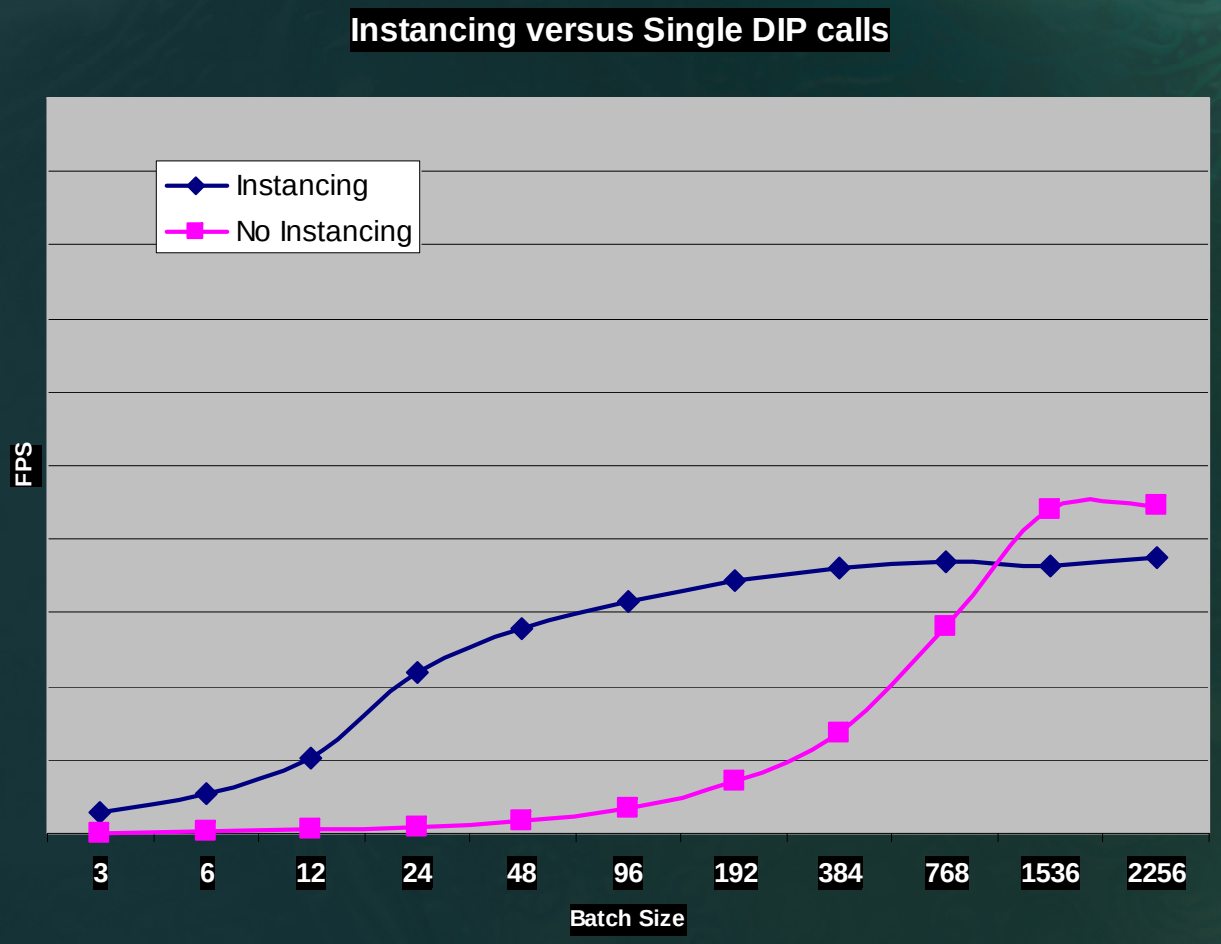
간단한 인스턴스화 예제

- 다각형 100개의 나무들
  - 스트림 0 에는 트리 모델 하나뿐
  - 스트림 1에는 모델 WVP 변형
    - 보기의 인스턴스를 기준으로 프레임별로 계산됨
  - 버텍스 쉐이더는 VS상수의 행렬 대신에 버텍스 스트림의 행렬을 사용한다는 점을 제외하고 일반 쉐이더와 동일
- 10k 트리를 그리면 draw call이 대폭 절감된다.
  - VB와 변형 전 버텍스를 조작할 수 있지만 이 작업은 대체로 까다로우며 대량의 데이터를 복제하게 된다.

다각형 100만개 테스트

- 메쉬 인스턴스를 그리는 실제 테스트이다.
- 변경 축은 메쉬당 다각형 개수이다.
- 장면의 총 다각형 수는 삼각형 1백만 개로 고정된다.
  - 따라서 메쉬 크기가 커질수록 인스턴스 개수는 적어진다.
- 매우 단순한 쉐이더
  - 더 복잡한 쉐이더는 장면의 행태를 변경한다.
    - 큰 픽셀 쉐이더는 메쉬 크기가 일정 수준을 넘어서면 병목 현상을 일으키며 따라서 인스턴스화는 항상 단일 DIP보다는 빠르다.

다각형 100만개 테스트 결과



다각형 100만개 테스트 결과 2

- 배치 크기가 작은 경우, draw call 별로 절약되므로 극도로 이익이다.
- 버텍스 스트림에 별도의 데이터가 추가되므로 고정 오버헤드가 있다.
  - 흔히 버텍스 속성 페치가 제약이 된다.
    - 버텍스 속도는 2배 이상 증가
  - 이 오버헤드는 시스템의 다른 병목 현상에 따라 문제가 될 수도 있고 안될 수도 있다.
- 작업하기 쉬운 지점은 여러가지 요건(CPU 속도,GPU 속도, 엔진 오버헤드 등) 에 따라 달라진다.

각종 개념

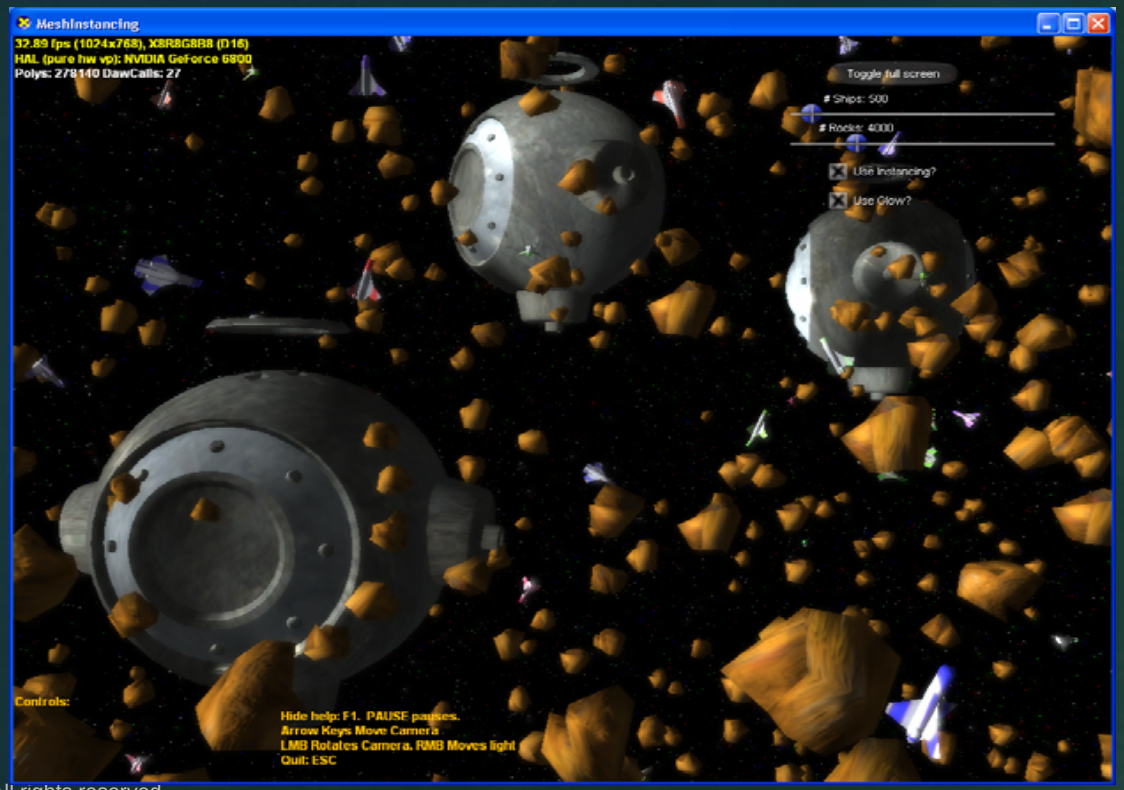
- VS 상수 사용
  - VS 상수 메모리를 각 인스턴스가 사용하는 데이터 섹션별로 분할한다.
  - 이 상수 메모리 섹션으로 옵셋하도록 인스턴스 스트림의 “색인”을 인코딩한다.
  - 그러면 너무 많은 버텍스 쉐이더 속성의 성능히트를 경감할 수 있다.
- 다른 텍스쳐의 인스턴스
  - 여러 텍스쳐를 결합하고 인스턴스 데이터를 사용하여 텍스쳐간에 도약한다.
  - 텍스쳐 페이지를 사용하여 UV옵셋을 2차 스트림에 넣는다.

각종 개념 (2)

- 색상 인스턴스
  - 인스턴스 색상을 누르고 각 인스턴스의 버텍스 색상을 바꾼다.
- <자신이 원하는 개념을 설명>
  - Draw call을 줄일 수 있다면 좋은 조건이다. 쉐이더 부담 및 버텍스 속성과 같은 추가 오버헤드에 주의해야 한다.
- 장면을 draw call 1회로 축소할 필요는 없다.
  - 인스턴스 4개씩 배치 처리할 수 있다면 #개의 draw 명령이 ¼로 감소되는 셈이다.

인스턴스화 데모

- 500+ ships, 4000+ rocks 의 스페이스 씬
- 복잡한 라이팅, 사후 처리(post-processing)
  - 몇몇 CPU 충돌(collision)또한 잘 작동한다
    - 실제로는 제한 요소가 된다
- 인스탄싱 이용으로 매우 빨라짐



인스탄싱(Instancing) 성능

- 인스탄싱 관련 백서
  - 인스탄싱을 이용한 성능 개선에 대한 자세한 정보는 엔비디아 게임 개발자 웹사이트에서 제공됩니다.
  - http://developer.nvidia.com
  - 그리기 명령(Draw call) 을 줄일 수 있는 여러 방법
  - 많은 그래픽 자료. :P

질문?

더 많은 정보를 원하신다면?

- http://developer.nvidia.com
- NVIDIA SDK
- bdudash@nvidia.com
  - 영어 혹은 일본어로 질문해 주십시오.

참고 문헌

[Mech01] Radomir Mech, “Hardware-Accelerated Real-Time Rendering of Gaseous Phenomena.”