実践パフォーマンス解析

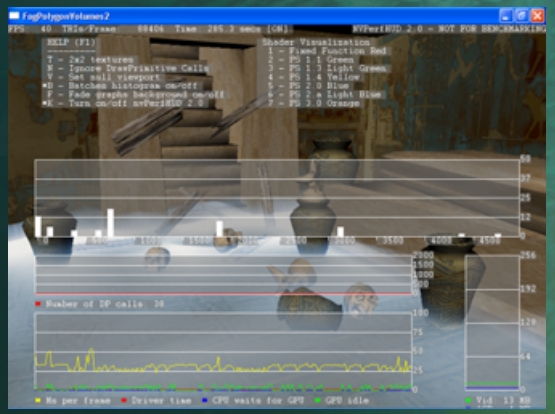
芦田幸治
NVIDIA
デベロッパー・テクノロジー

概要

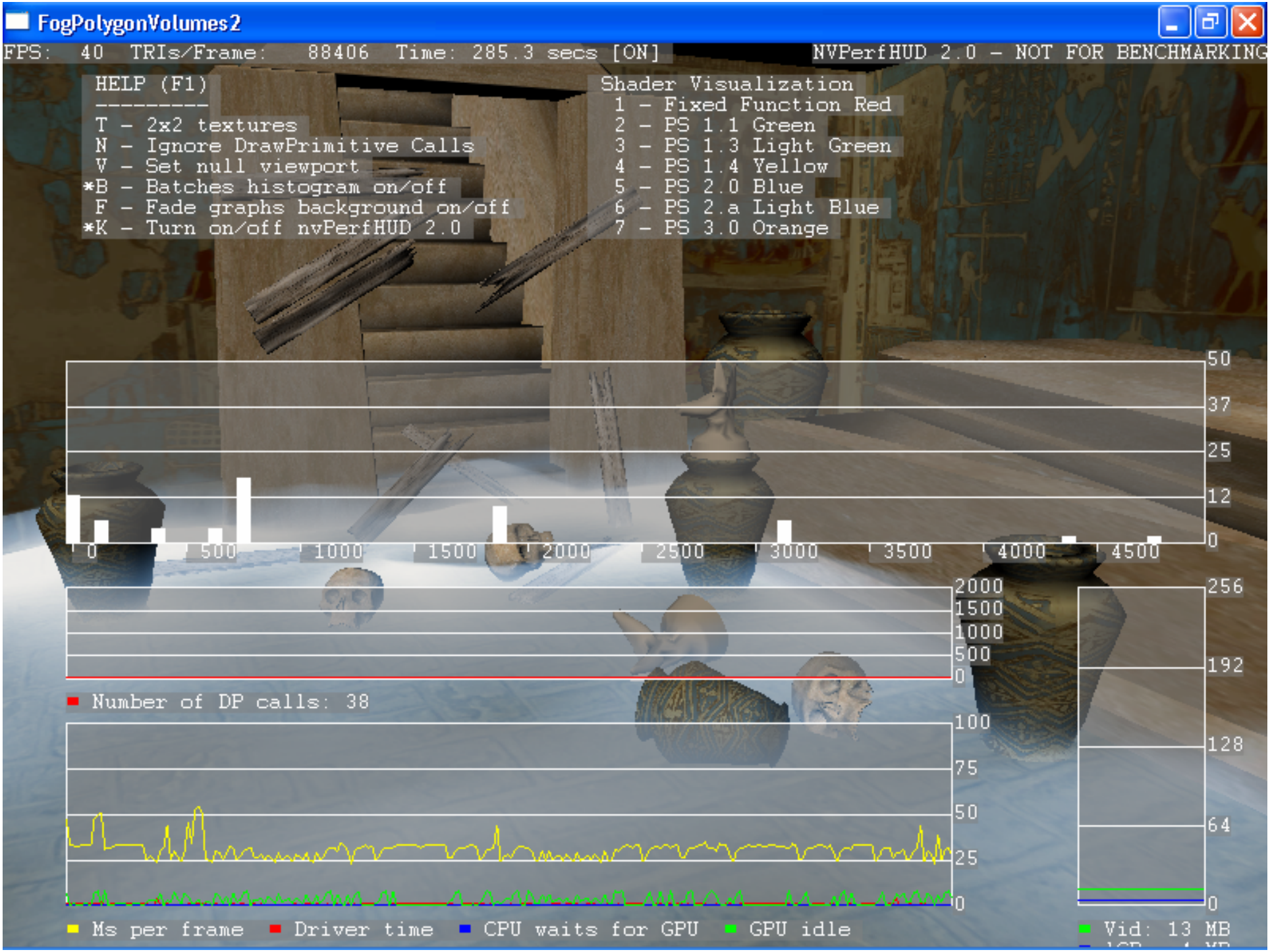
- 解析ツール
- パイプラインでのボトルネック発見
- 問題発見のデモ

解析ツール

NVPerfHUD

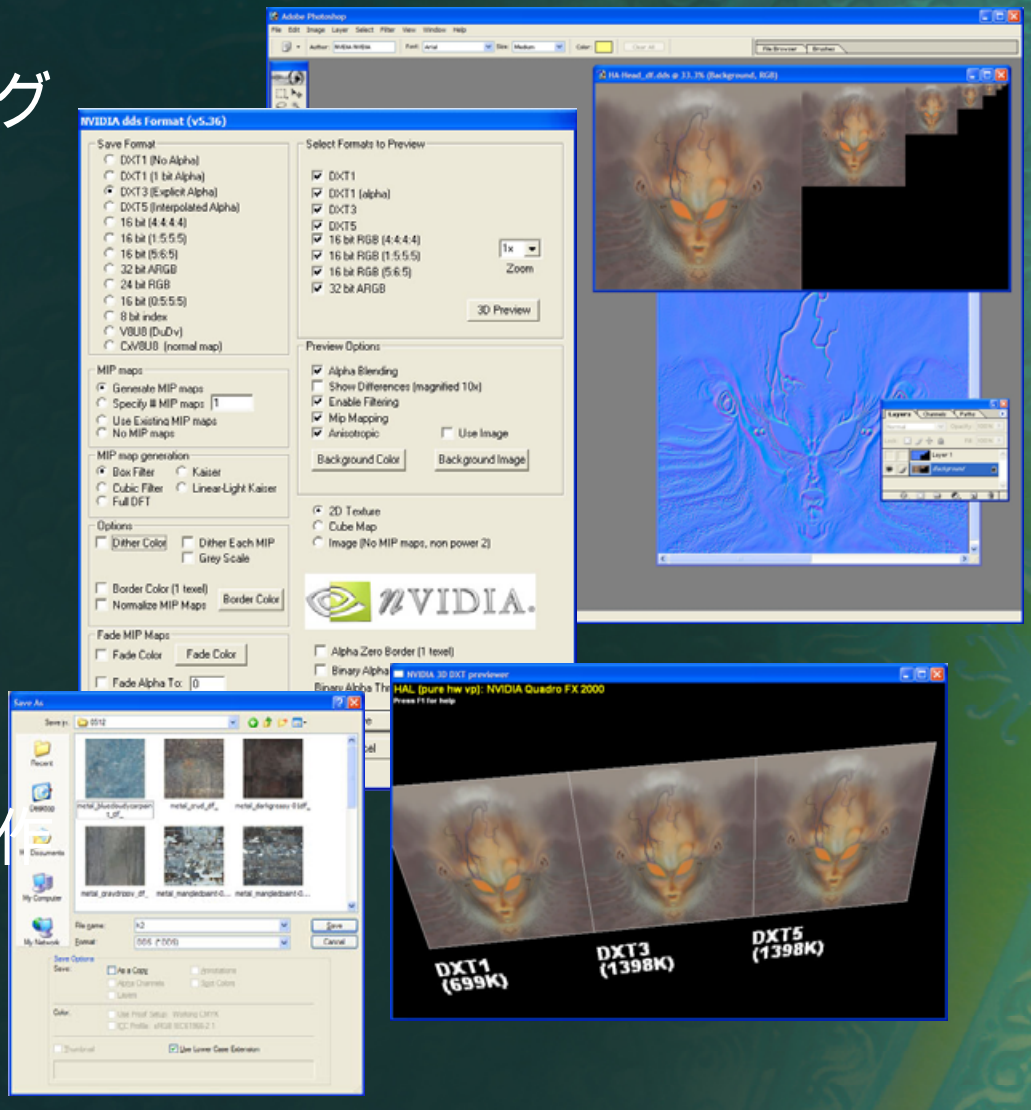


- 重要な各種統計データのグラフ
- 出力される情報:
  - GPU待ち時間
  - Driver待ち時間
  - Driverでの実行時間
  - 1フレームにかかる時間
  - AGPとビデオメモリの使用量
  - フレームごとのDIPやDPコールの回数と、そのヒストグラム
- ドライバーに依存せず、どんなDirect3D9のアプリケーションでも使用可
- 以前は登録デベロッパーのみに提供していた


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

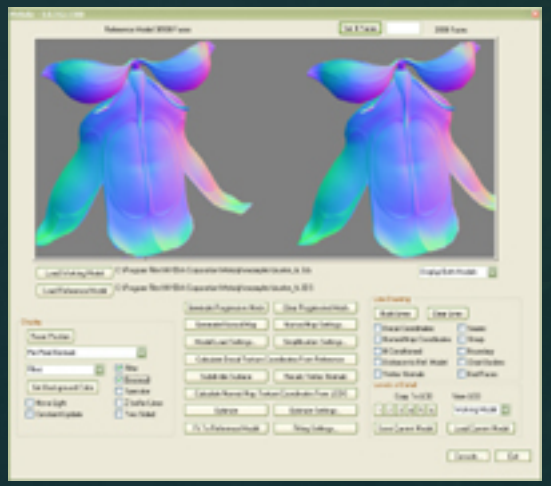
テクスチャのためのツールとプラグイン

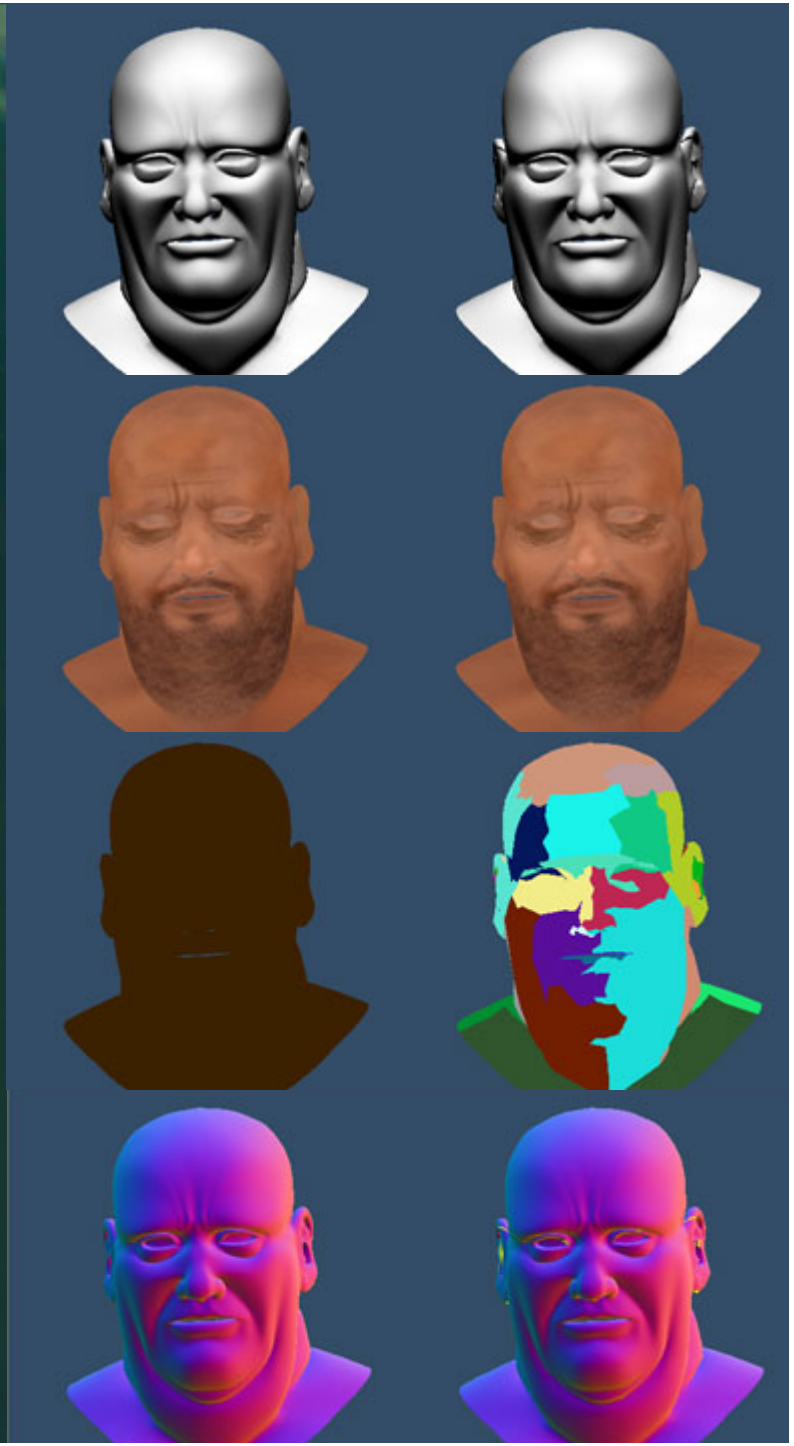
- フォトショップのプラグイン:
  - DXT圧縮(.dds)
  - 法線マップ作成
  - 3Dでの閲覧と差分表示
  - ミップマップ作成
- コマンドからの利用と.lib
- DDSのビューワ
- テクスチャアトラスの作成と閲覧

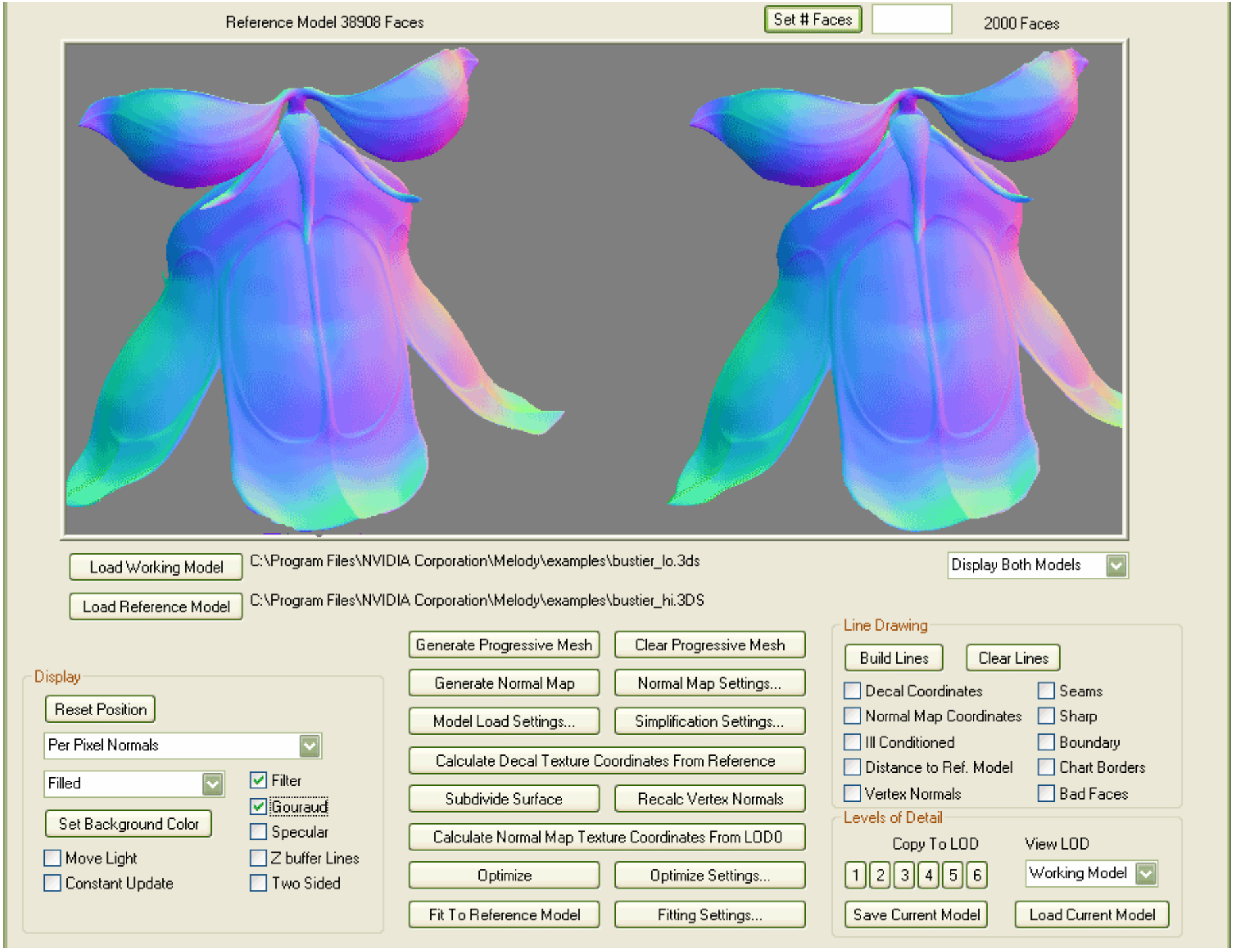


Melody

- 高解像度メッシュを入力 (~160万ポリゴン)
- 高度なプログレッシブ・メッシュによりLODを作成
- メッシュの最適化と簡略化
- チャートによるUV座標作成
- レイキャストによる法線マップの作成






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ライブラリ

- .EXR画像フォーマットをサポート
- ILMにより開発されたHDR画像フォーマット
  - www.openexr.org
- 色毎に16ビット浮動小数点精度
- .NET 2003で使用可能なライブラリ

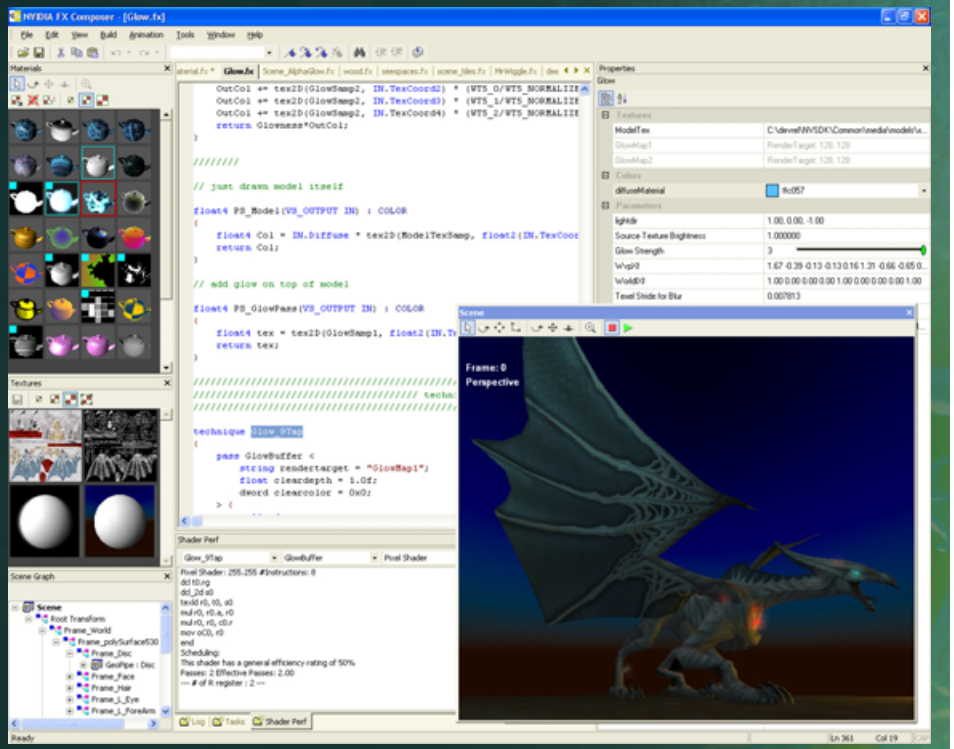
その他のツールやライブラリ

- NVShaderPerf
  - FX ComposernのShader Perfパネルと同じ技術
  - HLSL、!!FP1.0、!!ARBfp1.0、PS1.x、PS2.x で書かれたDirectXならびにOpenGLのシェーダーをサポート
  - NVIDIAのGPU全てに対してのパフォーマンス解析を表示
- NVMeshMender
  - 問題のあるメッシュを修復
  - ピクセル毎のライティングに対してメッシュを準備する
- NVTriStrip
  - キャッシュにやさしい三角形ストリップ作成
  - ストリップまたはリストの形で出力

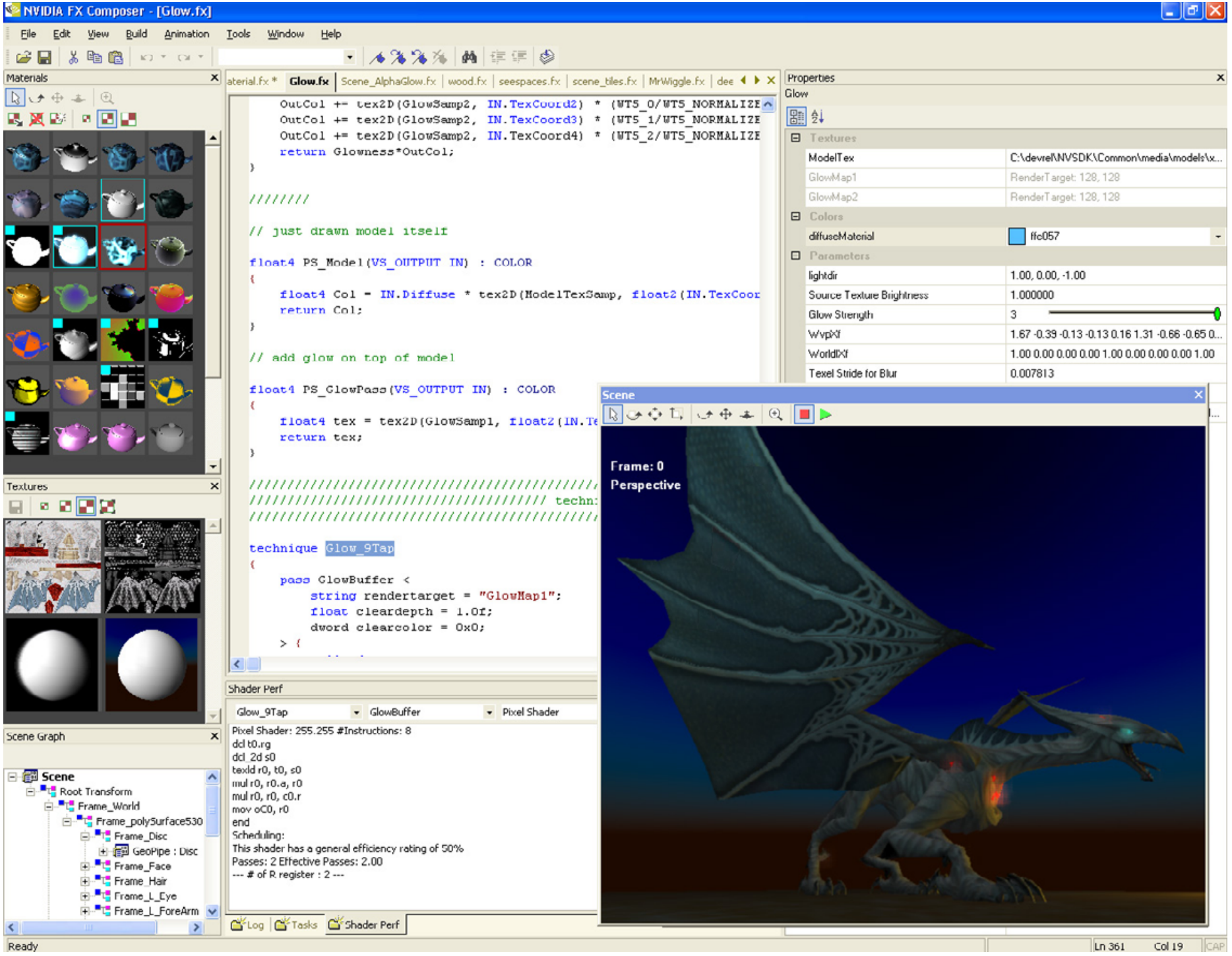
NVIDIA FX Composer

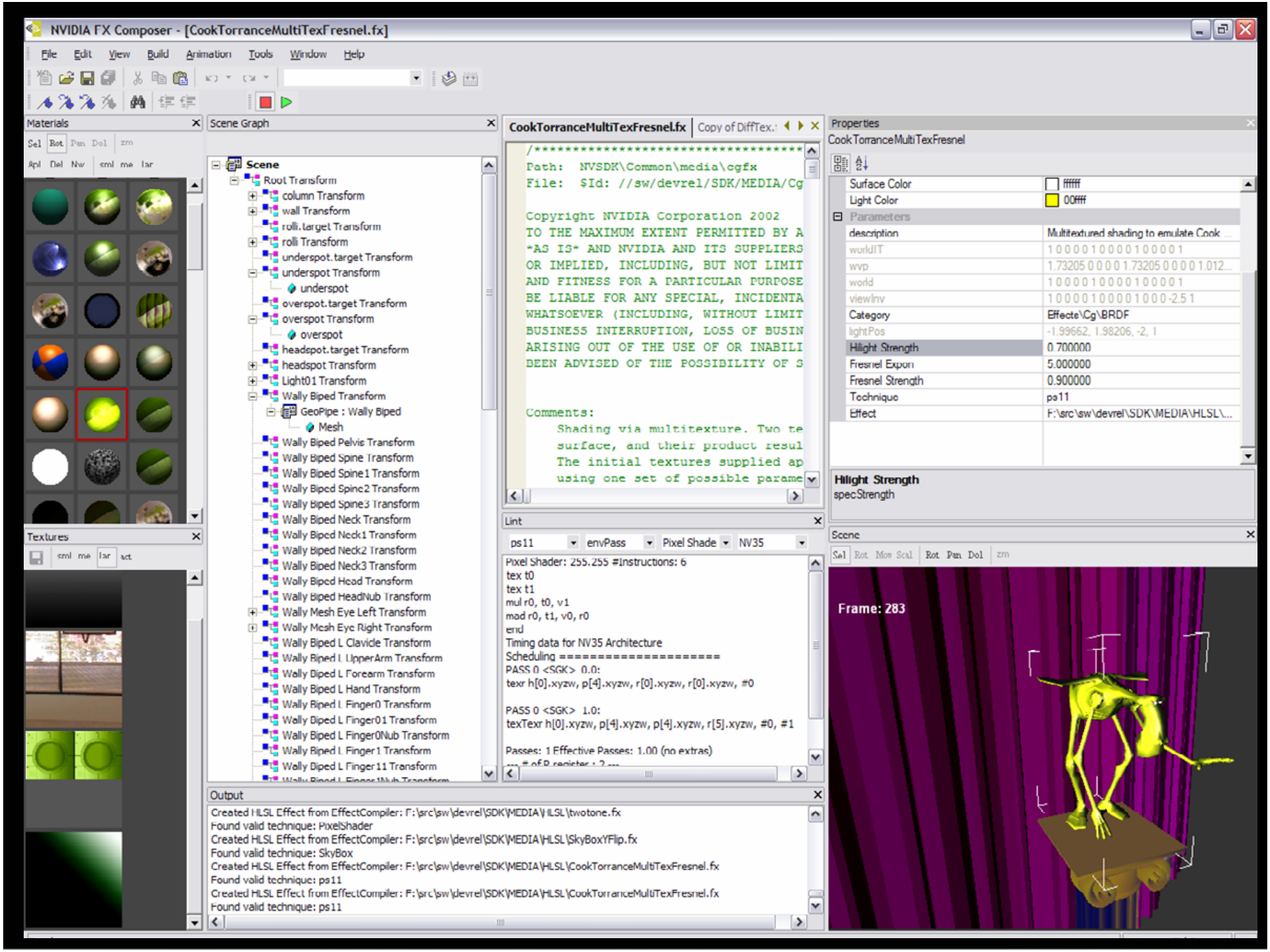
NVIDIAのみが供給しうる最適化機能とリアルタイムでのプレビューを可能にしたシェーダー開発環境により、開発者は高性能シェーダの開発が可能になる

- 高機能な開発環境でのシェーダー作成
- シェーダーデバッグ機能を用いたデバッグ
- 高機能解析、最適化機能を用いた最適化



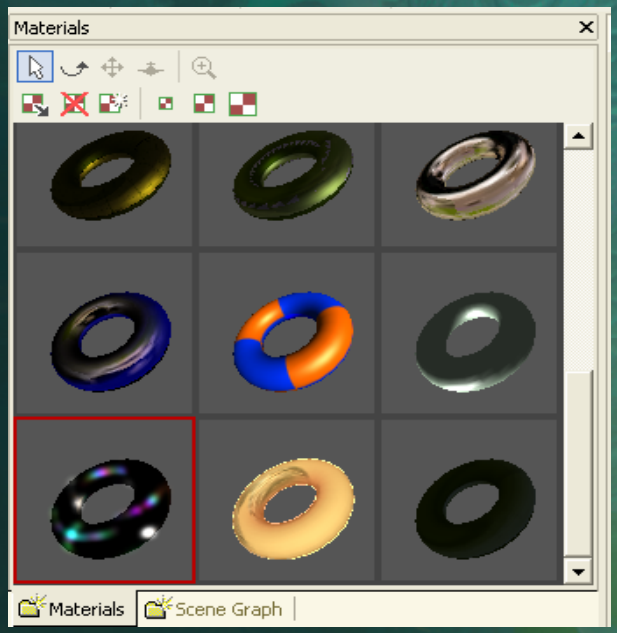
EverQuest® content courtesy Sony Online Entertainment Inc.


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
aterial.fx * Glow.fx Scene_AlphaGlow.fx wood.fx seespaces.fx scene_tiles.fx MrWiggle.fx dee
OutCol += tex2D(GlowSamp2, IN.TexCoord2) * (WT5_0/WT5_NORMALIZE
OutCol += tex2D(GlowSamp2, IN.TexCoord3) * (WT5_1/WT5_NORMALIZE
OutCol += tex2D(GlowSamp2, IN.TexCoord4) * (WT5_2/WT5_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;
}
//
////////////////////////////////////// techn
//
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

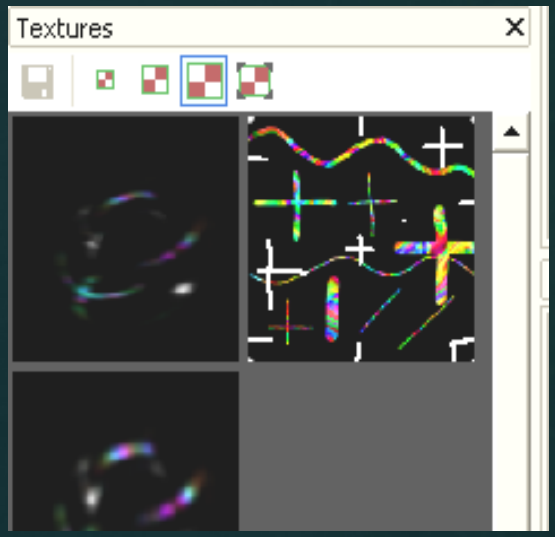

NVIDIA FX Composer - [CookTorranceMultiTexFresnel.fx]
File Edit View Build Animation Tools Window Help
Materials
Scene Graph
Scene
Root Transform
column Transform
wall Transform
rolli.target Transform
rolli Transform
underspot.target Transform
underspot Transform
underspot
overspot.target Transform
overspot 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

マテリアルとテクスチャ

- シーンで使用する全てのFXファイルを閲覧可能

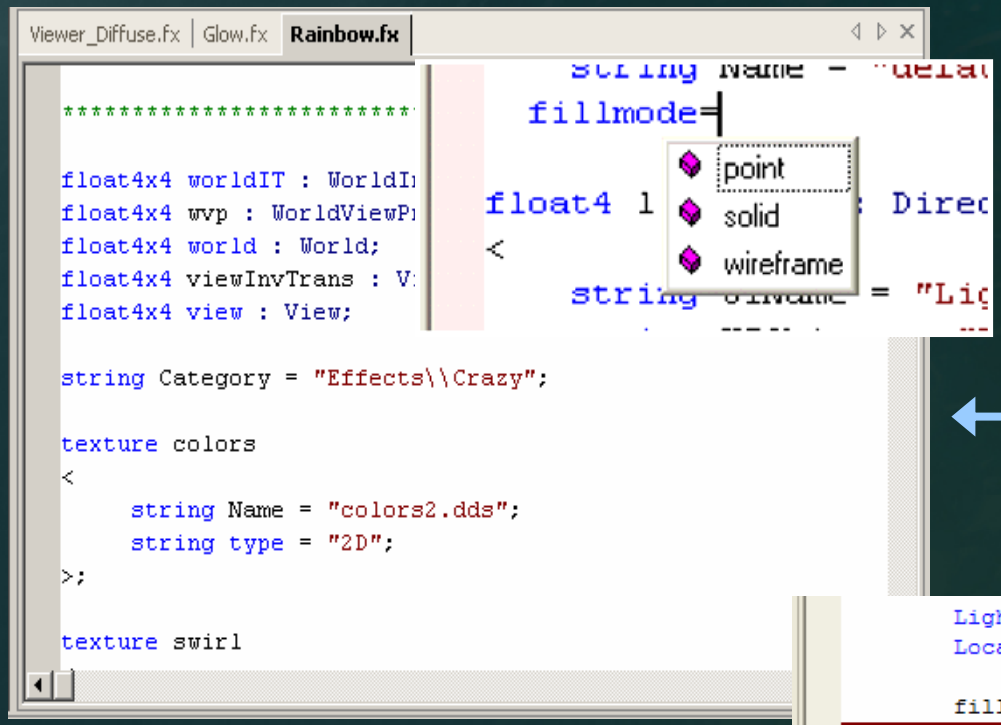


- シーンパネルから、これらをシーンの任意の部位に適用可能

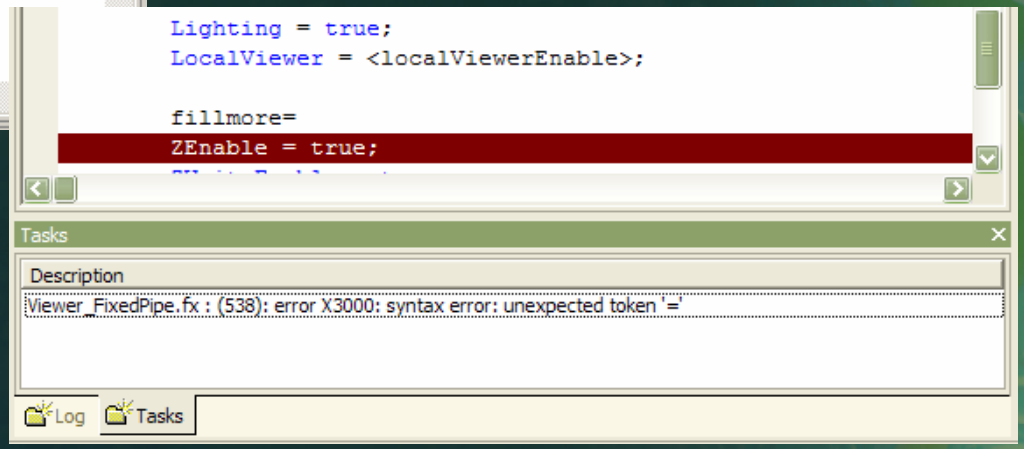


- 入力テクスチャの閲覧
- レンダー・ターゲットの確認
- テクスチャのファイル保存

編集とデバッグ

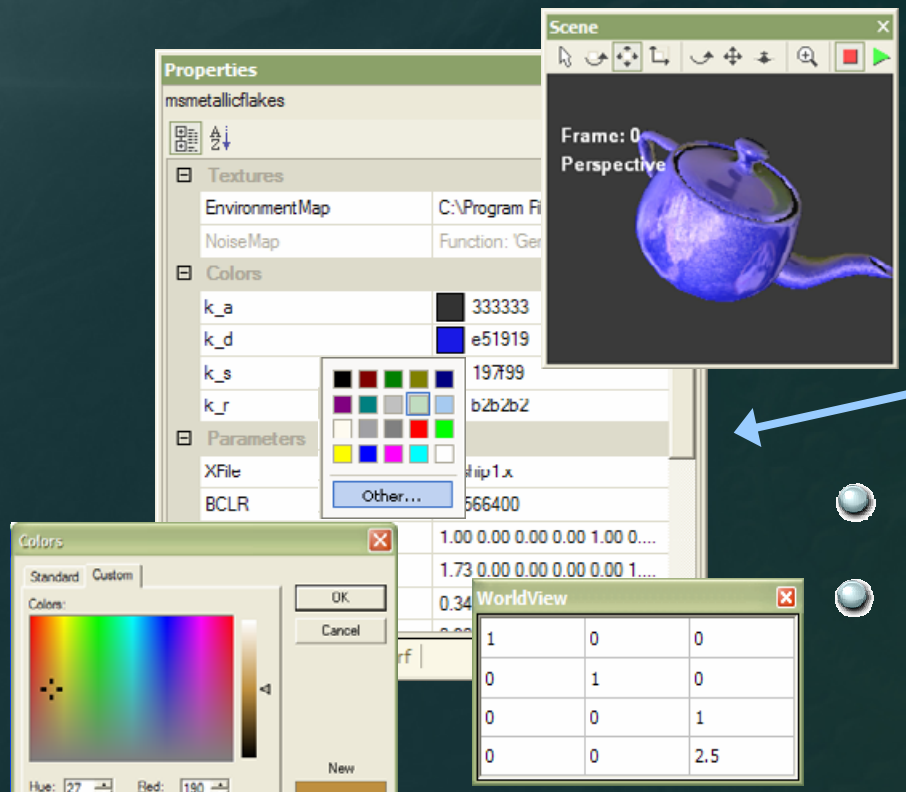


- 複数の.FXファイルを編集
- Intellisense (自動補足)
- プログラム文法による色付け

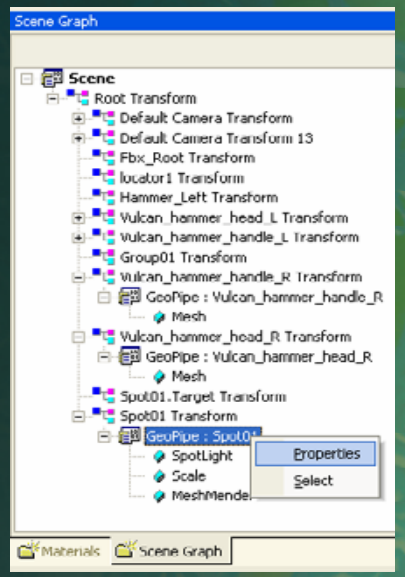


- エラー箇所への自動移動で、すばやいエラー修正が可能

閲覧とカスタマイズ

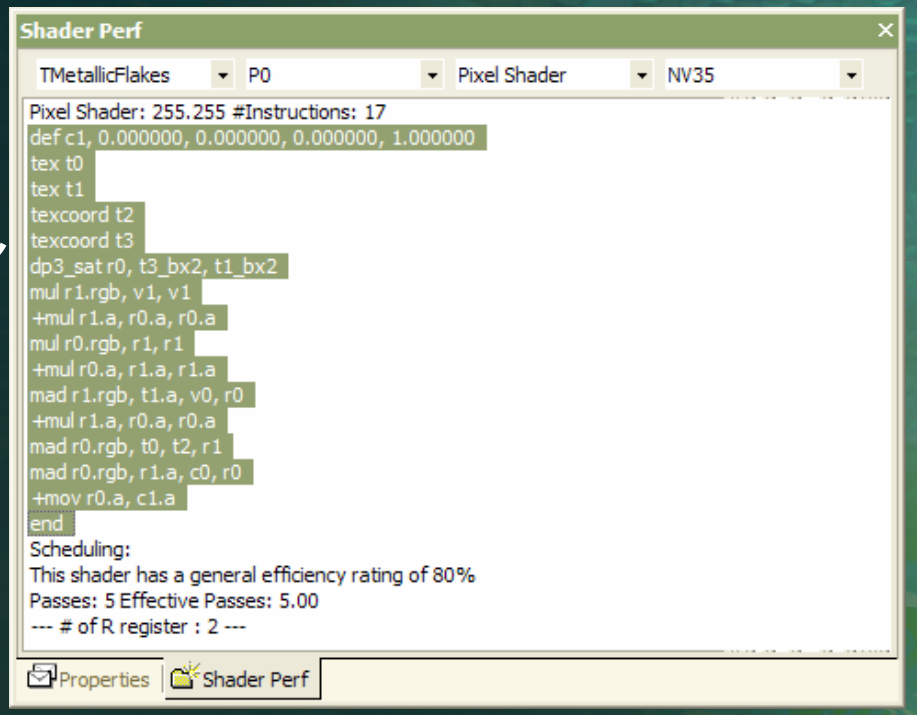


- シェーダーの属性値を手軽に調整
- セマンティクスとアノテイションの自動認識
- 手軽に色値を選択
- ダイアログにより、ベクターや行列の値を編集
- Scene Graphパネルにより、シーンの任意の部位の属性を調整

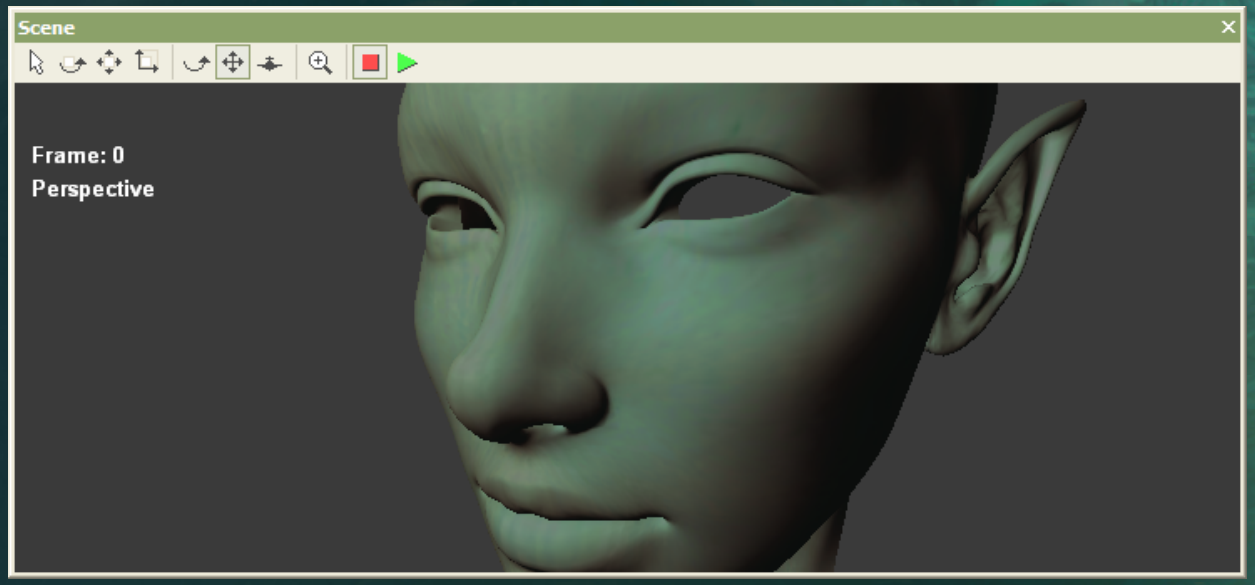


シェーダーのパフォーマンス最適化

- Shader Perfパネル
- テクニック、パス、バーテクス/ピクセルシェーダを選択して解析
- 最近の多くのNVIDIA製GPUでのパフォーマンスを確認
- DirectXアセンブリの最適化
- NVIDIAパフォーマンス解析
  - GPUサイクル数
  - 効率 / 使用率の評価
  - 必要なパスの数
  - レジスタの使用状況



NVIDIA FX Composer



Sceneパネル

- 読み込んだシーンをリアルタイムで確認
  - シーンの各部位にマテリアルを適用
  - シーンの各部、または全体を操作
  - 基本的な図形を選ぶか、.xや.nvbフォーマットのファイルを読み込む
  - キーフレームの設定や、既存のアニメーションを実行
  - ライトの設置とライト属性の設定
  - 設定したカメラか、デフォルトのカメラを使用

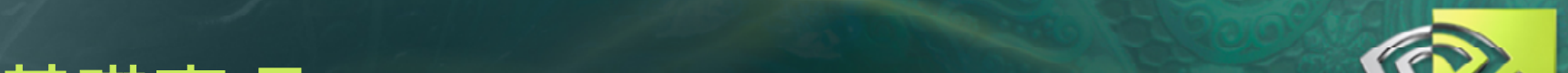
FX Composerよくある質問

- CgやGLSLのサポートは?
- 独自エンジンへの組み込みは?
- DCCアプリケーションとの関連は?
- RenderMonkeyやEffect Editなどと比較すると?
- ウィンドウズ以外の環境では?

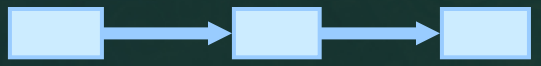
最新のFX ComposerのFAQは以下で
http://developer.nvidia.com/fxcomposer

パフォーマンス最適化

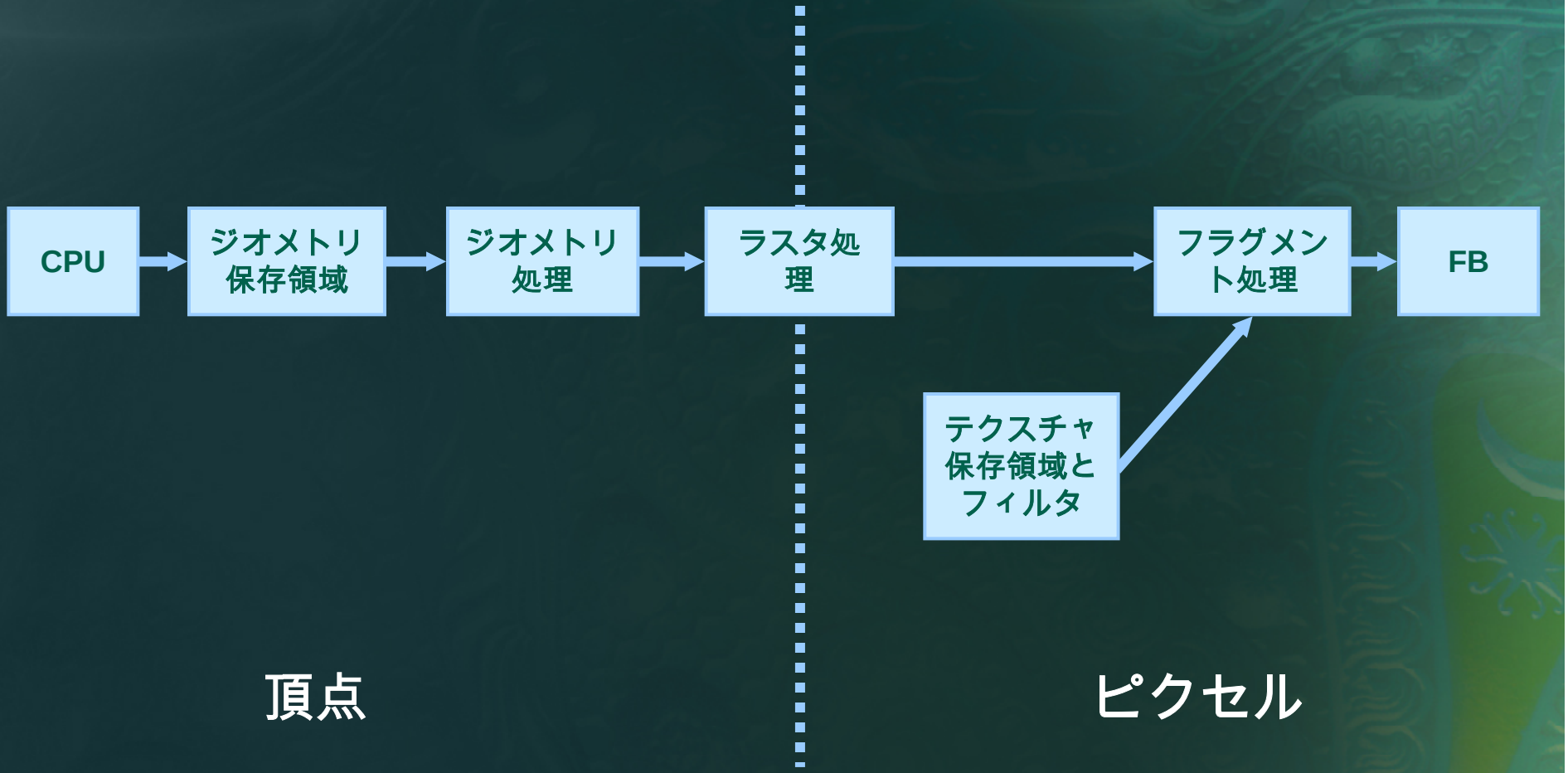
基礎事項



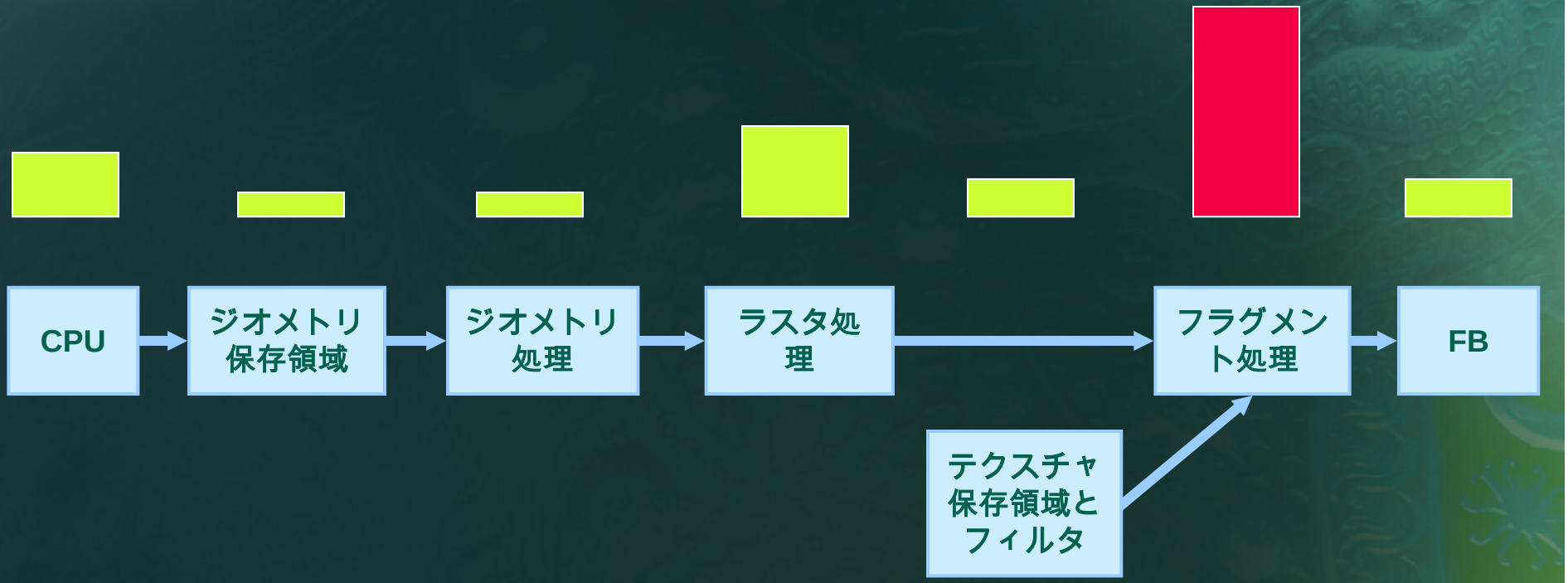
- パイプライン構成
- ボトルネックの発見と修正
- パイプラインのバランス



パイプライン構成

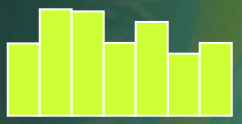


ボトルネックの存在

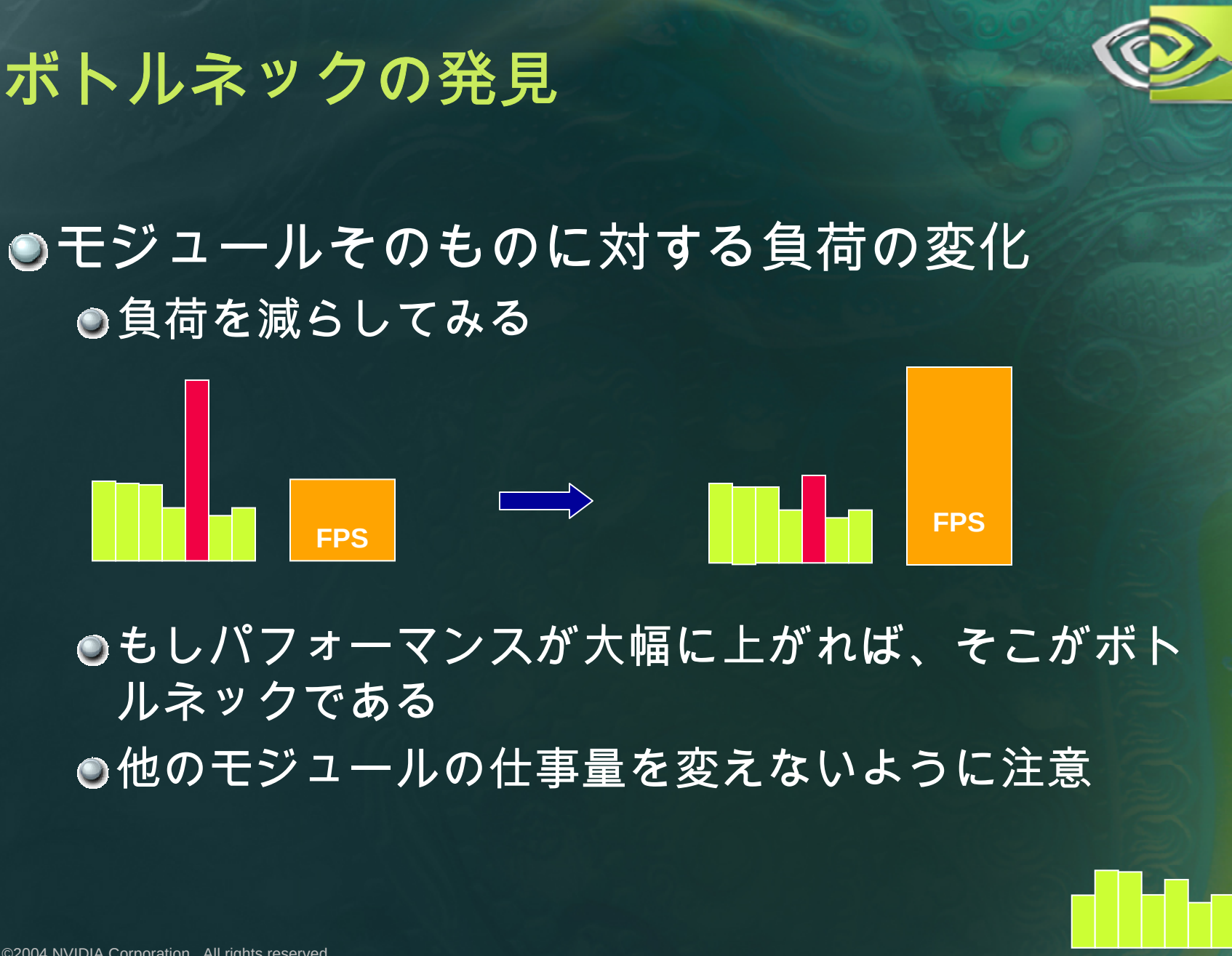


ボトルネックの発見

- ふたつの方法
- モジュールそのものに対する負荷の変化
- 他のモジュールに対する負荷の変化



ボトルネックの発見

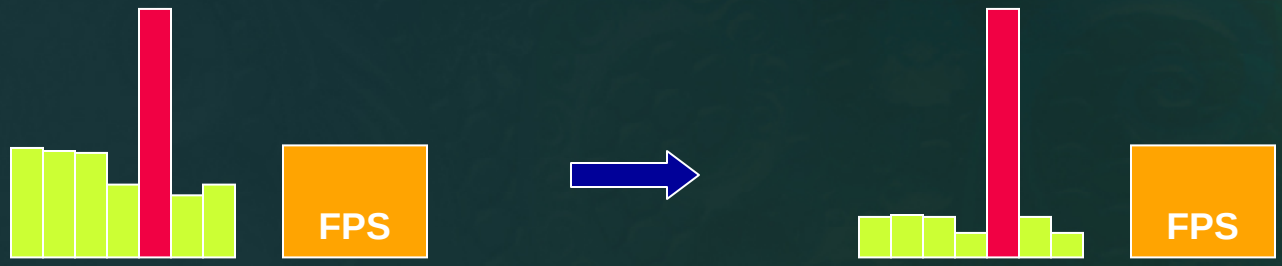


- モジュールそのものに対する負荷の変化
  - 負荷を減らしてみる

  - もしパフォーマンスが大幅に上がれば、そこがボトルネックである
  - 他のモジュールの仕事量を変えないように注意

ボトルネックの発見

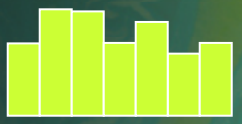
- 他のモジュールに対する負荷の変化
  - 他のユニットの仕事量を非常に小さくする



  - パフォーマンスが大幅に向上しない場合、このモジュールがボトルネック
  - このモジュールの仕事量を変化させないように注意

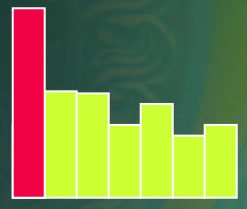
ボトルネックの発見

- 多くの場合、ひとつのモジュールに対する変更が他のモジュールにも影響
- テスト項目の選択は困難
- テスト項目を見てみる



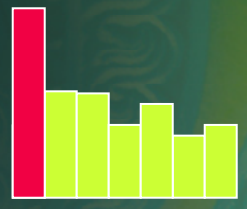
ボトルネックの発見: CPU

- 原因
- ゲームそのもの
  - 物理シュミレーション、AI、ゲームのロジック
  - メモリ管理
  - データ構造
- API使用方法の間違い
  - D3Dのデバッグ環境でエラーやワーニングを確認
- ビデオ・ ドライバ
  - バッチ数が多すぎる



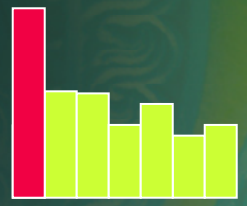
ボトルネックの発見: CPU

- 仕事量の軽減
- 一時的に、処理を行わない
  - ゲームのロジック
  - AI
  - 物理シュミレーション
  - その他にも、CPUで行っている大きな処理
    - 描画作業量を変更しないように注意



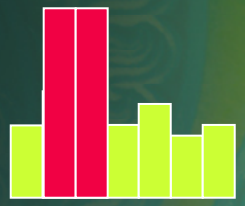
ボトルネックの発見: CPU

- 他のモジュールの仕事量
- DrawPrimitiveを呼ばない
  - 描画準備などを普通に行っておき、DrawPrimitive呼び出しのみを行わない
  - 問題点: API呼び出しの際に実行環境やドライバが何を行うか不透明
- VTUNEやNVPerfHUDを使う



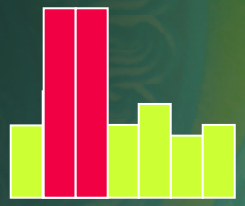
ボトルネックの発見: 頂点

- 原因
- 頂点やインデックスのGPUへの転送
- 頂点やインデックスから三角形を作る処理
- 頂点キャッシュの効率が良くない
- 時間のかかる頂点シェーダー



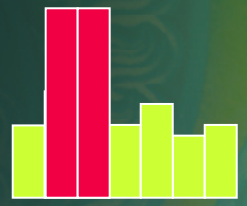
ボトルネックの発見: 頂点

- 仕事量の軽減
- 頂点シェーダーを簡単にする
- 転送する三角形を減らす??
  - これでは良くない
- AGPメモリを減らす??
  - これもあまりよくない
  - NVPerfHUDでビデオメモリの使用量を確認
  - もしいっぱいならテクスチャがAGPメモリにあるかもしれない



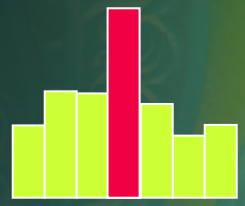
ボトルネックの発見: 頂点

- 他のモジュールの仕事量
- バックバッファを小さくする – これによって以下のモジュールが確認できる
  - テクスチャ、フレームバッファ、ピクセル・シェーダー
- CPUボトルネックの確認は必要
- バックバッファではなく、ビューポートを小さくしても良い



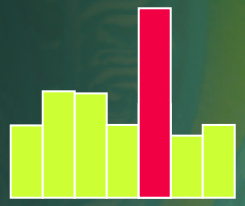
ボトルネックの発見: ラスタ処理

- ボトルネックであることはほとんどない



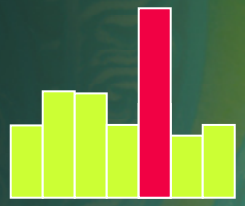
ボトルネックの発見: テクスチャ

- 原因
- テクスチャ・ キャッシュの効率が良くない
- テクスチャが大きすぎる
- 転送幅が足りない
- AGPからのテクスチャ転送



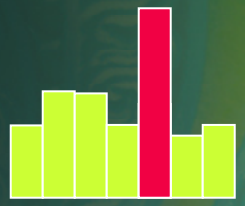
ボトルネックの発見: テクスチャ

- 仕事量の軽減
- 小さいテクスチャ(2x2)を使用してみる
  - アルファを使っていれば、仕事量が増えることもあるが、一般的に良いテスト
- ミップマップを使用してみる
- 異方性フィルタを使用しない



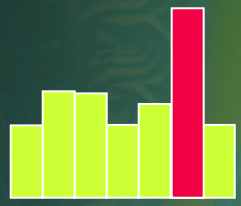
ボトルネックの発見: テクスチャ

- 他のモジュールの仕事量
- 直接このモジュールをテストしたほうが良い



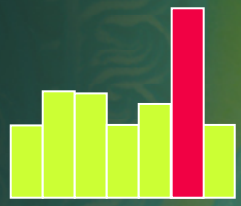
ボトルネックの発見: フラグメント

- 原因
- 時間のかかるピクセル・シェーダー
- 必要以上の数のフラグメントを処理
  - Z方向に複雑すぎる
  - Zカルが効いていない



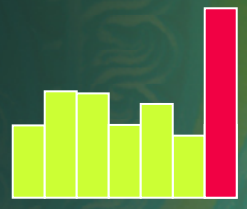
ボトルネックの発見: フラグメント

- 仕事量の軽減
- 単純色の出力
  - フラグメントごとの仕事量の軽減
  - テクスチャの仕事量も変わるので、そちらを先に確認する必要がある
- 計算の簡略化
  - フラグメントごとの仕事量の軽減
  - テクスチャに対するアクセスを変更しないように注意



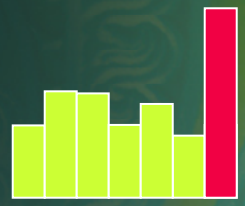
ボトルネックの発見: FB

- 原因
- 必要以上の回数バッファにアクセスしている
  - マルチパス
- アルファブレンディングが多すぎる
- バッファが大きすぎる
  - ステンシルは必要なければ使わない
  - 動的な反射環境マップなど、R8G8B8を使わなくてもR5G6B5ですむことが多い



ボトルネックの発見: FB

- 仕事量の軽減
- 24ビットの代わりに16ビットのZバッファを使う
- 32ビットの代わりに16ビットの色バッファを使う



実践

実践: システムをきれいにする

- 正しいドライバの使用
- ゲームはリリース版を使う(最適化を有効にする)
- D3Dのデバッグ出力を確認、しかし...
- リリース環境を使ってテスト
- Maximum Validationは使わない
- ドライバーによる異方性フィルタやアンチエイリアスを無効にする
- V-syncはオフ

実践: 例1

実践: 例1

- 動的頂点バッファ
  - 作成時のフラグが問題

```
HRESULT hr = pd3dDevice->CreateVertexBuffer(
                        6* sizeof( PARTICLE_VERT ),
                        0,        //declares this as static
                        PARTICLE_VERT::FVF,
                        D3DPOOL_DEFAULT,
                        &m_pVB,
                        NULL );
```

実践: 例1

- 動的頂点バッファ
  - 正しいフラグに修正

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

実践: 例1

- 動的頂点バッファ
  - ロックのフラグも問題

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

- フラグを使用しない!?
  - これではうまくいかない....

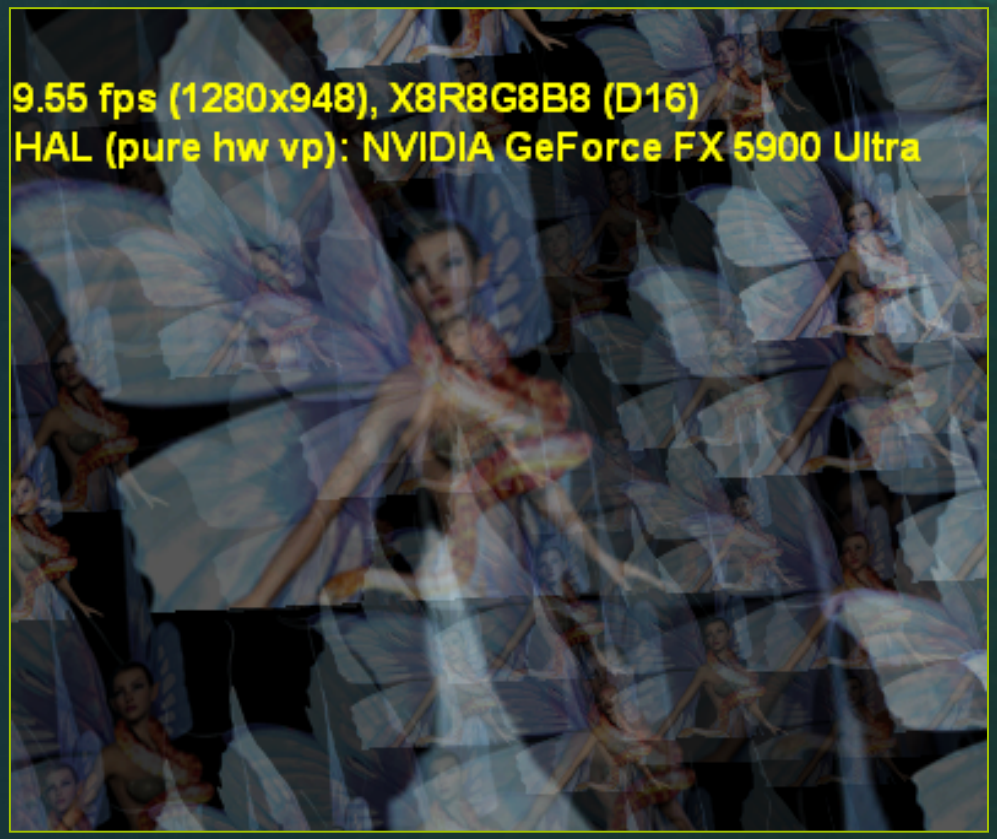
実践: 例1

- 動的頂点バッファ
  - ロックのフラグを修正

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

- 各フレームで最初にロックするときに D3DLOCK_DISCARDを使用
  - バッファがいっぱいになるたびに使用
  - その他ではNOSYSLOCKを使用

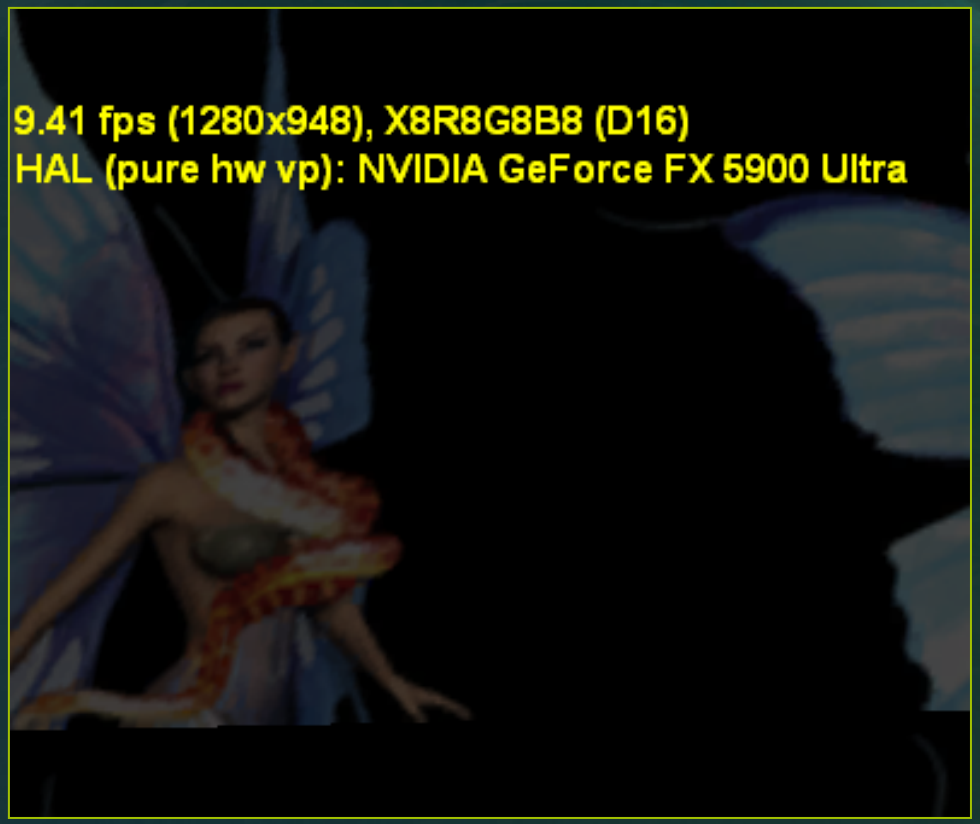
実践: 例2



実践: 例2

- テクスチャの帯域幅
- ミップマップを使用する
- 可能ならDXT1を使用する
  - カードによっては圧縮した形でキャッシュに格納
- テクスチャを小さくする
  - 例えば草の葉が1024x1024であるひつようがあるでしょうか？

実践: 例3



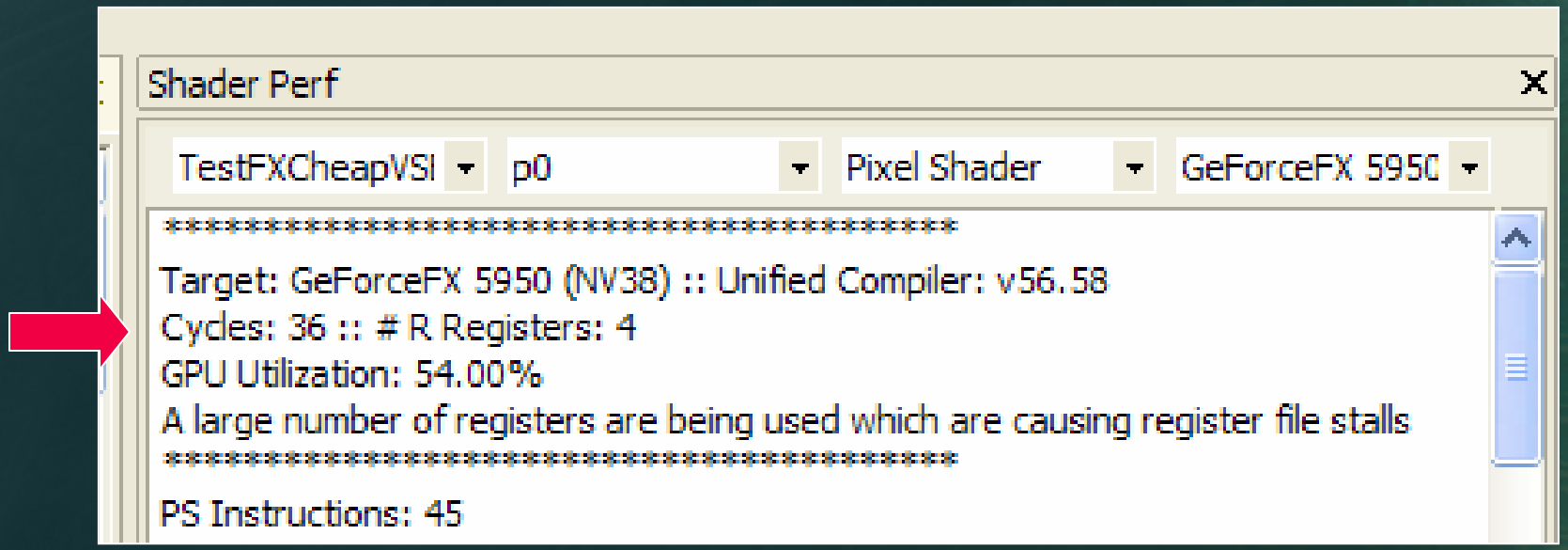
実践: 例3

- 時間のかかるピクセル・シェーダー
- パフォーマンスに対して影響大。
- たった3つの頂点、しかし100万のフラグメント
  - 1024x1024のケース

非常にたくさんのピクセル

実践: 例3

- 36サイクルはよくない



実践: 例3

- 11サイクルならずっと良い

Shader Perf

TestFXCheapVSI | p0 | Pixel Shader | GeForceFX 595C

```
************************************************
Target: GeForceFX 5950 (NV38) :: Unified Compiler: v56.58
Cycles: 11 :: # R Registers: 2
GPU Utilization: 54.00%
************************************************

PS Instructions: 26
```

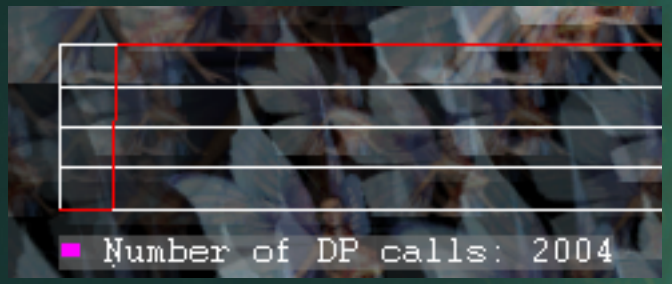
実践: 例3

- 変更点
- 三角形で一定な演算を頂点シェーダーで実行
- floatの代わりにhalfを使用
- 必要のないベクトル正規化を除いた
  - Normalization Heuristicsに詳細
  - http://developer.nvidia.com

実践: 例4

実践: 例4

- バッチ数が多すぎる
- 各四角形を全てここのバッチで送っていた。
- 全ての四角形をひとつの頂点バッファで転送

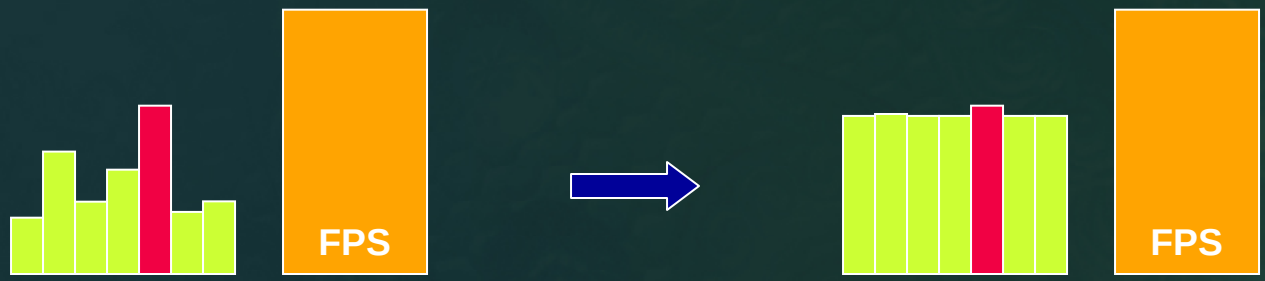


実践: 例4

- 使用するテクスチャが違ったときは？
- テクスチャ・アトラス
- ふたつのテクスチャをひとつにまとめて、バーテクスやピクセルシェーダーでテクスチャ座標を調整

パイプラインのバランス

- ボトルネックでないモジュールを更に使って、パイプラインのバランスを取る
- あまりやりすぎると問題があることもある



まとめ

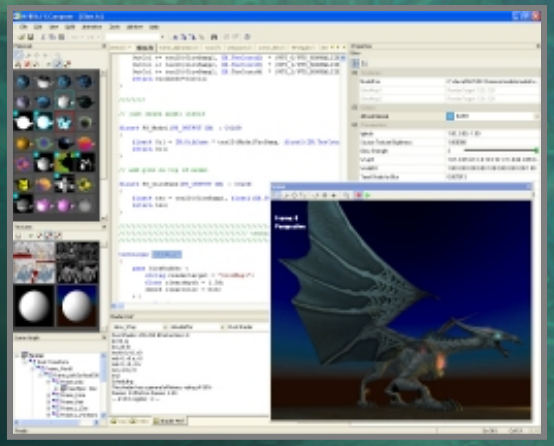
- NVIDIAは多くのパフォーマンス解析ツールを提供している
- パイプライン構成はボトルネックによってのみ制限される
- テストを行って、ボトルネックを発見
- NVPerfHUDによるパイプラインの解析
- FX Composerでのシェーダー最適化

情報源...

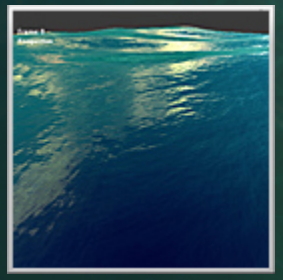
- プレゼンテーション、SDK、ツールのダウンロード http://developer.nvidia.com
- ツールやSDKに関する質問
  sdkfeedback@nvidia.com
- このセッションについて
  kashida@nvidia.com

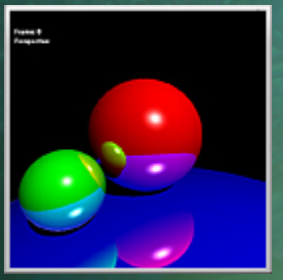
developer.nvidia.com
GPUプログラムの情報源

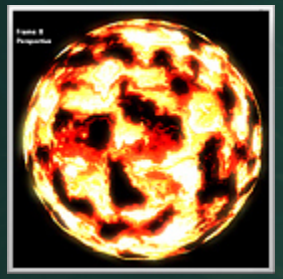
- 最新の文献
- SDK
- 最新鋭ツール
  - パフォーマンス解析ツール
  - データ作成ツール
- 何百ものエフェクト
- ビデオによるプレゼンテーション
- ライブラリ
- 最新情報とそのアーカイブ

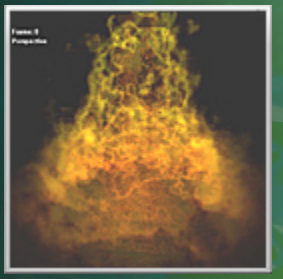


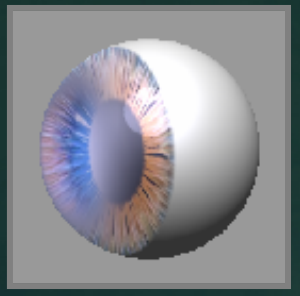
EverQuest® content courtesy Sony Online Entertainment Inc.

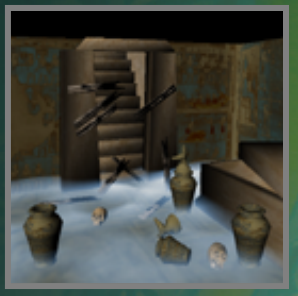










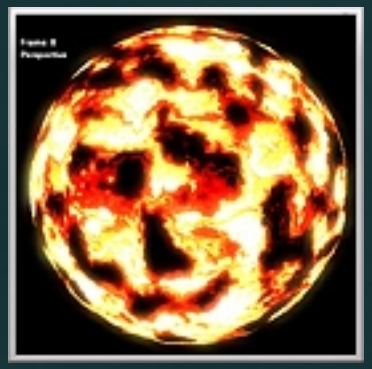


NVIDIA SDK

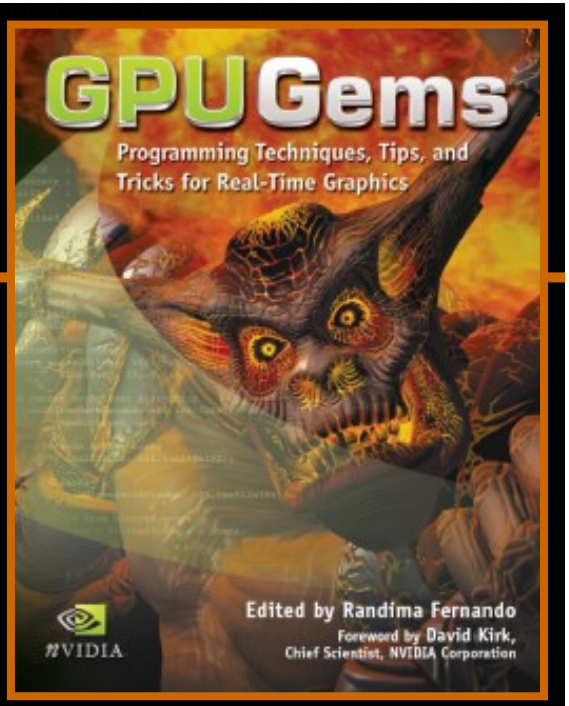
リアルタイム開発者の情報源

何百ものコードサンプルとエフェクトで最新グラフィクス技術の利用を助ける

- 膨大な量の新しいDirectXやOpenGLの解説付コードサンプル:
  - ジオメトリ・インスタンシング、レインボウ/フォグボウ、ブラッド・シェーダー、パースペクティブ・シャドウマップ、テクスチャ・アトラスツール...
- ジオメトリやアニメーション付の何百ものエフェクト:
  - スキン、プラスチック，炎、てかり、ゴーチ、画像フィルタ、HLSLデバッグテクニック、テクスチャBRDF、テクスチャ・ディスプレイスメント、トーンマッ[illegible]トレー[illegible]



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



- 先端企業、大学などのエキスパートによる、実践的なリアルタイム・グラフィクス技術解説
- 豊富な内容:
  - 業界のエキスパートによる執筆
  - 前頁カラー (300以上の図やスクリーンキャプチャ)
  - ハードカバー
  - 816ページ

詳細は:
http://developer.nvidia.com/GPUGems

"*GPU Gems* is a cool toolbox of advanced graphics techniques. Novice programmers and graphics gurus alike will find the gems practical, intriguing, and useful."

Tim Sweeney

Lead programmer of *Unreal* at Epic Games

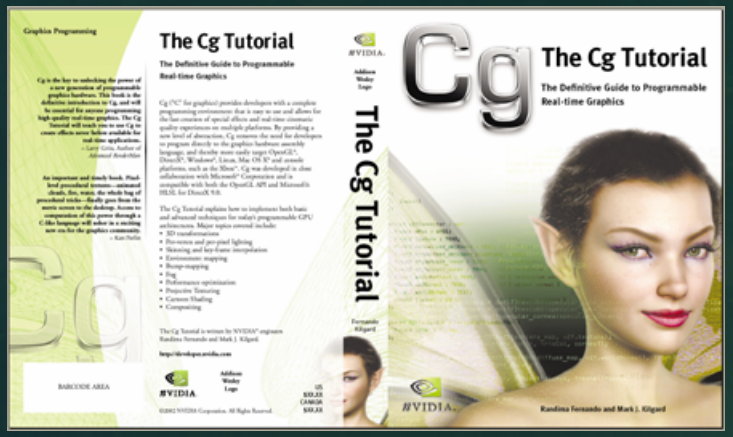
"This collection of articles is particularly impressive for its depth and breadth. The book includes product-oriented case studies, previously unpublished state-of-the-art research, comprehensive tutorials, and extensive code samples and demos throughout."

Eric Haines

Author of *Real-Time Rendering*

The Cg Toolkit

- NVIDIA Cgコンパイラ
  - 頂点 (DirectX 9, OpenGL 1.4)
  - ピクセル (DirectX 9)
- Cg標準ライブラリ
- DirectXとOpenGLのためのCg実行環境
- NVIDIA Cgブラウザ
- Cg言語仕様書
- Cgユーザー・マニュアル
- Cgシェーダー
  (サンプルプログラム)



- The Cg Tutorial
  (developer.nvidia.com/CgTutorial)