실용적 성능 분석

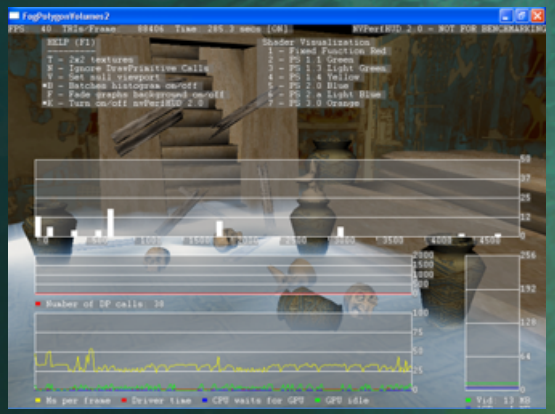
Koji Ashida
NVIDIA
Developer Technology Group

개요

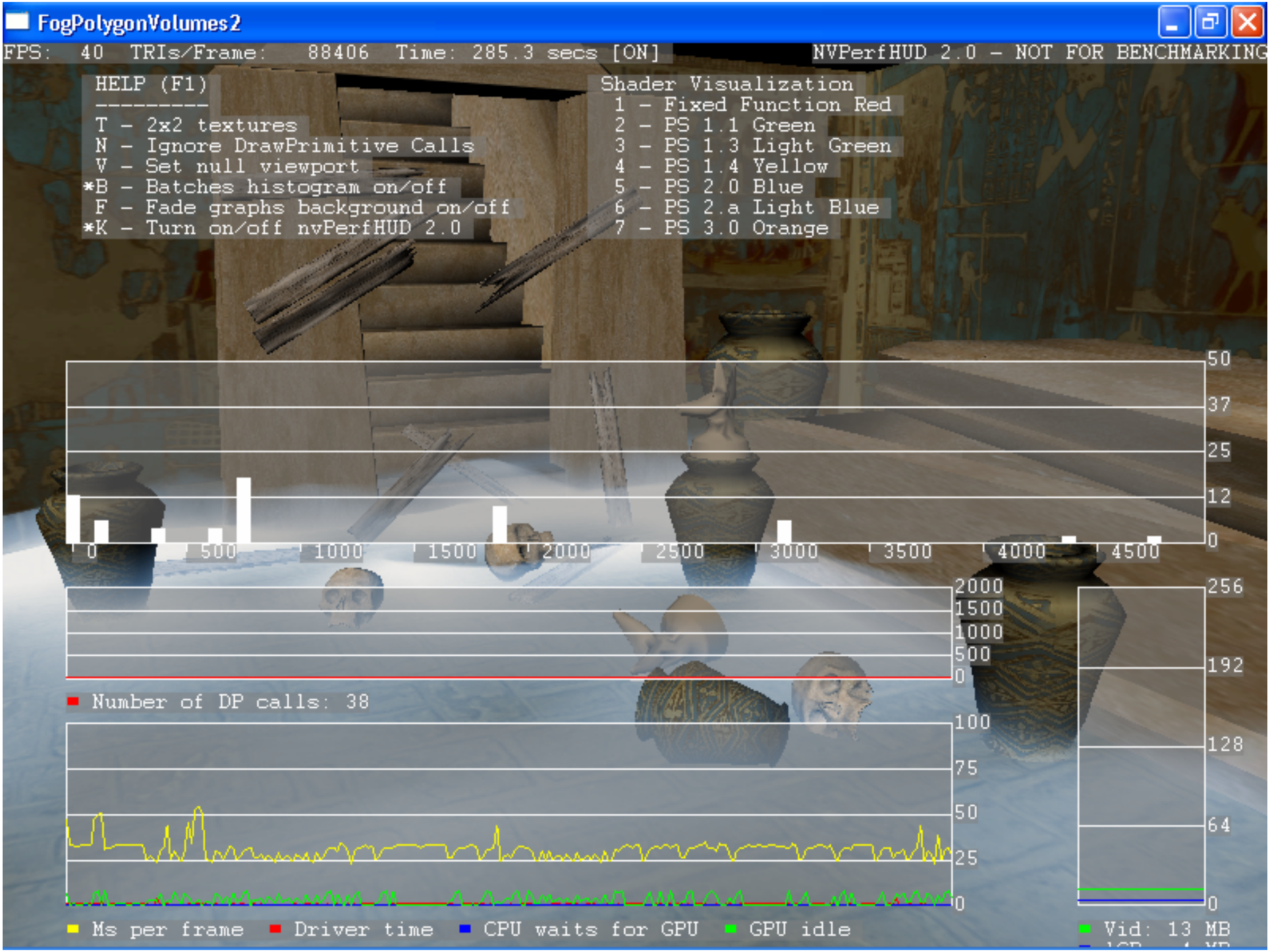
- 분석툴
- 파이프라인 병목현상 발견
- 문제를 지목하는 방법

분석 툴

NVPerfHUD

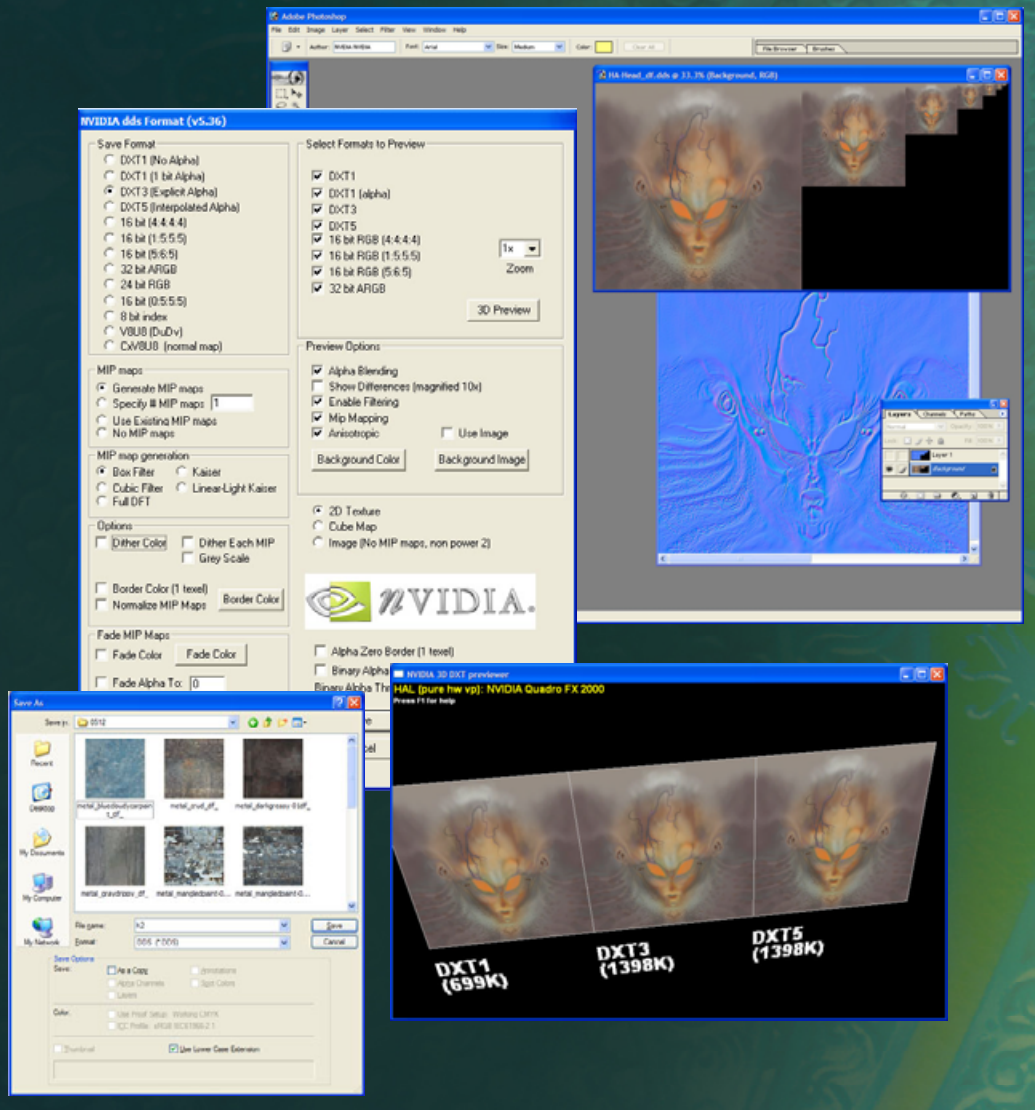


- 다양한 주요 통계의 그래프 오버레이
- 보고되는 측정값들은 다음을 포함:
  - GPU_Idle
  - Driver_Waiting
  - Time_in_Driver
  - Frame_Time
  - AGP / Video memory usage
  - 프레임 당 DIP/DP 호출의 숫자와 배치 크기의 히스토그램
- 현재 드라이버의 외부에서 모든 Direct3D9 애플리케이션 지원
- 이전에는 등록된 개발자들에게만 제공


FogPolygonVolumes2
FPS: 40 TRIs/Frame: 88406 Time: 285.3 secs [ON]
NVPerfHUD 2.0 - NOT FOR BENCHMARKING
HELP (F1)
T - 2x2 textures
N - Ignore DrawPrimitive Calls
V - Set null viewport
*B - Batches histogram on/off
F - Fade graphs background on/off
*K - Turn on/off nvPerfHUD 2.0
Shader Visualization
1 - Fixed Function Red
2 - PS 1.1 Green
3 - PS 1.3 Light Green
4 - PS 1.4 Yellow
5 - PS 2.0 Blue
6 - PS 2.a Light Blue
7 - PS 3.0 Orange
50
37
25
12
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
2000
1500
1000
500
0
Number of DP calls: 38
100
75
50
25
0
256
192
128
64
0
Ms per frame
Driver time
CPU waits for GPU
GPU idle
Vid: 13 MB

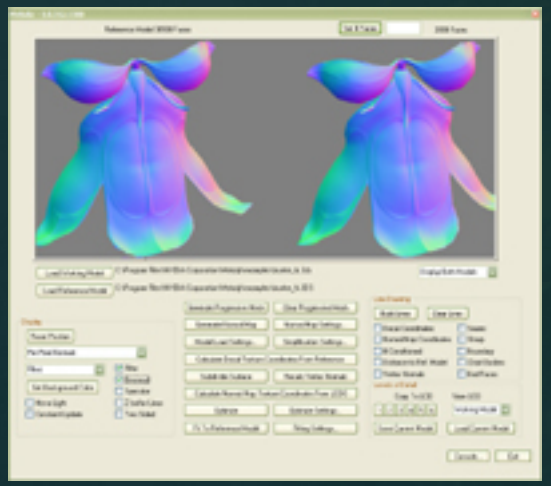
텍스쳐 도구와 플러그인

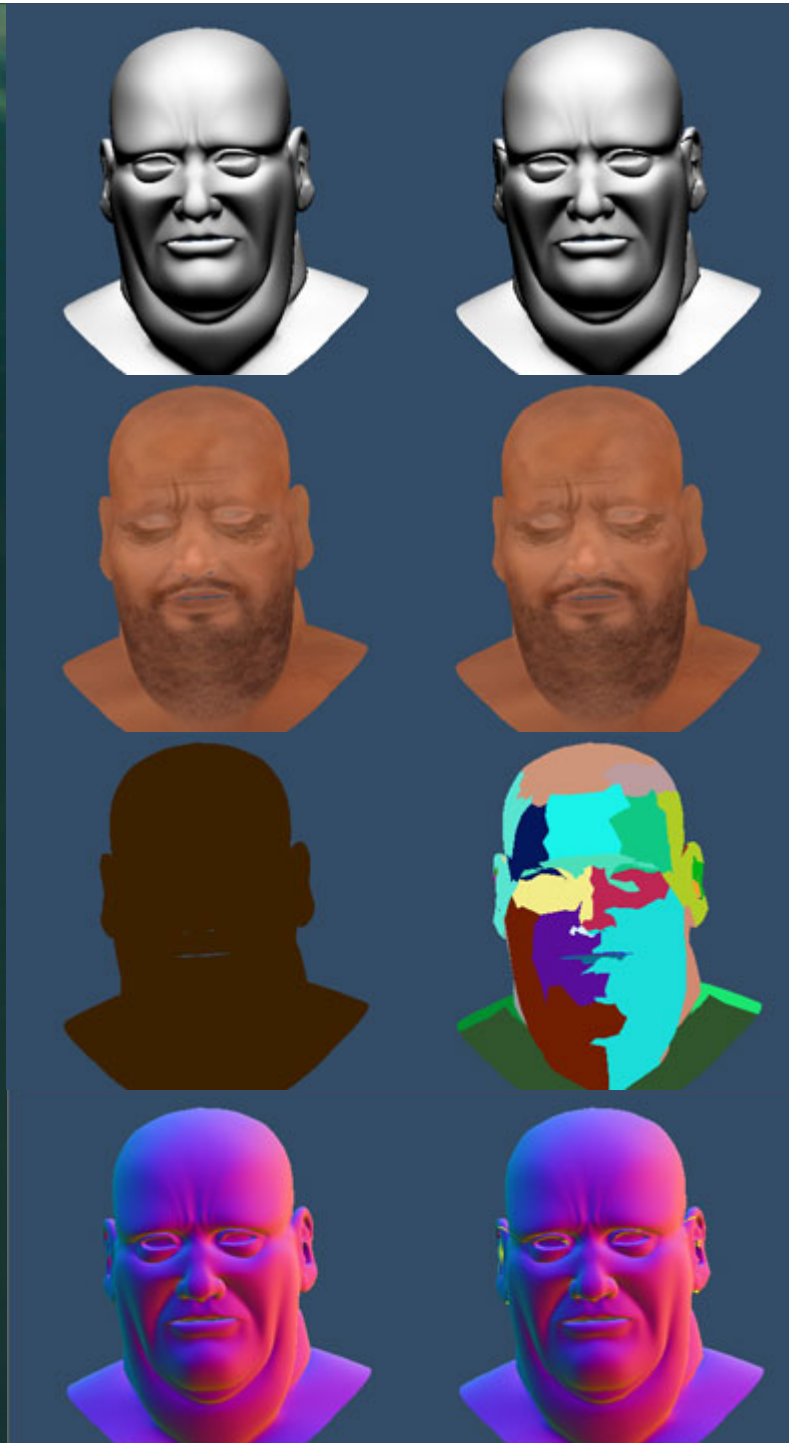
- 포토샵 플러그인:
  - DXT 압축 (.dds)
  - Normal Map 생성
  - 3D preview 와 diff
  - MIP map 생성
- 명령어 입력과.lib
- DDS thumbnail viewer
- Texture Atlas Viewer 와 Creation Utility

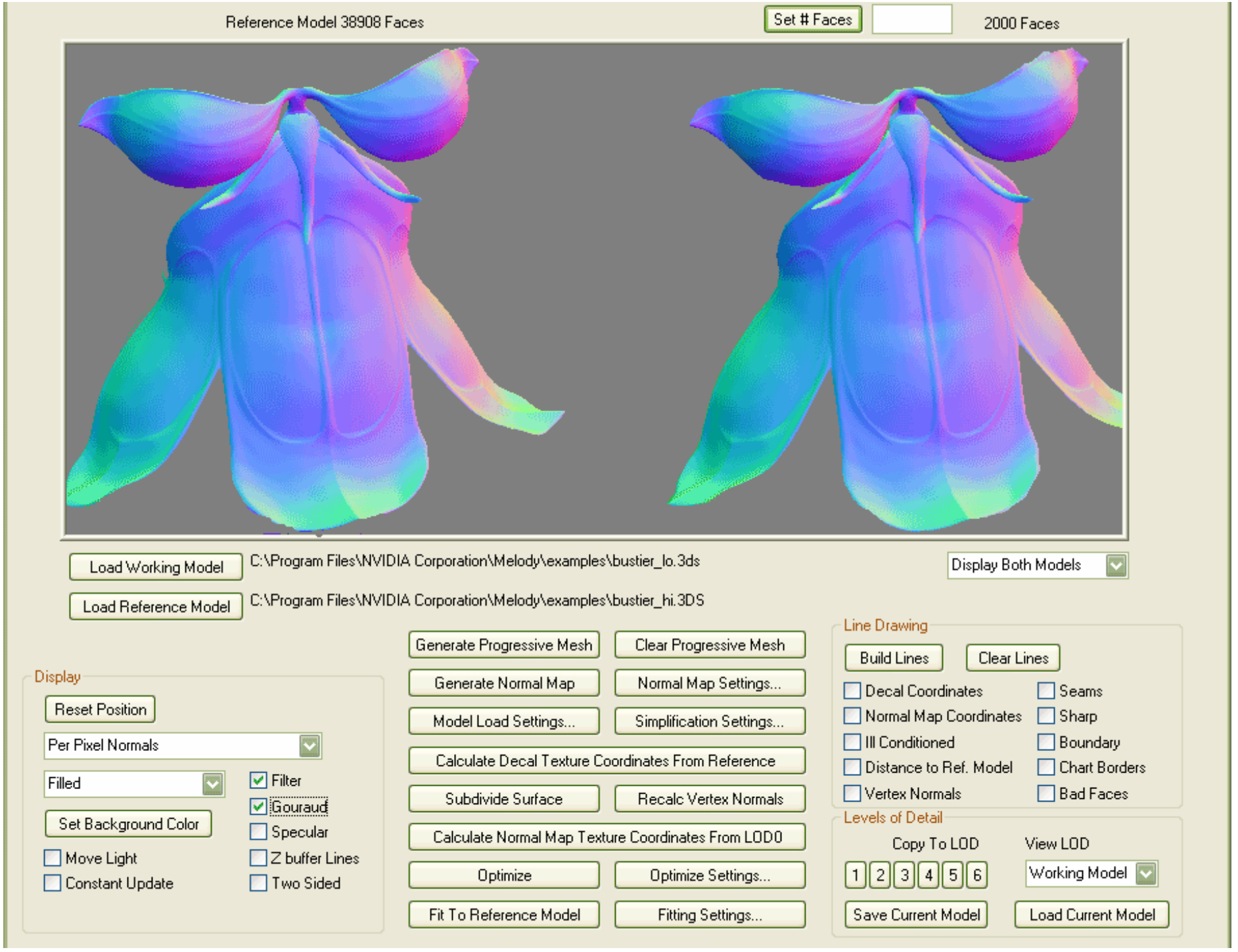


Melody

- 고 해상도 메시상에서 작동함 (~160만 폴리곤)
- advanced progressive mesh decimation을 사용하여 LOD를 생성
- Mesh 최적화 & 단순화
- Chart-based UV parameterization
- Raycast normal map generation






Reference Model 38908 Faces
Set # Faces
2000 Faces
Load Working Model
C:\Program Files\NVIDIA Corporation\Melody\examples\bustier_lo.3ds
Display Both Models
Load Reference Model
C:\Program Files\NVIDIA Corporation\Melody\examples\bustier_hi.3DS
Display
Reset Position
Per Pixel Normals
Filled
Set Background Color
Move Light
Constant Update
Filter
Gouraud
Specular
Z buffer Lines
Two Sided
Generate Progressive Mesh
Clear Progressive Mesh
Generate Normal Map
Normal Map Settings...
Model Load Settings...
Simplification Settings...
Calculate Decal Texture Coordinates From Reference
Subdivide Surface
Recalc Vertex Normals
Calculate Normal Map Texture Coordinates From LOD0
Optimize
Optimize Settings...
Fit To Reference Model
Fitting Settings...
Line Drawing
Build Lines
Clear Lines
Decal Coordinates
Normal Map Coordinates
Ill Conditioned
Distance to Ref. Model
Vertex Normals
Seams
Sharp
Boundary
Chart Borders
Bad Faces
Levels of Detail
Copy To LOD
1 2 3 4 5 6
View LOD
Working Model
Save Current Model
Load Current Model

Open EXR Library

- EXR 이미지 포맷 지원
- ILM이 개발한 HDR 이미지 포맷
  - www.openexr.org
- 컴포넌트 당 16비트 부동소수
- .NET 2003 라이브러리

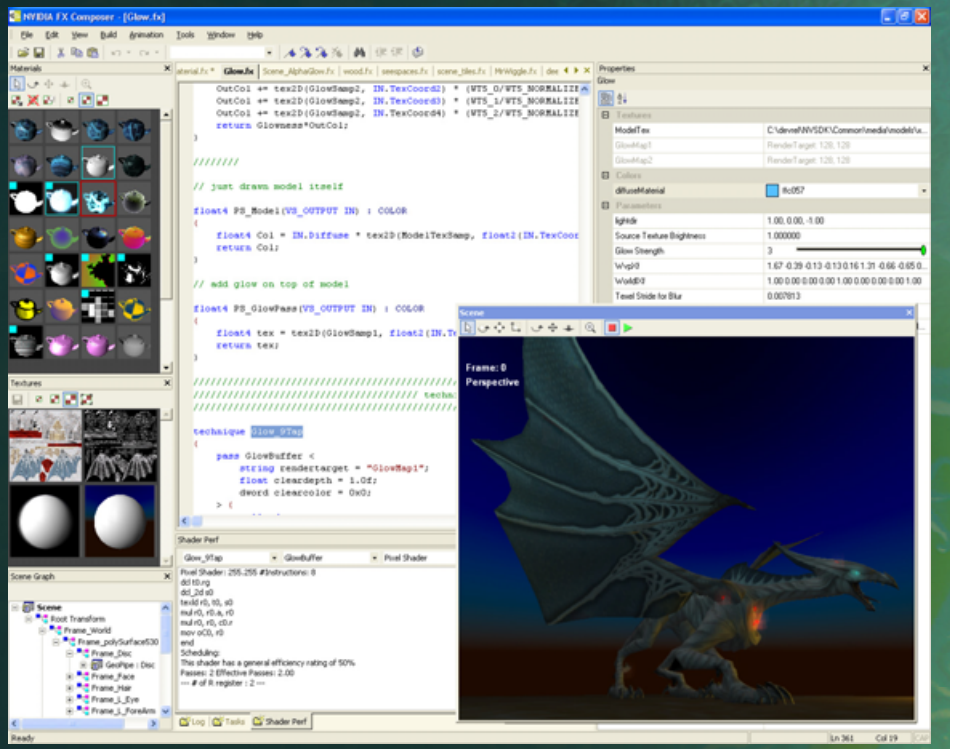
유틸리티, 라이브러리 및 그 외.

- NVShaderPerf
  - FX Composer Shader Perf panel과 같은 기술
  - HLSL, !!FP1.0, !!ARBfp1.0, PS1.x 및 PS2.x로 쓰여진 DirectX와 OpenGL shaders 지원
- NVMeshMender
  - 형상 이미지 문제 해결
  - 픽셀단위 라이팅(per-pixel lighting)을 위하여 메쉬를 준비
- NVTriStrip
  - 캐시를 고려하는 triangle stripping
  - 출력물은 strips 혹은 리스트

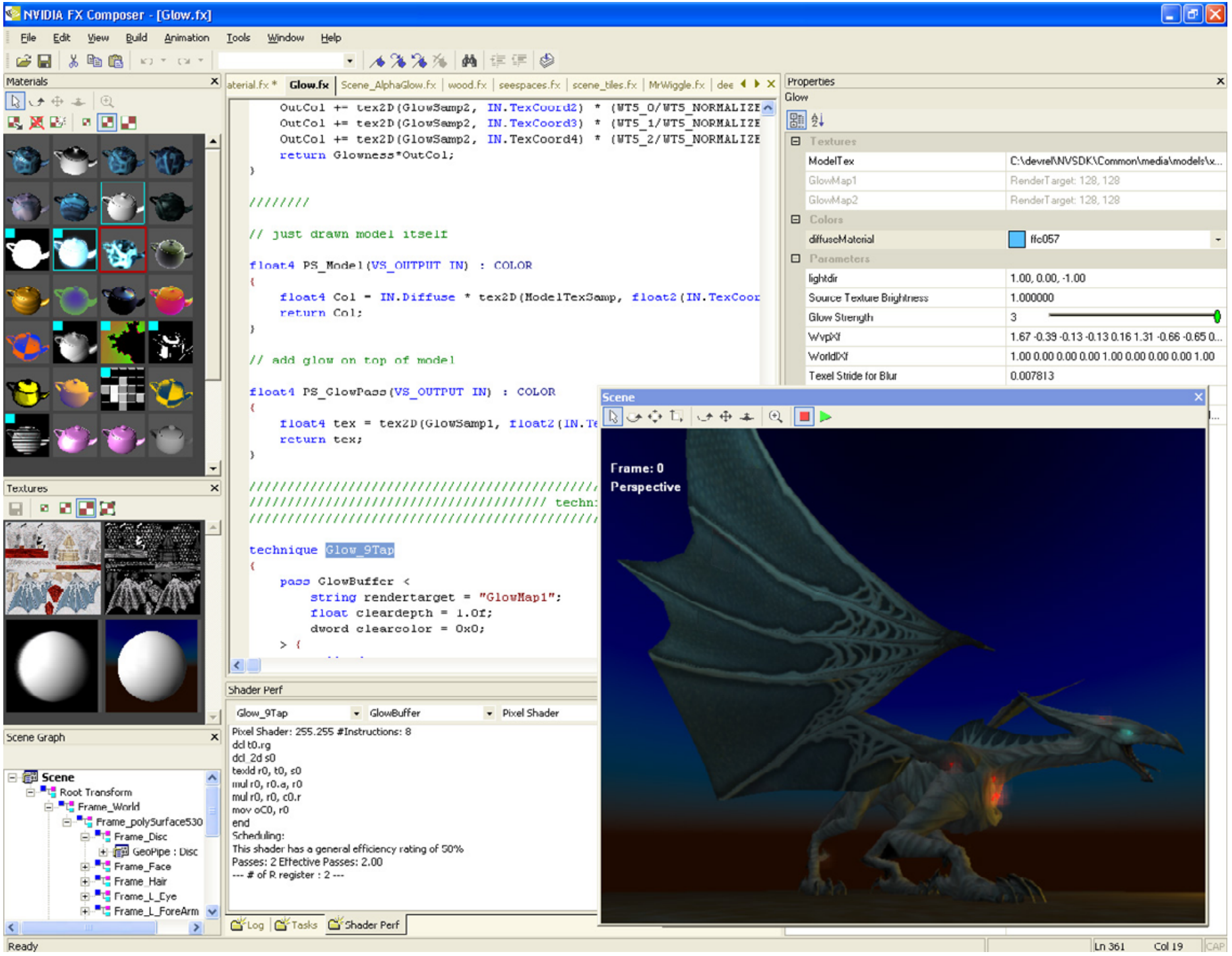
NVIDIA FX Composer

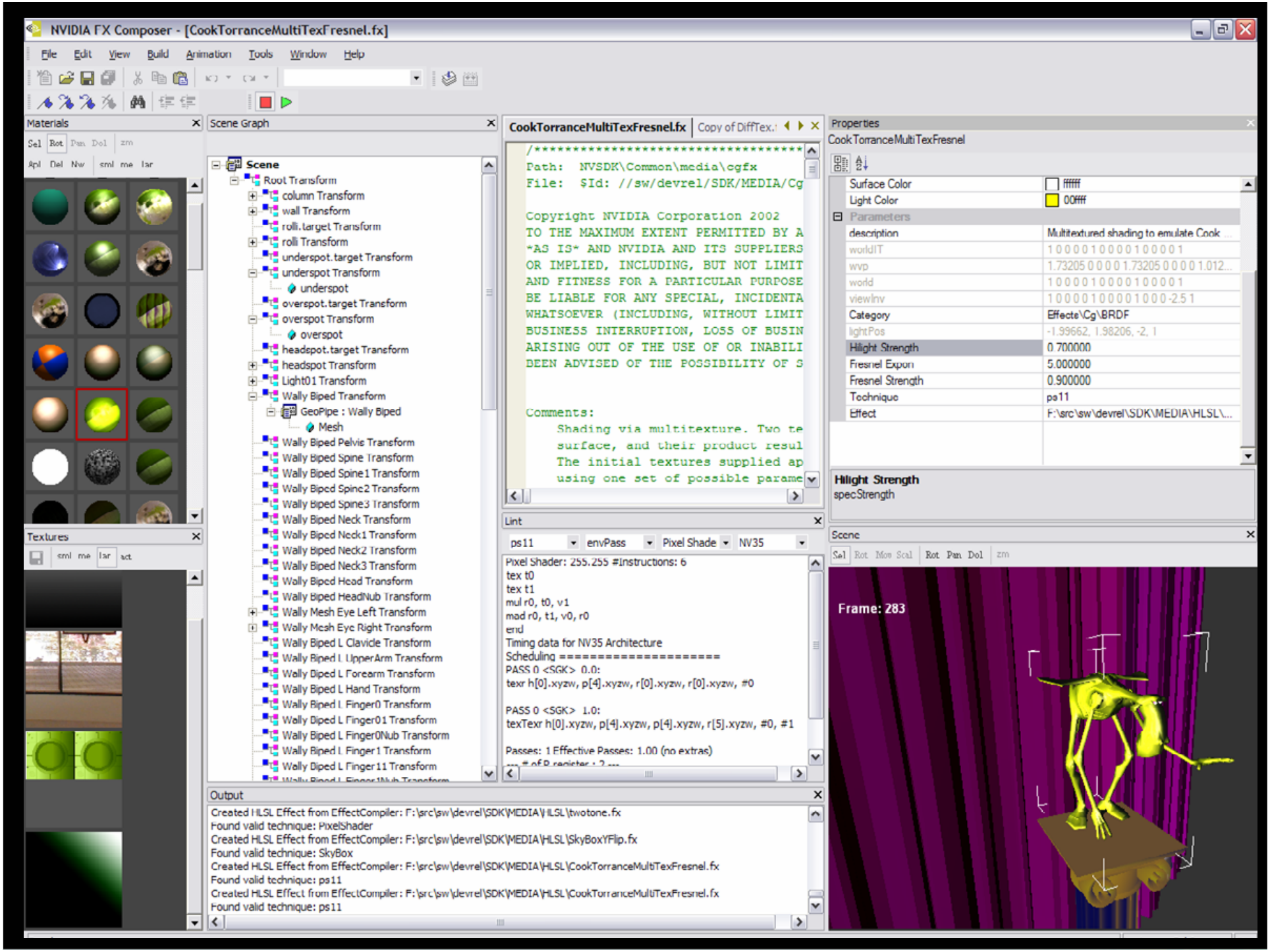
FX Composer는 **NVIDIA**만이 제공하는 것으로 개발자들에게 통합된 개발 환경에서 실시간으로 보고 기능들을 최적화 할 수 있는 고성능의 쉐이더를 제작할 수 있도록 힘을 실어주는 툴이다.

- 쉐이더를 고성능의 개발자 환경에서 **생성** 하십시오
- 쉐이더를 기본 쉐이더 디버깅 기능으로 **디버깅** 하십시오
- 쉐이더 성능을 향상된 분석 및 최적화 기능으로 **튜닝** 하십시오



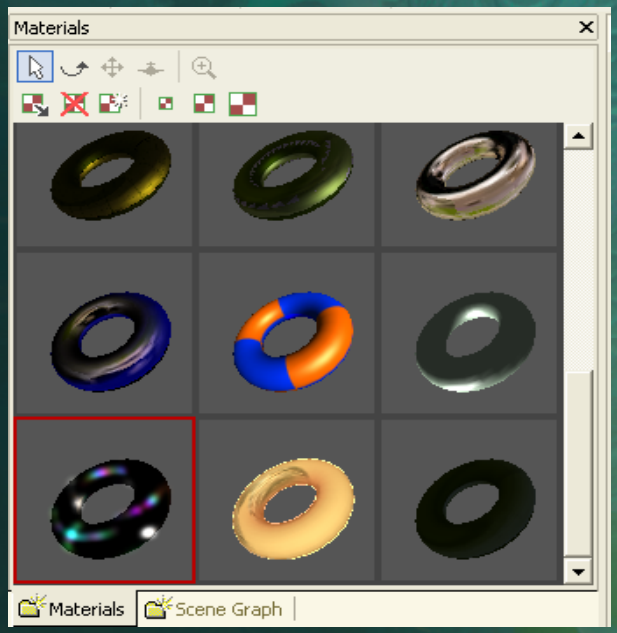
EverQuest® content courtesy Sony Online Entertainment Inc.

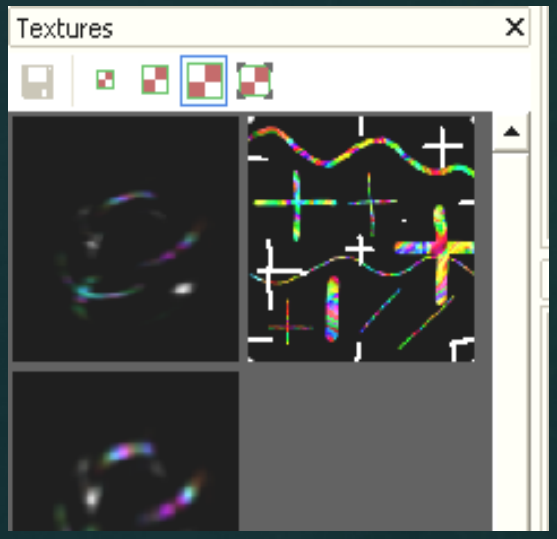

NVIDIA FX Composer - [Glow.fx]
File Edit View Build Animation Tools Window Help
Materials
Textures
Scene Graph
Scene
Root Transform
Frame_World
Frame_polySurface530
Frame_Disc
GeoPipe : Disc
Frame_Face
Frame_Hair
Frame_L_Eye
Frame_L_ForeArm
Glow.fx
Scene_AlphaGlow.fx
wood.fx
seespaces.fx
scene_tiles.fx
MrWiggle.fx
OutCol += tex2D(GlowSamp2, IN.TexCoord2) * (WTS_0/WTS_NORMALIZE
OutCol += tex2D(GlowSamp2, IN.TexCoord3) * (WTS_1/WTS_NORMALIZE
OutCol += tex2D(GlowSamp2, IN.TexCoord4) * (WTS_2/WTS_NORMALIZE
return Glowness*OutCol;
}
////////
// just drawn model itself
float4 PS_Model(VS_OUTPUT IN) : COLOR
{
float4 Col = IN.Diffuse * tex2D(ModelTexSamp, float2(IN.TexCoor
return Col;
}
// add glow on top of model
float4 PS_GlowPass(VS_OUTPUT IN) : COLOR
{
float4 tex = tex2D(GlowSamp1, float2(IN.Te
return tex;
}
technique Glow_9Tap
{
pass GlowBuffer <
string rendertarget = "GlowMap1";
float cleardepth = 1.0f;
dword clearcolor = 0x0;
> {
Shader Perf
Glow_9Tap
GlowBuffer
Pixel Shader
Pixel Shader: 255.255 #Instructions: 8
dcl t0.rg
dcl_2d s0
texld r0, t0, s0
mul r0, r0.a, r0
mul r0, r0, c0.r
mov oC0, r0
end
Scheduling:
This shader has a general efficiency rating of 50%
Passes: 2 Effective Passes: 2.00
--- # of R register : 2 ---
Log
Tasks
Shader Perf
Properties
Glow
Textures
ModelTex
C:\devrel\NVSDK\Common\media\models\x...
GlowMap1
RenderTarget: 128, 128
GlowMap2
RenderTarget: 128, 128
Colors
diffuseMaterial
ffc057
Parameters
lightdir
1.00, 0.00, -1.00
Source Texture Brightness
1.000000
Glow Strength
3
WvpXf
1.67 -0.39 -0.13 -0.13 0.16 1.31 -0.66 -0.65 0...
WorldXf
1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00
Texel Stride for Blur
0.007813
Scene
Frame: 0
Perspective
Ready
Ln 361
Col 19
CAP


NVIDIA FX Composer - [CookTorranceMultiTexFresnel.fx]
File Edit View Build Animation Tools Window Help
Materials
Scene Graph
Scene
Root Transform
column Transform
wall Transform
rolli.target Transform
rolli Transform
underspot.target Transform
underspot Transform
underspot
overspot.target Transform
overspot Transform
overspot
headspot.target Transform
headspot Transform
Light01 Transform
Wally Biped Transform
GeoPipe : Wally Biped
Mesh
Wally Biped Pelvis Transform
Wally Biped Spine Transform
Wally Biped Spine1 Transform
Wally Biped Spine2 Transform
Wally Biped Spine3 Transform
Wally Biped Neck Transform
Wally Biped Neck1 Transform
Wally Biped Neck2 Transform
Wally Biped Neck3 Transform
Wally Biped Head Transform
Wally Biped HeadNub Transform
Wally Mesh Eye Left Transform
Wally Mesh Eye Right Transform
Wally Biped L Clavicle Transform
Wally Biped L UpperArm Transform
Wally Biped L Forearm Transform
Wally Biped L Hand Transform
Wally Biped L Finger0 Transform
Wally Biped L Finger01 Transform
Wally Biped L Finger0Nub Transform
Wally Biped L Finger1 Transform
Wally Biped L Finger11 Transform
CookTorranceMultiTexFresnel.fx
Copy of DiffTex.
/**********
Path: NVSDK\Common\media\cgfx
File: $Id: //sw/devrel/SDK/MEDIA/Cg
Copyright NVIDIA Corporation 2002
TO THE MAXIMUM EXTENT PERMITTED BY A
"AS IS" AND NVIDIA AND ITS SUPPLIERS
OR IMPLIED, INCLUDING, BUT NOT LIMIT
AND FITNESS FOR A PARTICULAR PURPOSE
BE LIABLE FOR ANY SPECIAL, INCIDENTA
WHATSOEVER (INCLUDING, WITHOUT LIMIT
BUSINESS INTERRUPTION, LOSS OF BUSIN
ARISING OUT OF THE USE OF OR INABILI
BEEN ADVISED OF THE POSSIBILITY OF S
Comments:
Shading via multitexture. Two te
surface, and their product resul
The initial textures supplied ap
using one set of possible parame
Properties
CookTorranceMultiTexFresnel
Surface Color ffffff
Light Color 00ffff
Parameters
description Multitextured shading to emulate Cook
worldIT 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
wvp 1.73205 0 0 0 0 1.73205 0 0 0 0 1.012...
world 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
viewInv 1 0 0 0 0 1 0 0 0 0 1 0 0 0 -2.5 1
Category Effects\Cg\BRDF
lightPos -1.99662, 1.98206, -2, 1
Hilight Strength 0.700000
Fresnel Expon 5.000000
Fresnel Strength 0.900000
Technique ps11
Effect F:\src\sw\devrel\SDK\MEDIA\HLSL\...
Hilight Strength
specStrength
Lint
ps11 envPass Pixel Shade NV35
Pixel Shader: 255.255 #instructions: 6
tex t0
tex t1
mul r0, t0, v1
mad r0, t1, v0, r0
end
Timing data for NV35 Architecture
Scheduling ==========
PASS 0 <SGK> 0.0:
texr h[0].xyzw, p[4].xyzw, r[0].xyzw, r[0].xyzw, #0
PASS 0 <SGK> 1.0:
texTexr h[0].xyzw, p[4].xyzw, p[4].xyzw, r[5].xyzw, #0, #1
Passes: 1 Effective Passes: 1.00 (no extras)
Textures
Scene
Rot Pan Dol
Frame: 283
Output
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\twotone.fx
Found valid technique: PixelShader
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\SkyBoxYFlip.fx
Found valid technique: SkyBox
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\CookTorranceMultiTexFresnel.fx
Found valid technique: ps11
Created HLSL Effect from EffectCompiler: F:\src\sw\devrel\SDK\MEDIA\HLSL\CookTorranceMultiTexFresnel.fx
Found valid technique: ps11

Material과 텍스쳐

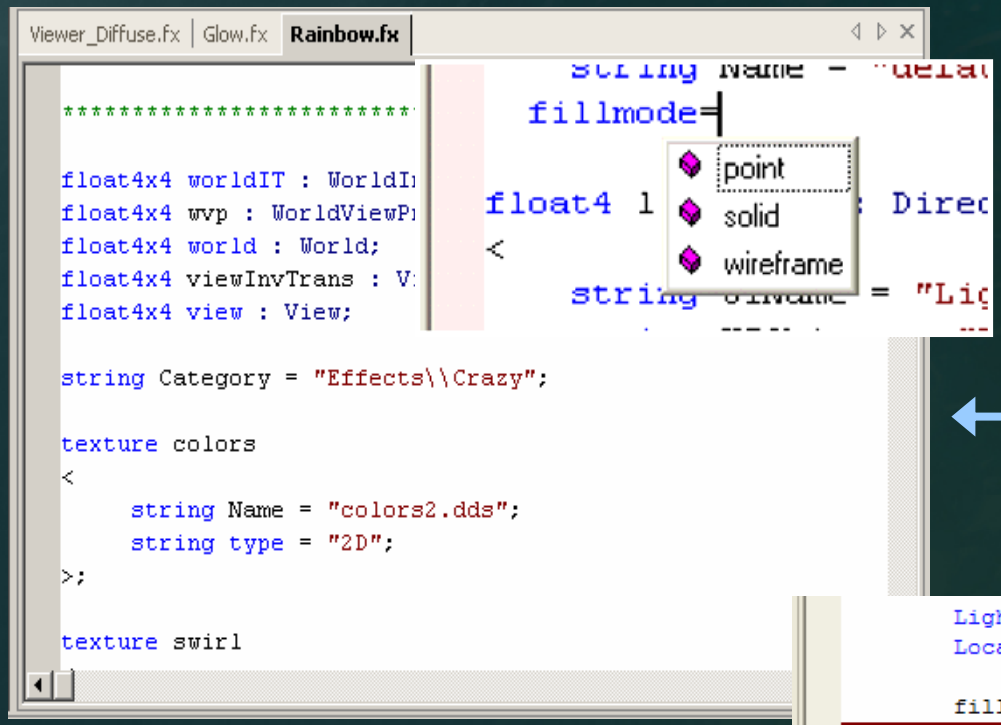
- 한 scene 안의 모든 FX 파일들을 동시에 미리보기
- Scene panel에서 위의 파일들을 scene의 적당한 부분에 적용



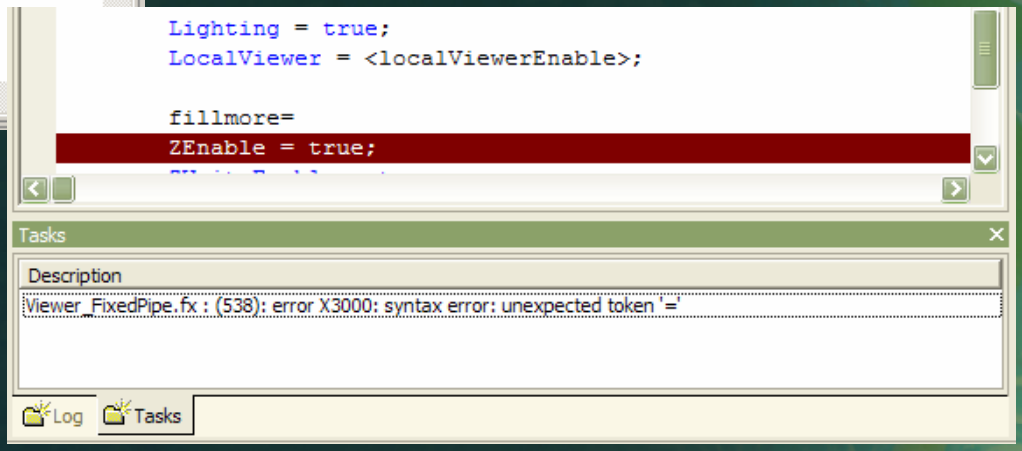


- source textures 보기
- render targets 미리보기
- 어떠한 texture라도 디스크에 저장!

편집과 디버깅

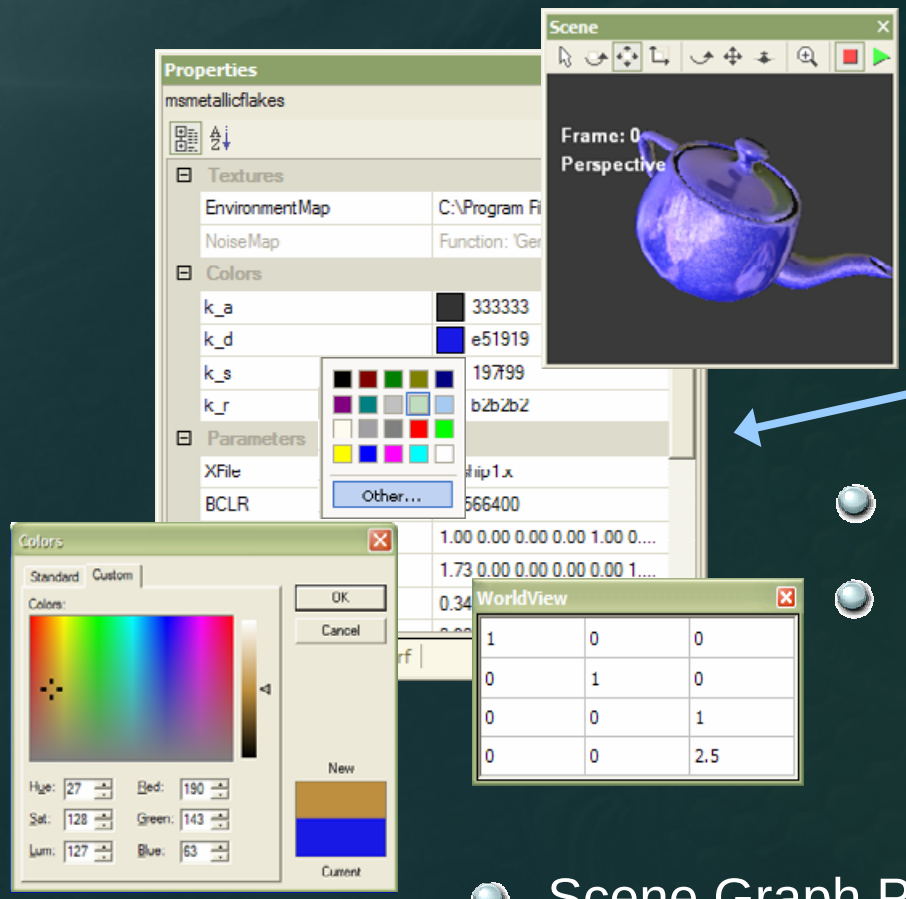


- .FX 파일 수정
- Intellisense (자동-완성)
- Syntax highlighting

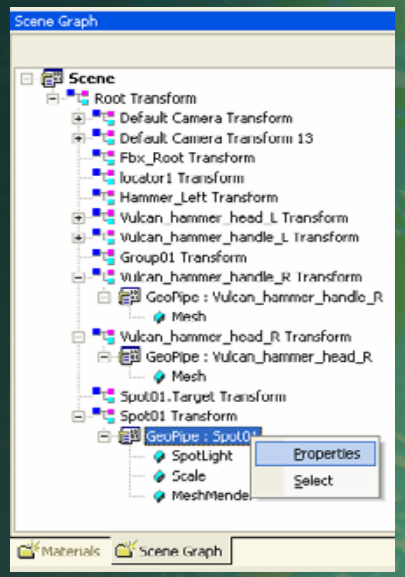


- 문제를 빨리 찾아 해결할 수 있도록 한다.

미리보기와 사용자 정의

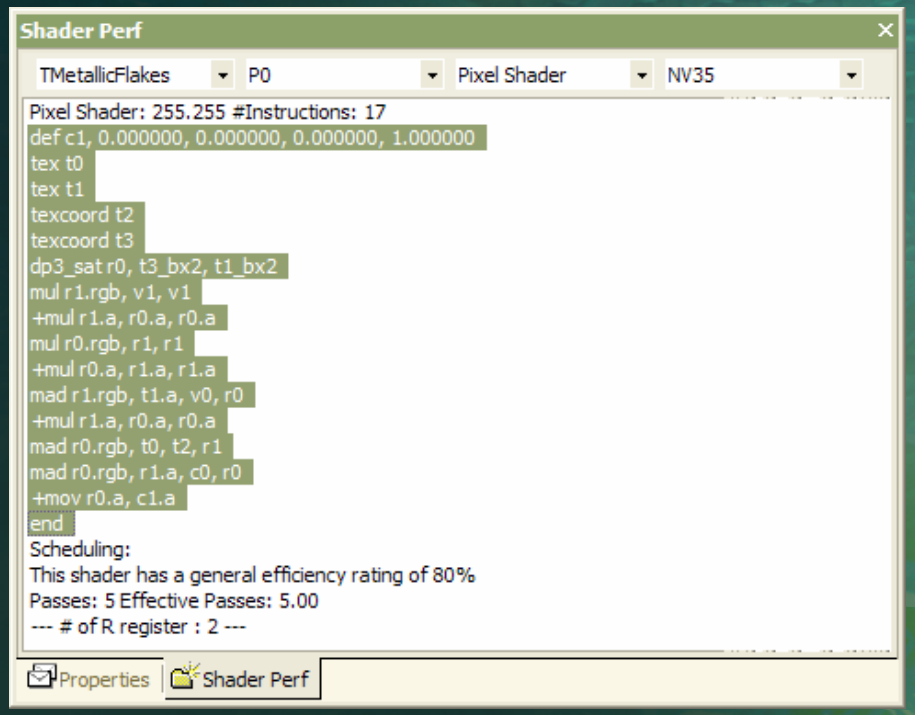


- 쉐이더의 변수들을 간편하게 편집
- semantics 과 annotations의 자동화된 parsing
- 빠르게 custom 색상 값을 선택
- 분리형 대화상자를 이용하여 벡터와 매트릭스 값을 업데이트
- Scene Graph Panel에서 scene 요소들을 선택하고 속성을 편집하기 위하여 탐색

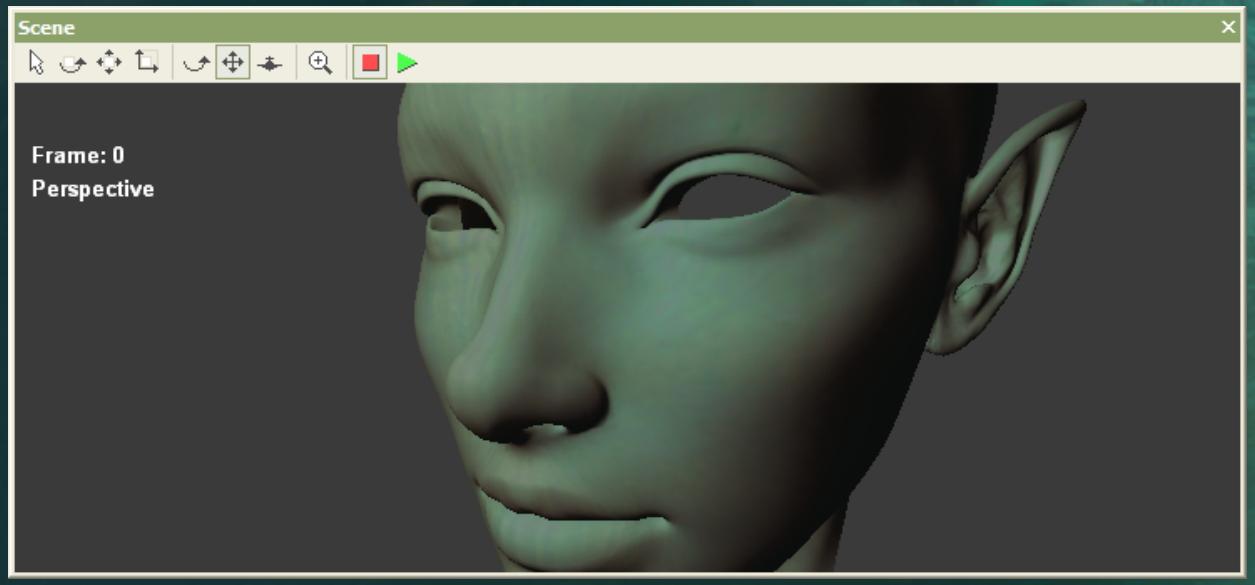


쉐이더 성능 조정

- 쉐이더 성능 패널 사용
- 분석할 기법, 경로, 버텍스, 픽셀 쉐이더를 선택
- 최신 NVIDIA GPU에서 픽셀 쉐이더 성능을 시뮬레이션
- 최적화된DirectX 어셈블리
- NVIDIA 성능분석
  - GPU 주기횟수
  - 효율성 / 활용도평가
  - 경로 수
  - 등록 사례



NVIDIA FX Composer



Scene 패널

- 3D 장면의 실시간 미리보기
  - 장면 요소에 자재 적용
  - 장면 요소 또는 전체 장면의 조작
  - 기본형 사용 또는 .x 모델 및 .nvb 장면 가져오기
  - 고유의 키 프레임 설정 또는 기존 애니메이션 재생
  - 광원 배치 및 조명 속성의 사용자 정의
  - 사용자 정의 카메라 또는 기본 장면 카메라 선택

FAQ

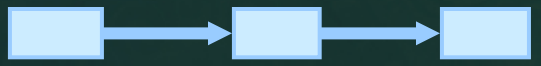
- Cg 또는 CLSL 지원여부
- 각 제작사의 엔진과의 통합성
- DCC 애플리케이션 통합성
- RenderMonkey, Effect Edit 및 기타와 비교
- 다른 플랫폼 지원여부

FX Composer 최신 버전과 전체 FAQ는 다음 주소를 참조하십시오.
http://developer.nvidia.com/fxcomposer

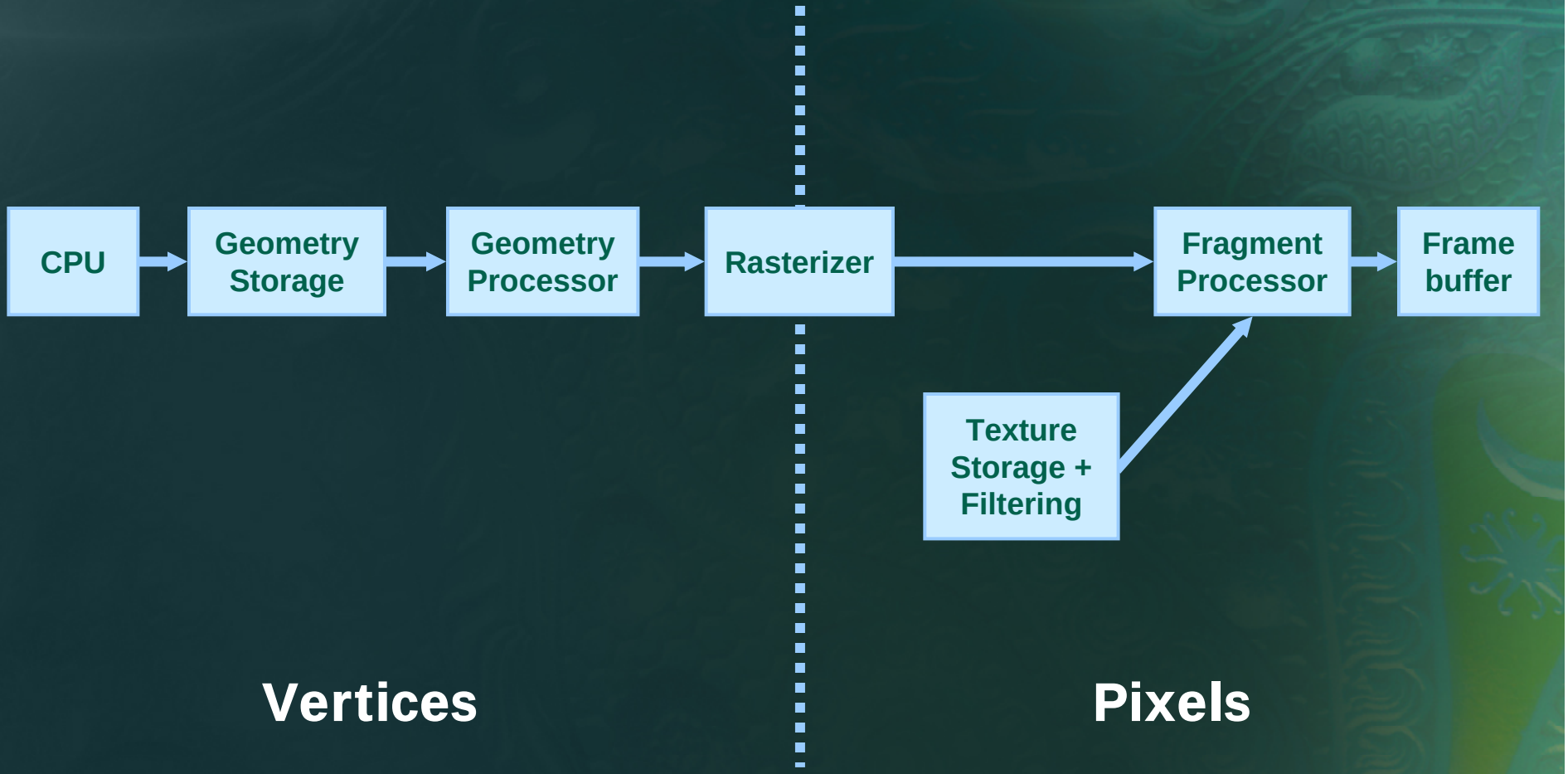
성능 조정

기본 원리

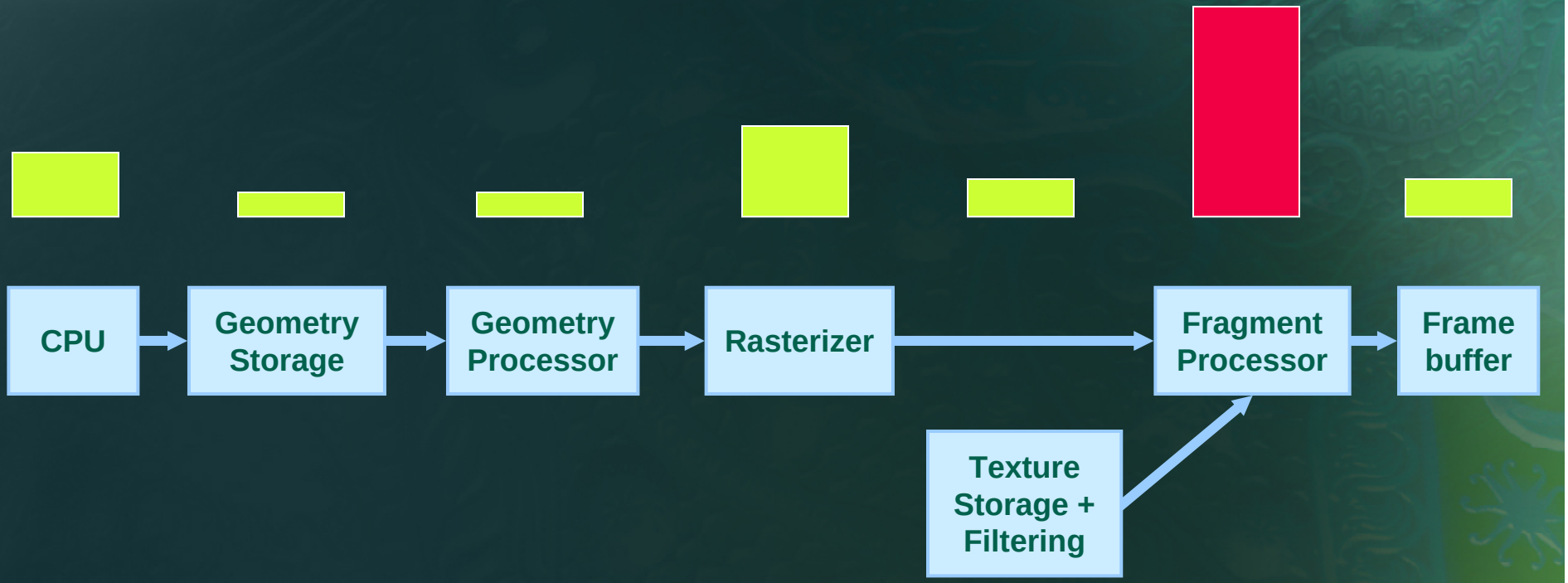
- 파이프라인 방식의 아키텍쳐
- 병목 현상의 파악과 제거
- 파이프라인 밸런싱



파이프라인 방식의 아키텍처

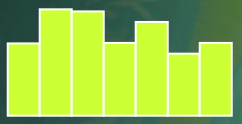


심각한 병목현상

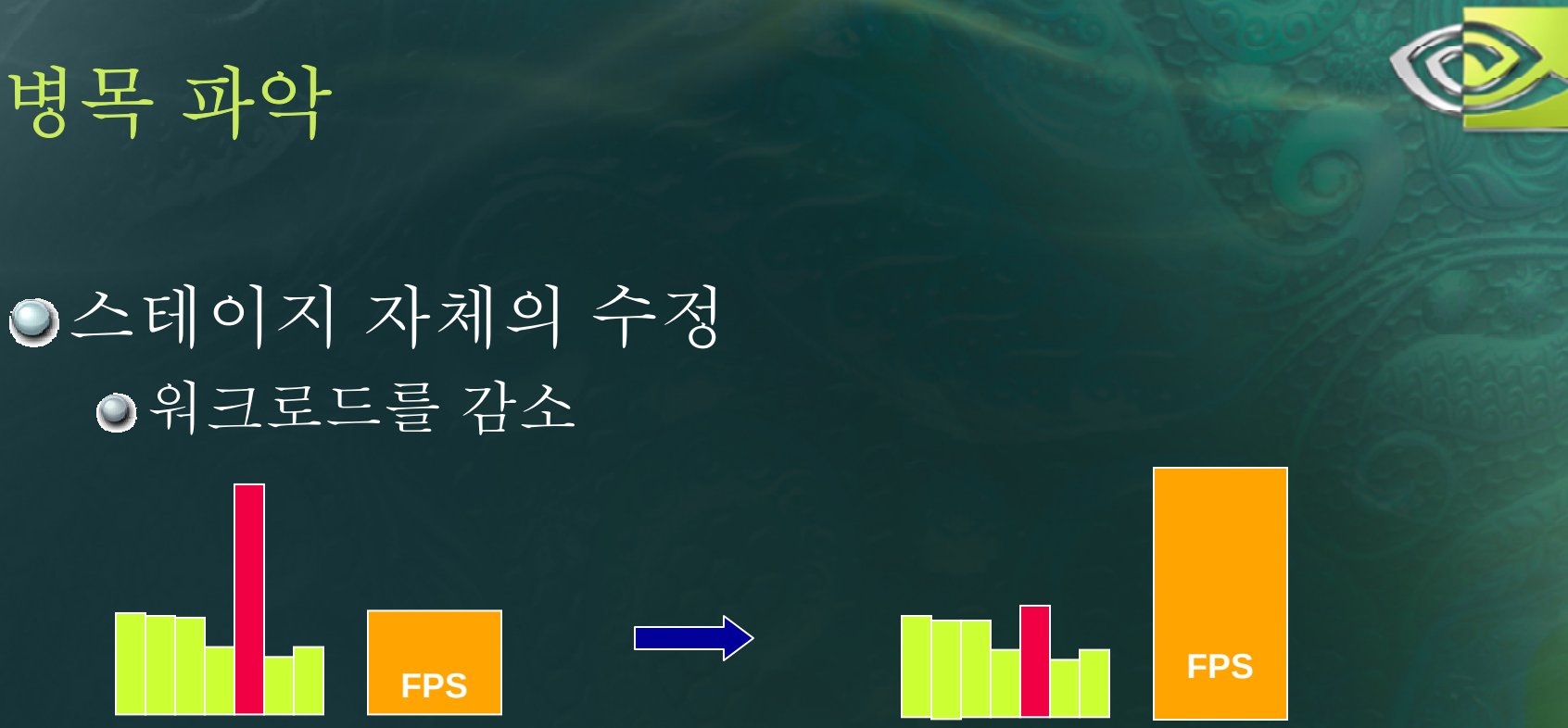


병목현상 파악

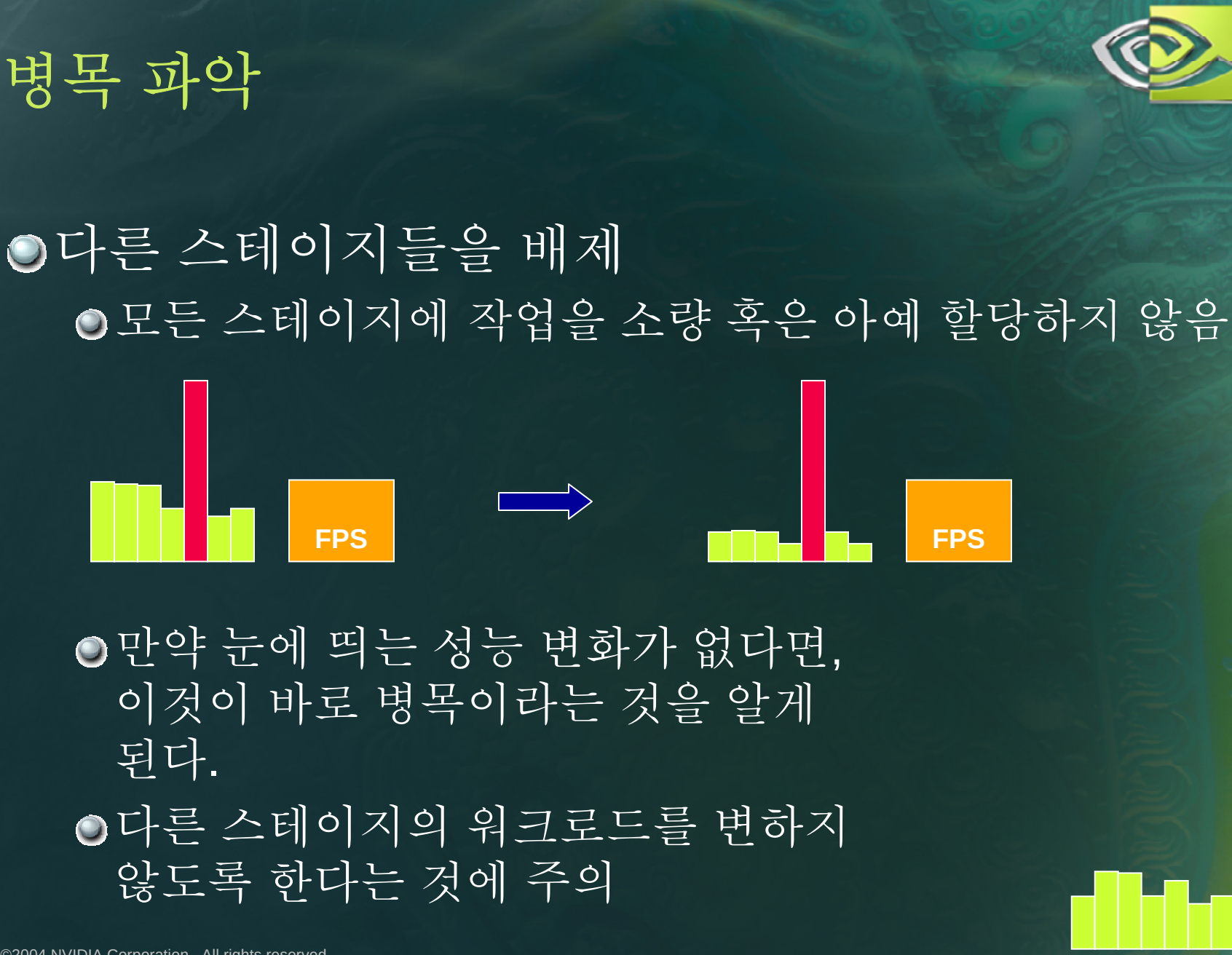
- 2가지 병목 파악 방법
- 스테이지 자체의 수정
- 기타 스테이지의 배제



병목 파악



병목 파악

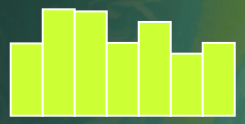


- 다른 스테이지들을 배제
  - 모든 스테이지에 작업을 소량 혹은 아예 할당하지 않음

  - 만약 눈에 띄는 성능 변화가 없다면, 이것이 바로 병목이라는 것을 알게 된다.
  - 다른 스테이지의 워크로드를 변하지 않도록 한다는 것에 주의

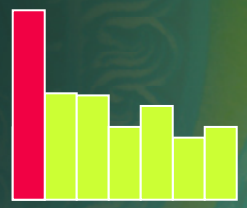
병목파악

- 대부분의 경우 한 스테이지 변경은 다른 스테이지에도 영향을 미침
- 필요한 테스트의 선택이 어려울 수 있음
- 몇몇 테스트를 검토해봅시다!



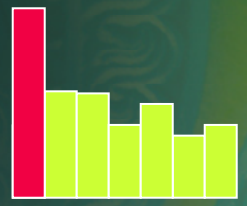
병목현상 파악: CPU

- 문제는?
- 게임 자체의 문제일 수 있다
  - 복잡한 물리엔진, 인공지능, 게임로직
  - 메모리 관리
  - 데이터 구조
- API를 올바르지 않게 사용하고 있을 수 있다
  - 오류와 경고를 위해 디버그 런타임 출력을 확인하십시오
- 디스플레이 드러이버 문제일 수 있다
  - Batch의 숫자가 너무 많을 수도 있음



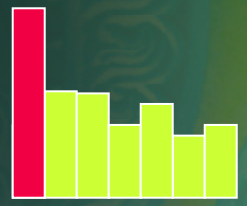
병목현상 파악: CPU

- CPU 워크로드를 줄여야 한다.
- 일시적으로 작동을 중지시킴
  - 게임 로직
  - 인공지능
  - 물리엔진
  - 렌더링 워크로드에 영향을 주지 않으면서 CPU를 많이 점유하는 그 어떤 것들



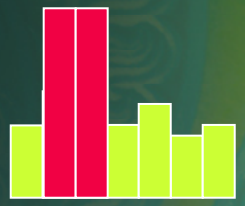
병목현상 파악: CPU

- 다른 스테이지를 배제하여야 한다
- DrawPrimitive 호출을 Kill 한다
  - 평소처럼 모든 것을 세팅한다. 그러나 무엇인가를 렌더할 때 단순히 DrawPrimitive* 호출을 하지 않는다
  - 문제: draw primitive 호출이 발생할 때 런타임 혹은 드라이버가 무슨 일을 하는 지를 알 수 없게 된다
- VTUNE 나 NVPerfHUD 사용



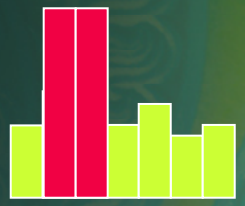
병목현상 파악: **Vertex**

- 문제는?
- vertices와 indices를 그래픽 카드로 전송
- vertices와 indices를 triangles를 변환
- Vertex cache misses
- 코스트가 비싼 vertex shader의 사용



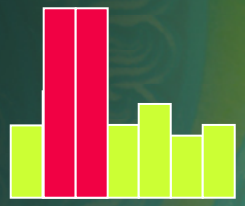
병목현상 파악: Vertex

- Vertex 오버헤드 줄이기
- 간단한 vertex shader를 사용한다
- 더 작은 수의 Triangles를 전송한다??
  - 좋지 않다
- AGP Aperture를 감소한다??
  - 아마 안 좋을 것
  - 비디오 메모리를 확인하기 위하여 NVPerfHUD를 사용한다
  - 가득 차 있다면 AGP에 텍스쳐가 있을 수 있다



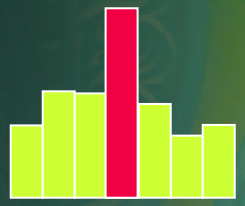
병목현상 파악: Vertex

- 다른 스테이지들을 배제한다
- 더 작은 백버퍼에 렌더해 본다; 이는 다음 항목들을 배제하게 된다
  - 텍스쳐, 프레임 버퍼, 픽셀 쉐이더
- CPU 병목을 검사한다
- 작은 백버퍼대신 작은 뷰 포트에 렌더해 볼 수도 있다. 이 역시 다음 항목들을 배제하게 된다
  - 텍스쳐, 프레임 버퍼, 픽셀 쉐이더



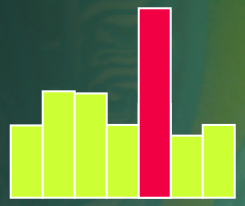
병목현상 파악: **Raster**

- 병목을 일으키는 경우가 희박함. 다른 스테이지들을 먼저 검사해 본다



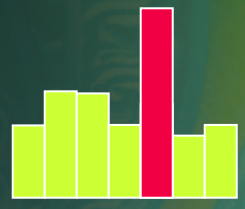
병목현상 파악: **Texture**

- 문제는?
- Texture cache misses
- 큰 용량의 Textures
- Bandwidth
- AGP에서 Texturing



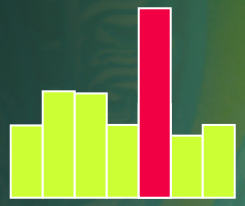
병목현상 파악: **Texture**

- Texture bandwidth를 감소한다
- 아주 작은 (2x2) 텍스쳐들을 사용한다
  - 좋은 방법이긴 하나 텍스쳐 알파로 알파 테스트를 하고 있다면 이는 증가하는 fill 때문에 실제로 실행이 더 느려질 수 있다. 그래도 좋은 테스트 방법이다
- 아직 사용하지 않고 있다면 mipmap을 사용한다
- anisotropic filtering이 켜져 있다면 끈다



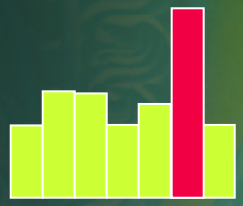
병목현상 파악: Texture

- 다른 단계들을 배제한다
- texture는 쉽게 직접 검사가 가능함으로, 그 방식을 사용할 것을 권장함



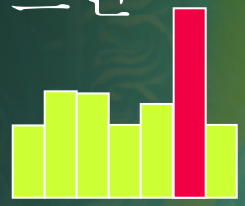
병목현상 파악: **Fragment**

- 문제는?
- 코스트가 비싼 픽셀 쉐이더
- 필요이상의 fragments를 렌더링 함
  - 높은 depth complexity
  - z-cull 좋지 않음



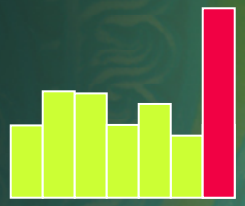
병목현상 파악: **Fragment**

- 스테이지 자체를 수정한다
- solid color만을 출력해 본다
  - 좋은 점: 각 fragment 당 작업을 하지 않는다
  - 하지만 또한 텍스쳐에 영향을 끼치기 때문에 텍스쳐를 배제하여야만 한다
- 단순화된 수학을 사용하라
  - 좋은 점: 각 fragment 당 더 적은 작업을 한다
  - 하지만 단순화된 수학이 동일한 방식으로 텍스쳐에 인덱싱(index)을 하도록 하여야 한다 그렇지 않으면 텍스쳐 스테이지도 변화하게 되는 것이다



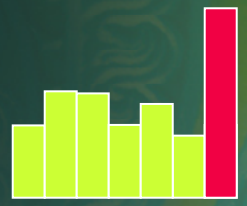
병목현상 파악: FB

- 문제점?
- 필요 이상으로 버퍼를 건드리게 된다
  - Multiple passes
- 수 많은 알파 블렌딩(alpha blending)
- 너무 큰 버퍼를 사용하게 되면
  - 필요 없을 때 스텐실 사용
  - 대부분의 경우 dynamic reflection cube-maps 는 x8r8g8b8 색상 대신 r5g6b5 색상으로 대치될 수 있다



병목현상 파악: FB

- 스테이지 자체를 수정한다
- 24 비트 depth 버퍼 대신 16비트 depth 버퍼를 사용한다
- 32 비트 depth 버퍼 대신 16비트 depth 버퍼를 사용한다



연습

연습: Clean the Machine

- 올바른 드라이버 사용
- 게임 빌드의 릴리즈 (지속적 최적화)
- 경고나 오류에 대한 디버깅 결과 확인
- 릴리즈 D3D 런타임 사용
- 최대한의 검증이란 없음
- 비등방성(anisotropic) 필터링 또는 앤티 앨레어싱에 우선하는 드라이버는 없음
- V-sync 해제

연습: 예제 1

연습: 예제 1

- 동적 버텍스 버퍼
  - 나쁜 생성(creation) 플래그

```
HRESULT hr = pd3dDevice->CreateVertexBuffer(
                    6* sizeof( PARTICLE_VERT ),
                    0,      //이것을 static으로 선언
                    PARTICLE_VERT::FVF,
                    D3DPOOL_DEFAULT,
                    &m_pVB,
                    NULL );
```

연습: 예제1

- 동적 버텍스 버퍼
  - 좋은 생성(creation) 플래그

```
HRESULT hr = pd3dDevice->CreateVertexBuffer(
                    6* sizeof( PARTICLE_VERT ),
                    D3DUSAGE_DYNAMIC |
                    D3DUSAGE_WRITEONLY,
                    PARTICLE_VERT::FVF,
                    D3DPOOL_DEFAULT,
                    &m_pVB,
                    NULL );
```

연습; 예제 1

- 동적 버텍스 버퍼
  - 나쁜 잠금(Lock) 플래그

```
m_pVB->Lock(0, 0,(void**)&quadTris, 0);
```

- 플래그가 하나도 없다니!?
  - 정말 안 좋은 걸....

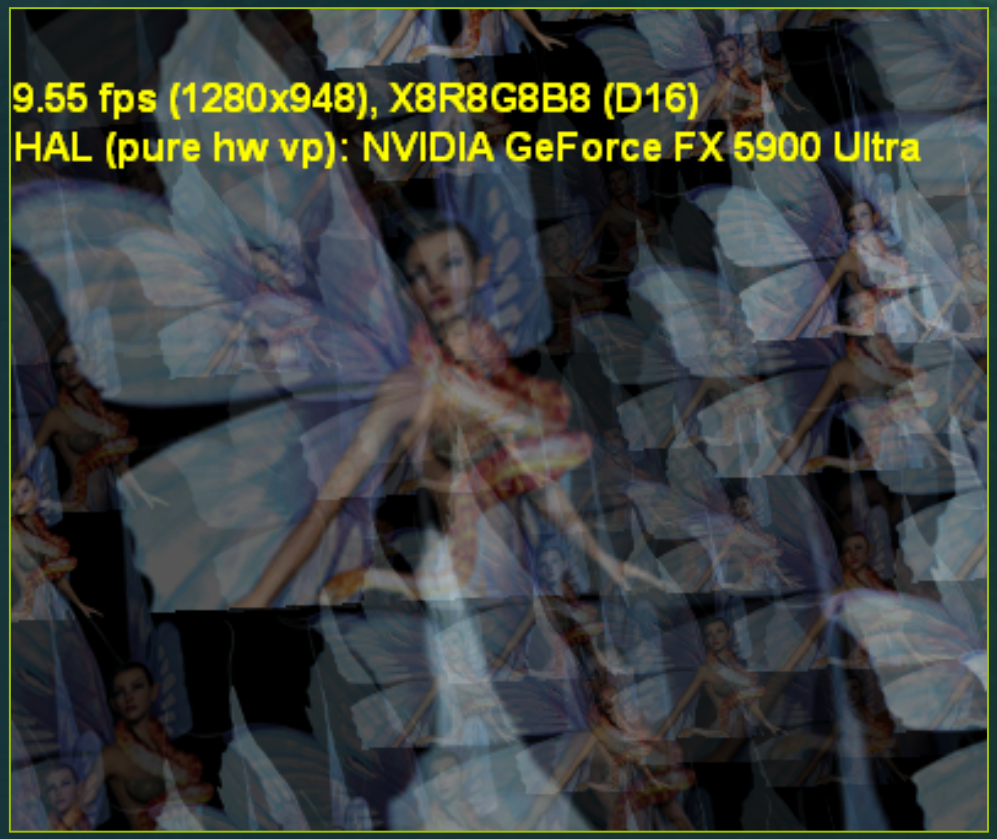
연습: 예제 1

- 동적 버텍스 버퍼
  - GOOD Lock flags

```
m_pVB->Lock(0, 0,(void**)&quadTris,
D3DLOCK_NOSYSLOCK | D3DLOCK_DISCARD);
```

- 각 프레임에서 버텍스 버퍼를 처음으로 lock 할 때 D3DLOCK_DISCARD 를 사용한다
  - 그리고 그 버퍼가 가득 찰 때 다시 사용한다
  - 그 외의 경우 단순히 NOSYSLOCK을 사용

연습: 예제 2



연습: 예제 2

- 텍스쳐 대역폭 과도한 사용(overkill)
- 밉맵 사용
- 가능하면 dxt1 사용
  - 캐시에 압축 데이터를 저장하는 카드도 있음
- 무방한 경우 더 작은 텍스쳐 사용
  - 풀잎 하나에 정말로 1024x1024의 텍스쳐가 필요한가? --- 그럴 수도 있지만.

연습: 예제 3

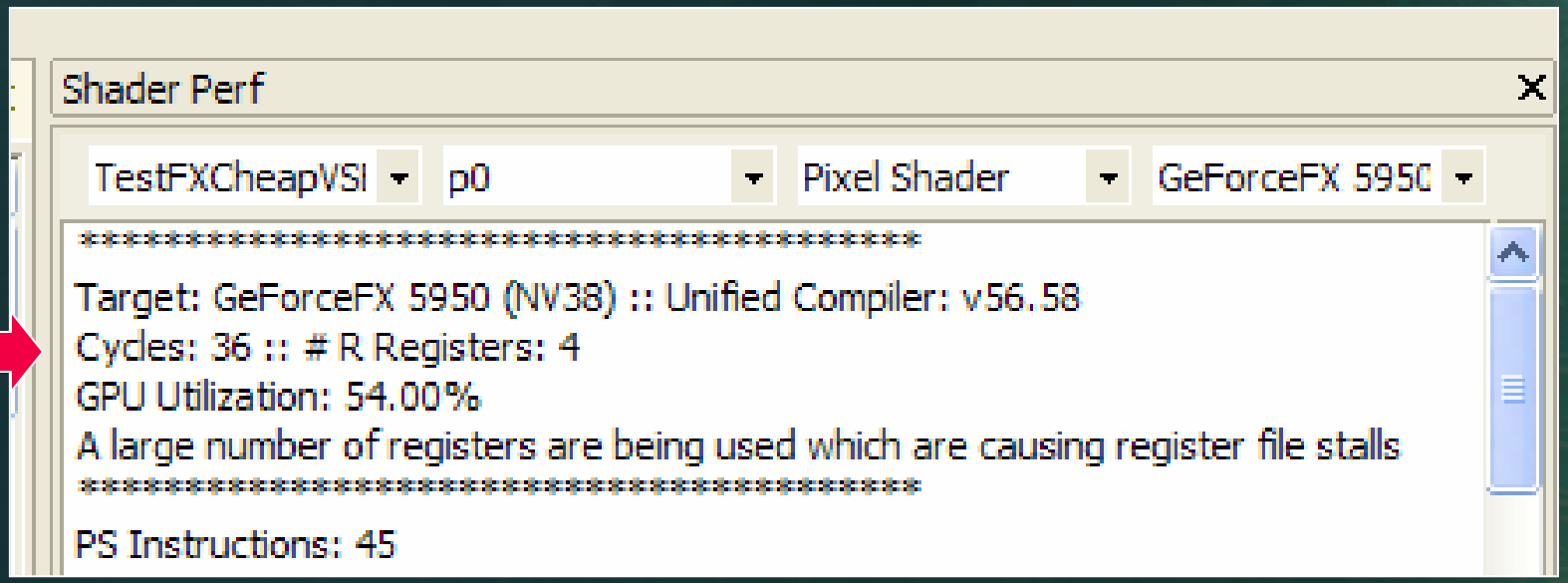
연습: 예제 3

- 코스트가 비싼 픽셀 쉐이더
- 막대한 성능 효과
- 버텍스는 3개 뿐이지만 백 만 픽셀일 수도 있다.
  - 1024x1024에 불과

전체 픽셀을 보라!!

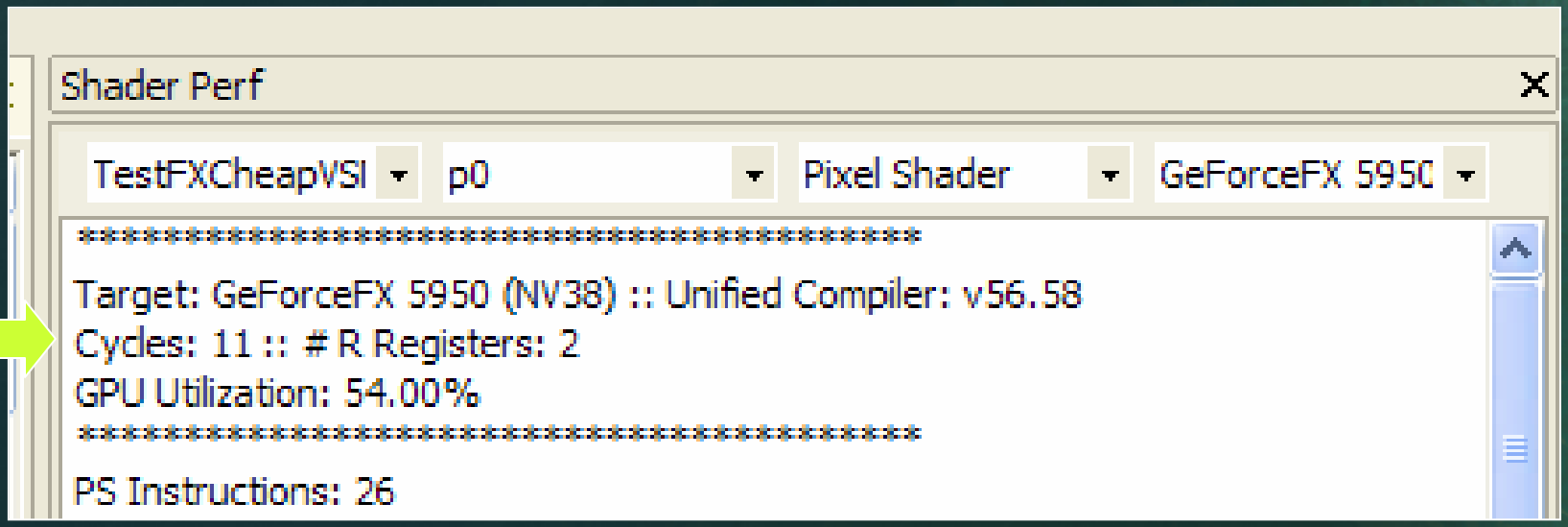
연습: 예제 3

- 36 사이클 -> 나쁜 예



연습: 예제 3

- 11 사이클 -> 좋은 예



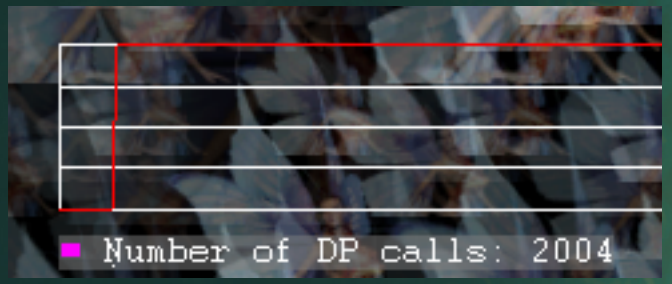
연습: 예제 3

- 무엇이 바뀌었나?
- 삼각형들간의 고정된 수학을 버텍스 쉐이더로 이동
- ‘float’대신에 ‘half’ 을 사용
- 불필요한 경우에 노멀라이즈(normalize) 제거
  - 노멀라이제이션 유리스틱스(Normalization Heuristics) 참조
  - http://developer.nvidia.com

연습: 예제 4

연습: 예제 4

- 배치가 너무 많음
- 과거에는 모든 쿼드를 자체 배치로 전송
- 대신, 쿼드를 하나의 대형 VB로 그룹화한 후 1회 호출로 전송

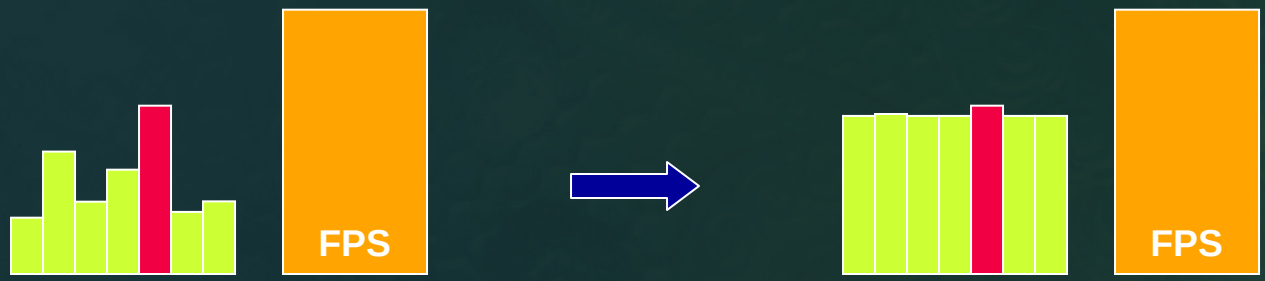


연습: 예제 4

- 만약 다른 텍스쳐를 사용하려면?
- 텍스쳐 아틀라스 (texture atlases)사용
- 텍스쳐 두 개를 단일 텍스쳐로 통합하고 버텍스와 픽셀 쉐이더를 사용하여 텍스쳐 좌표를 옵셋

파이프라인 밸런싱

- 병목 없는 스테이지를 더 많이 활용하여 파이프라인을 밸런싱
- 너무 많이 사용하지 않도록 주의



요약

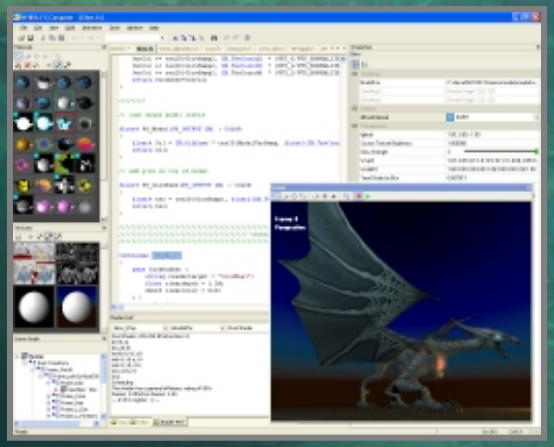
- NVIDIA는 다양한 성능 분석 도구를 제공한다
- 파이프라인 아키텍처는 병목에 의해 좌우된다.
- 빠른 테스트들을 사용하여 병목 파악
- NVPerfHUD 를 사용하여 파이프라인 분석
- FX Composer 로 쉐이더 조정

추가정보

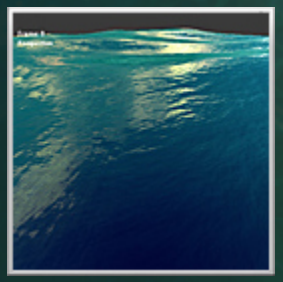
- 프리젠테이션, SDK, 툴 등 다운로드:
  http://developer.nvidia.com
- 툴 및 SDK에 대한 질문, 요청, 의견:
  sdkfeedback@nvidia.com
- 이 프리젠테이션에 대한 문의:
  kashida@nvidia.com

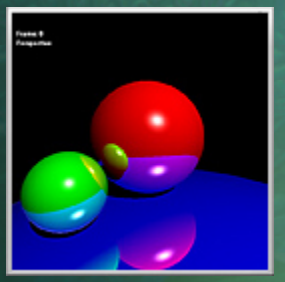
developer.nvidia.com
GPU Programming 소스

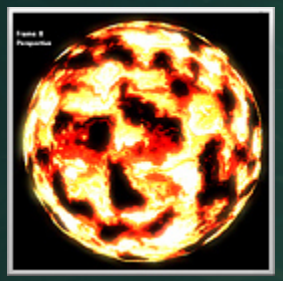
- 최신설명자료
- SDK
- 최첨단 툴
  - 성능 분석 도구
  - 컨텐츠 작성 도구
- 수 백 가지 효과
- 비디오 프리젠테이션 및 자습서
- 라이브러리와 유틸리티
- 뉴스 및 뉴스레터 아카이브

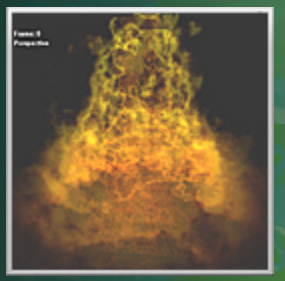


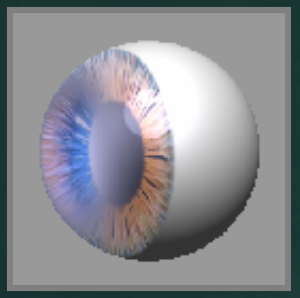
EverQuest® content courtesy Sony Online Entertainment Inc.











NVIDIA SDK

실시간 개발자들을 위한 소스

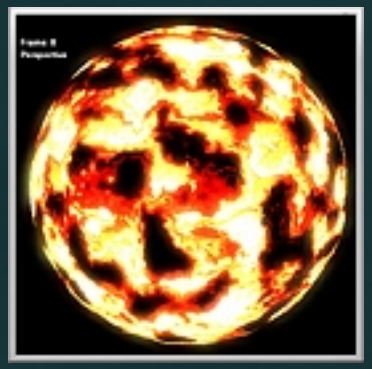
최신 그래픽 기술의 장점을 도입하도록 도움을 주는 수 백개의 샘플 코드와 이펙트.

- 업데이트되었거나 완전히 새로운 **DirectX** 와 **OpenGL** 의 수 많은 샘플 코드들과 백서들:

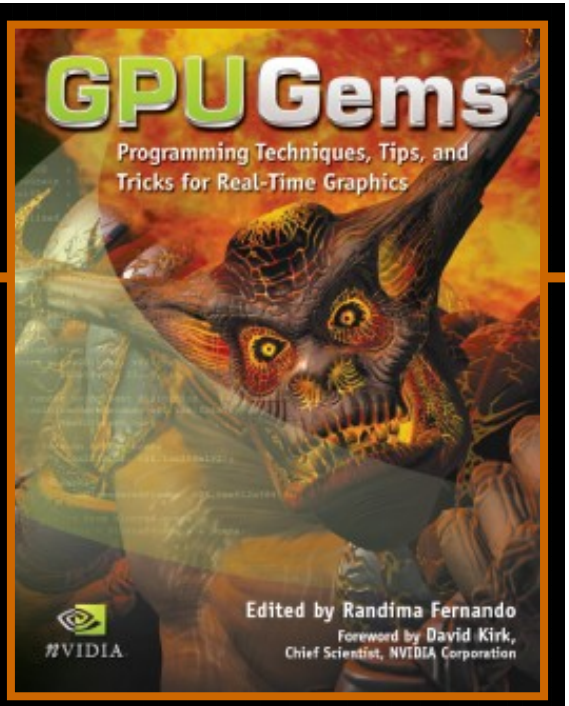
  Geometry Instancing, Rainbow Fogbow, Blood Shader, Perspective Shadow Maps, Texture Atlas Utility, ...

- **Custom geometry,** 애니메이션, 및 기타등등과 함께 제공되는 수 많은 효과들:

  Skin, Plastics, Flame/Fire, Glow, Gooch, Image Filters, HLSL Debugging Techniques, Texture BRDFs, Texture Displacements, Tonemapping, 및 심지어는 간단한 **Ray Tracer**까지



GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics



- 일류기업과 대학의 전문가들이 내놓은 실용적인 실시간 그래픽 기법
- 엄청난 가치:
  - 업계 전문가들이 기여
  - 풀 컬러 (300+ 도표 및 스크린 샷)
  - 하드 커버
  - 816 페이지

더 많은 정보는 다음 사이트에서:
http://developer.nvidia.com/GPUGems

"***GPU Gems*** 은 진보된 그래픽 기술의 훌륭한 도구상자입니다. 초보 프로그래머들과 그래픽 도사들도 이 책이 실용적이고 흥미있고 유용하다는 것을 느낄 것입니다."

팀 스위니(Tim Sweeney)

에픽 게임스社의 *언리얼*의 메인 프로그래머

"이 논문 컬렉션은 그 깊이와 범위에서 특히 인상적입니다. 이 책은 제품위주의 사례연구들과 기존에 출판되지 않았던 최고의 연구사례, 이해하기 쉬운 학습과정, 및 대규모의 샘플코드들과 데모등을 전반에 걸쳐 포함하고 있습니다."

Eric Haines

*Real-Time Rendering*의 저자

The Cg Toolkit

- NVIDIA Cg Compiler
  - Vertex (DirectX 9, OpenGL 1.4)
  - Pixel (DirectX 9)
- Cg Standard Library
- Cg Runtime Libraries for DirectX and OpenGL
- NVIDIA Cg Browser
- Cg Language Specification
- Cg User's Manual
- Cg Shaders
  (기존의 작성된 프로그램들과 함께 나뉘어 분류되어 있음)
- The Cg Tutorial
  (developer.nvidia.com/CgTutorial)