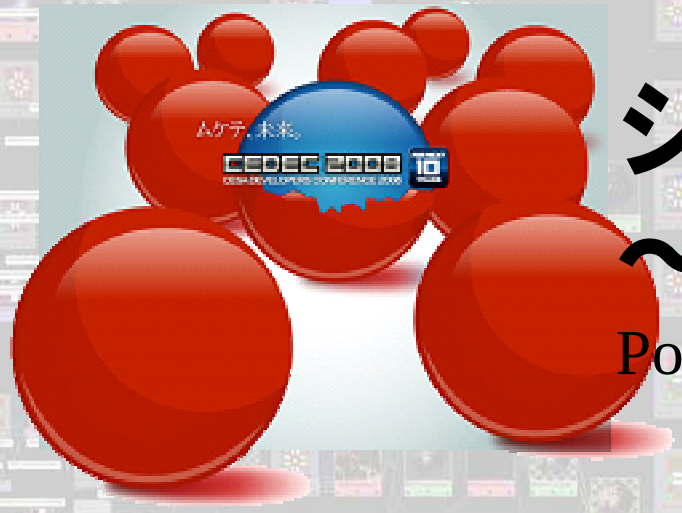


シェーダ管理事例
～自由度とひきかえに～

Postmortem of Shader Management

(株) トライエース 研究開発部
五反田義治 / 庄子達哉

Research and Development Department tri-Ace

このセッションについて

- 弊社開発ゲームエンジンにおいて
  - シェーダ(マテリアル)製作
  - シェーダ管理
  - それらのメリットおよびデメリット
    - 高度な技術というよりは泥臭いお話☺

シェーダ管理手法(1)

- 大きなシェーダの中で(静的)分岐(Ubershader)
  - メリット
    - シェーダ管理の概念がソースレベル
    - 管理が簡単?
      - 複雑になればかえって簡単ではない
    - シェーダバイナリサイズ
      - 非常に小さい
      - 管理が不要
  - デメリット
    - パフォーマンス的に劣る場合がある
      - パフォーマンスが原因でシェーダの自由度が制限されるケース
    - シェーダの自由度が完全にプログラマに依存する

シェーダ管理手法(2)

- 手動でシェーダのバリエーションを作成
  - 一番多いパターン?
  - メリット
    - パフォーマンスをコントロールしやすい
    - シェーダバイナリ
      - サイズが少ない
      - 管理もほとんど必要ない
  - デメリット
    - シェーダのバリエーションを手動で生成
      - シェーダプログラマのマンパワーに依存

シェーダ管理手法(3)

- 動的にシェーダを作成(オフライン)
  - メリット
    - パフォーマンスはコンパイラの性能に依存
      - それほど悪くない
    - デザイナーがシェーダを自由に作成できる
  - デメリット
    - シェーダバイナリ
      - リソースファイルに付属する
      - すべてのバリエーションを事前に生成する必要がある
        - » 半リアルタイムでランタイムで生成する方法も
    - シェーダバイナリの管理が煩雑
      - バージョン管理

シェーダ管理手法(4)

- ランタイムにシェーダを生成
  - 弊社が選択した手法
  - メリット
    - リソースファイルにシェーダを追加する必要が無い
    - デザイナーがシェーダを自由に作成できる
    - シェーダのランタイムのバリエーションに簡単に対応できる
    - シェーダバイナリの管理は簡単
  - デメリット
    - シェーダバリエーション数の爆発
      - シェーダバイナリサイズの肥大化
    - すべてのシェーダのバリエーションの作成
      - 可能性のあるすべてのゲーム内容をプレイしないといけない

デザインポリシー

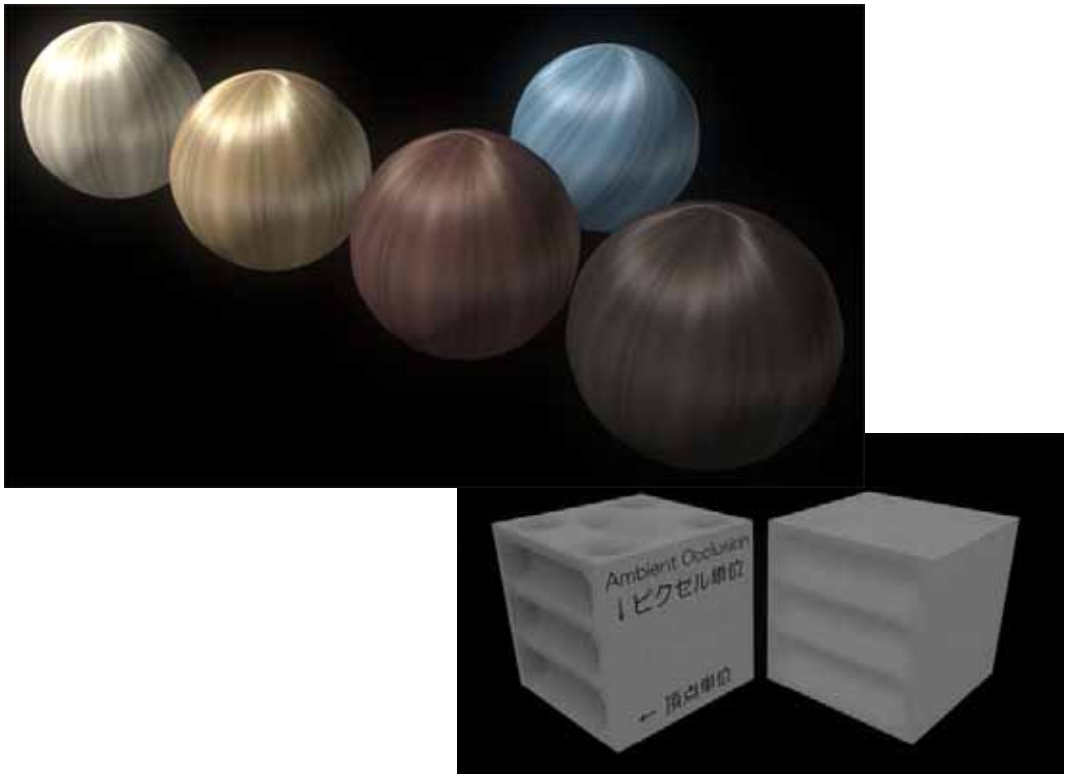
- Maya上でデザイナーがシェーダを構成
  - 既定のシェーダ以外のシェーダをデザイナーのみで作成できる
    - プログラマーが想像しない新しい表現
    - 試してみたいことをすぐ試すことが出来る
  - デザイナーの教育が必要
    - パラメータの設定やシェーダの構成の仕方など
    - ある程度の物理的知識
    - テンプレートが必要

シェーダの細分化

- そのため各シェーダノードを細分化
  - それぞれの小さい機能をひとつのシェーダノードとして実装
    - Mayaのシェーダノードに対応
  - それぞれのInputとOutputをデザイナーが自由につなぐことが出来る
    - UV, Color, Normal, Alpha…

シェーダ種別 - ライティング

- ライティングされた結果によりサーフェースをシェーディングする
  - Phong
  - Blinn-Phong
  - Anisotropic Phong
  - Normalized Blinn
  - Ashikmin
  - Kajiya-Kay
  - Marschner
    - Albedo Map
    - Specular Map
    - Gloss(shiness) Map
    - Fresnel Map
    - Offset Map
    - Translucency
    - Ambient Occlusion

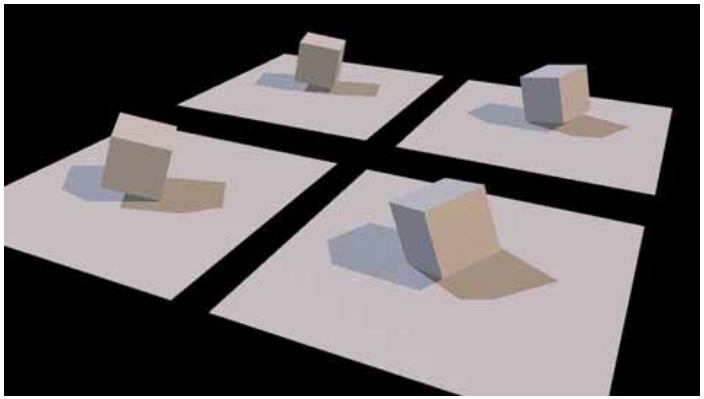


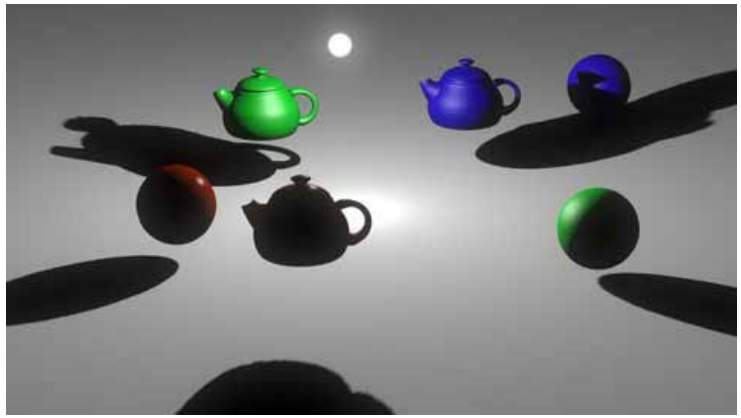
シェーダ - Normal, UV系

- Normal**系**
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping
- UV**生成系**
  - **反射**
    - Sphere
    - Dual Paraboloid
    - Cube
  - UV**オフセット(屈折)**
  - UV**アニメーション**
- Blur**系**
  - Rectangle Distortion
  - Spherical Distortion
  - Radial

シェーダ - Shadow, Projection系

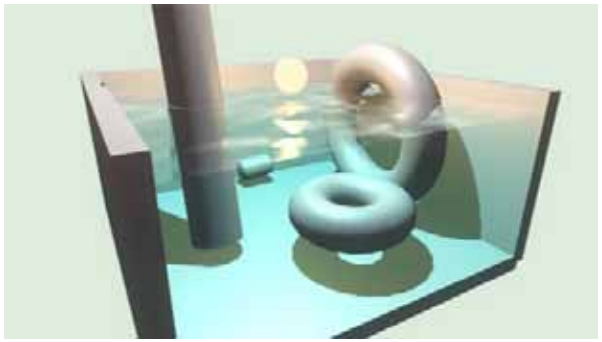
- Shadow系
  - Multiple Uniform
  - Multiple LiSPSM
  - Omnidirectional Cube Shadow
  - Cascade Shadow
    - PCF
  - Projection Shadow
  - Static Vertex Shadow
  - Static Shadow
- Projector系
  - Color
  - Cube
  - Normal map





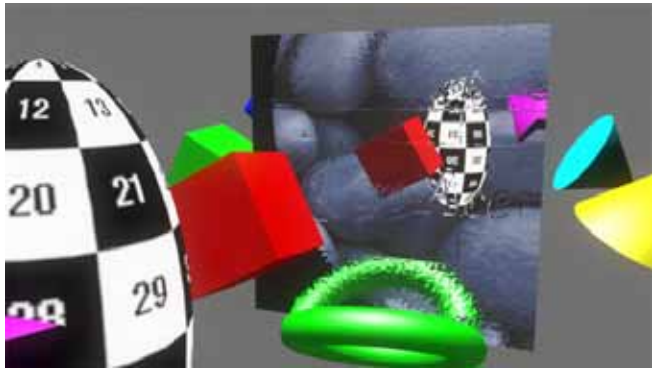
シェーダ - 計算系

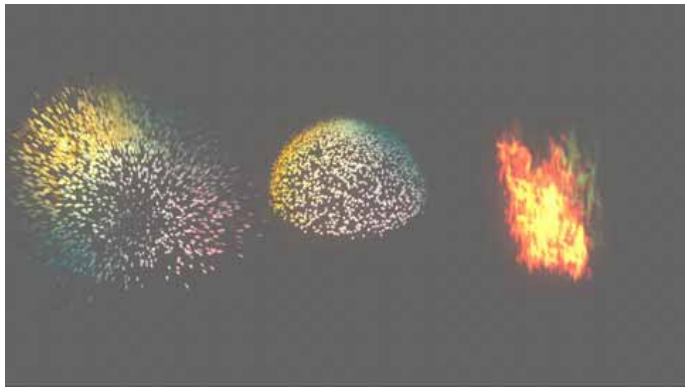
- ブレンディング
  - 計算されたカラーやNormalなどをシンプルにブレンディングするシェーダノード
    - カラーの加算、減算、アルファブレンディング、Photoshop的な処理
  - Normalのブレンディング
- Absorption
- Swizzle
- Depth系
  - Depthをベースに処理をする
    - Detail Mapping
      - DepthをベースにNormal Mapをブレンディング
    - シェーダベースのLOD
- Branch系
  - シェーダの処理を状況によって分岐できるシェーダ
    - (HWによって)高速化に利用できるケースもある



シェーダ - その他

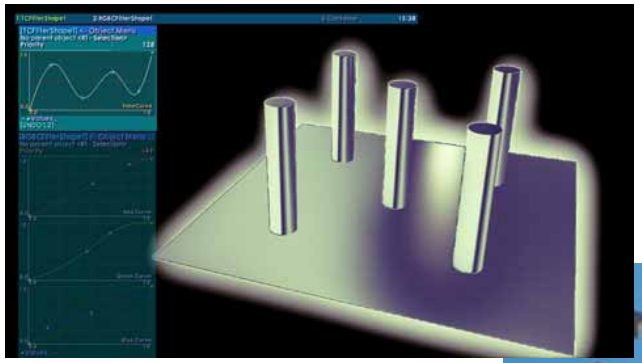
- Alpha**系**
  - Fresnel
  - Soft Polygon
- **テクスチャサンプリング**
  - **テクスチャフェッチをするシェーダノード**
    - **通常のテクスチャ**
    - **マルチパステクスチャ**
    - **グローバルテクスチャ**
  - **圧縮**HDR**テクスチャ**
- Vertex**系**
  - Skinning
  - Volume Rendering
  - Vertex Color
- **パーティクルレンダリングシェーダ**
  - **いろいろな種類のパーティクルレンダラ**
- **その他細かいものいろいろ**



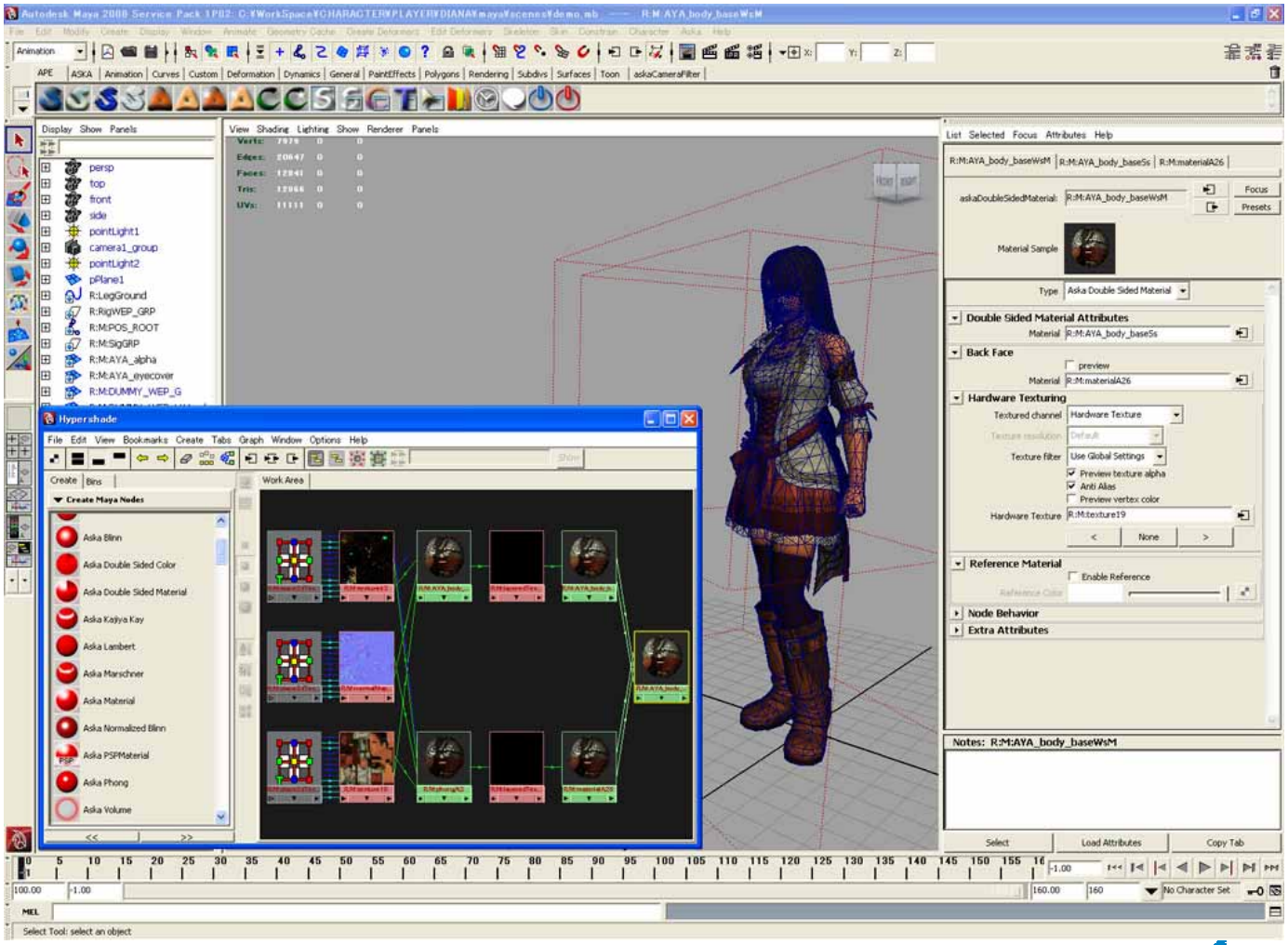


シェーダ - ポストエフェクト系

- トーンマッピング
  - 通常のトーンマッピング
  - Film Simulation
    - フィルム(or CCD)特性の再現
    - フィルムグレイン、デジタルノイズの再現
  - Dithering
- レンズシミュレーション
  - Focus Blur (DOF)
  - Glare
  - Physically-based Lens Structure
  - モーションブラー
    - Camera
    - Object
- カラーフィルタ
  - Contrast
  - Brightness
  - Monotone
  - Tone Curve
  - Color Temperature
- スキャッタリングシミュレーション
  - Outdoor Light Scattering
  - Light Shaft simulation

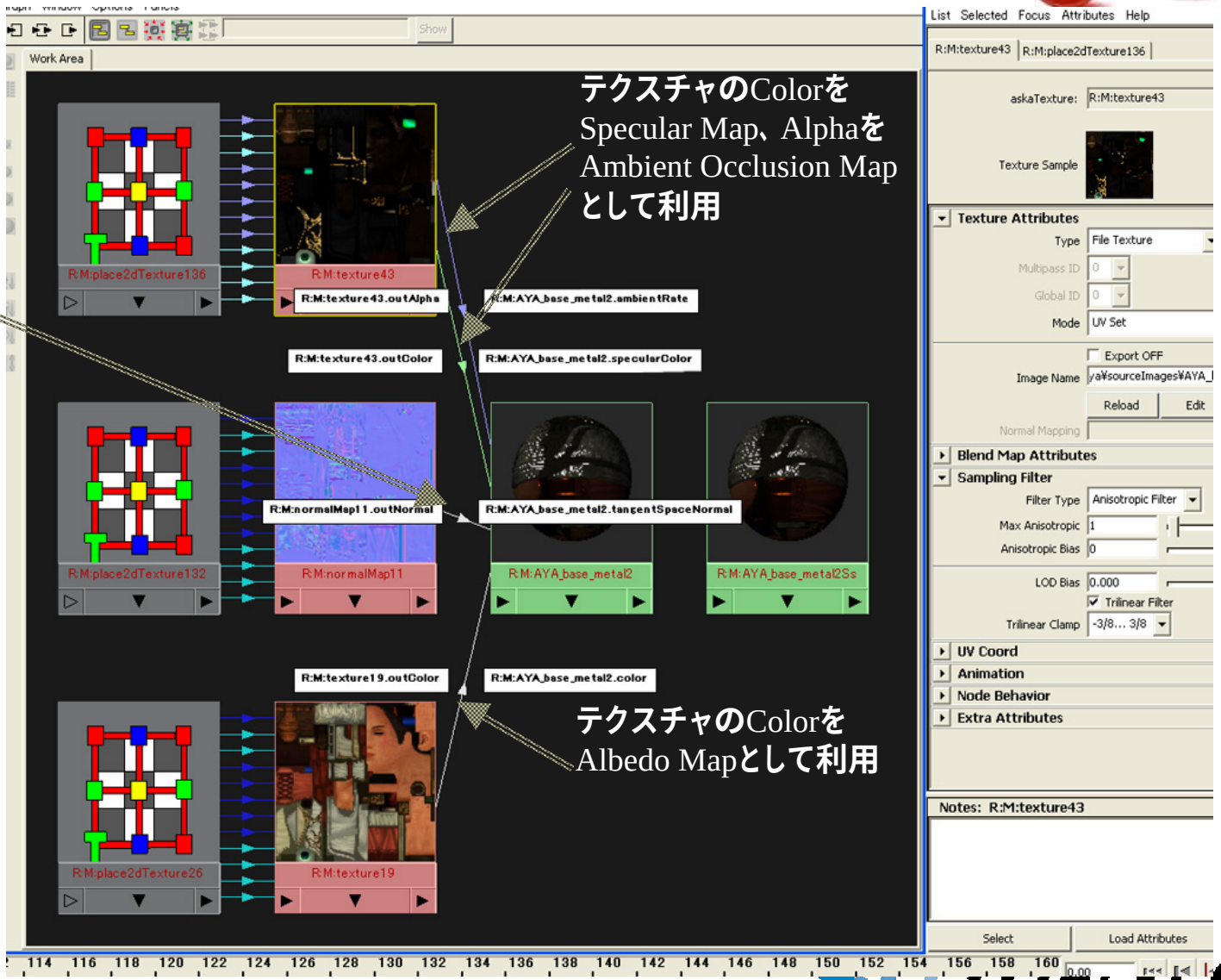


エディット画面

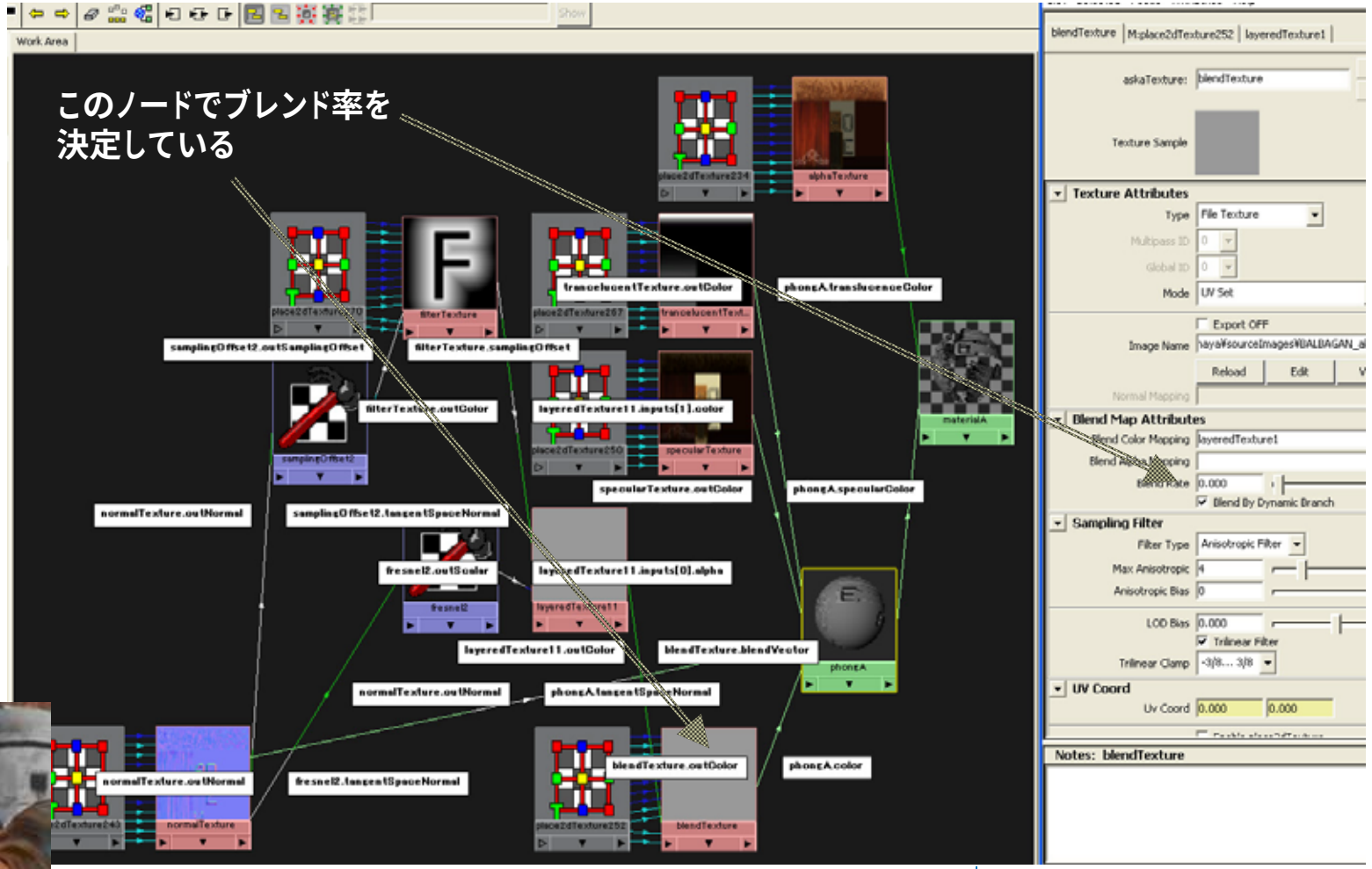

Hypershade
Material Sample
Double Sided Material Attributes
Back Face
Hardware Texturing
Reference Material
Node Behavior
Extra Attributes
Notes: R:M:AYA_body_baseWsM

基本的なシェーダの例

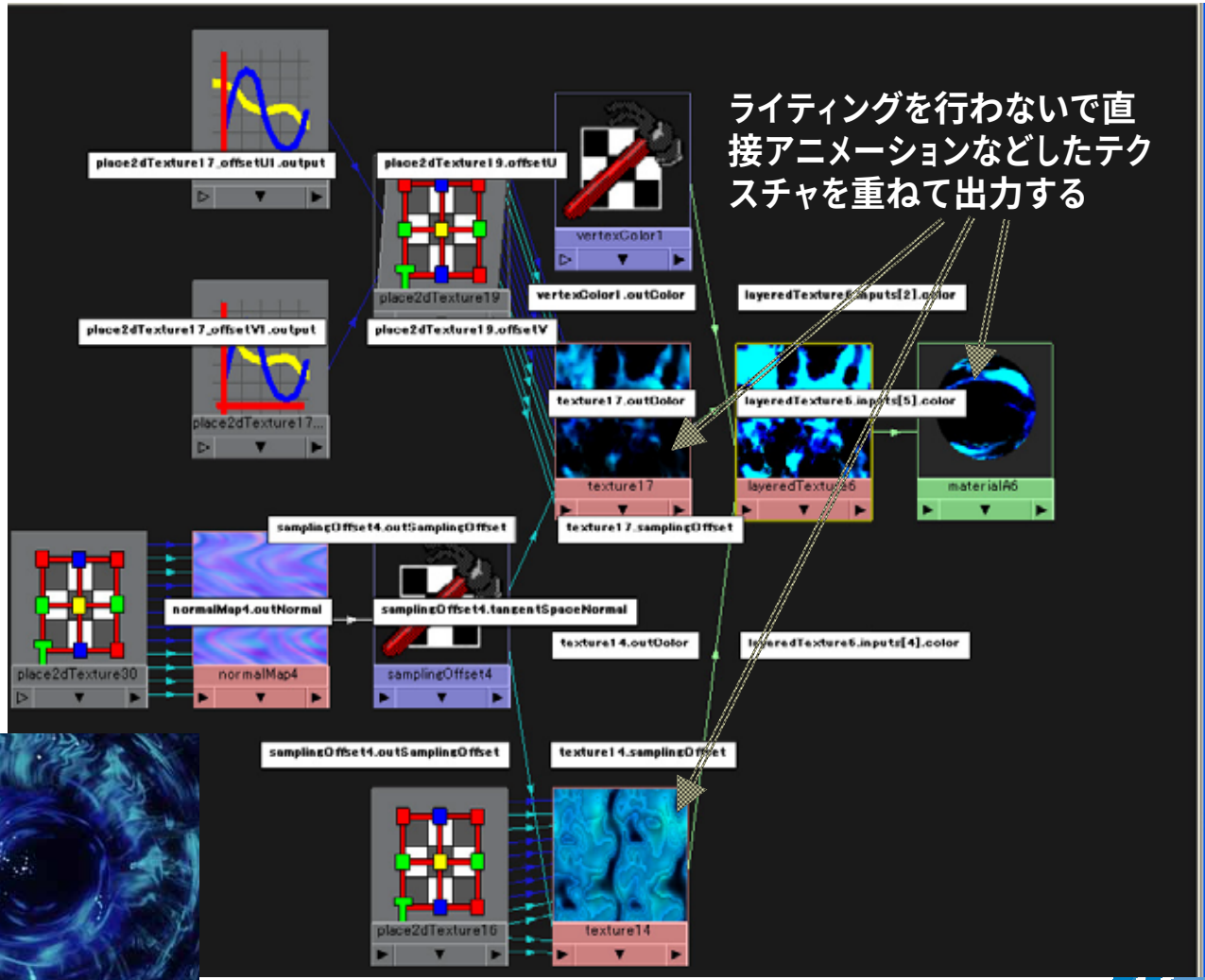
テクスチャをノーマルとして出力し、Anisotropic Phongシェーダ側でTangent Space Normal Mapとして利用

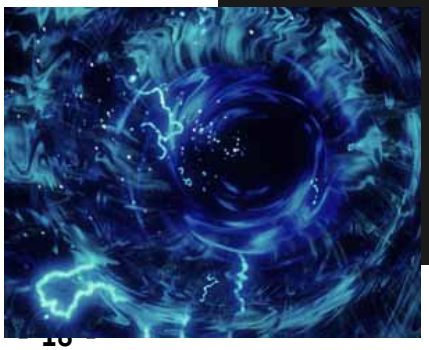


ランタイムで変化するシェーダの例

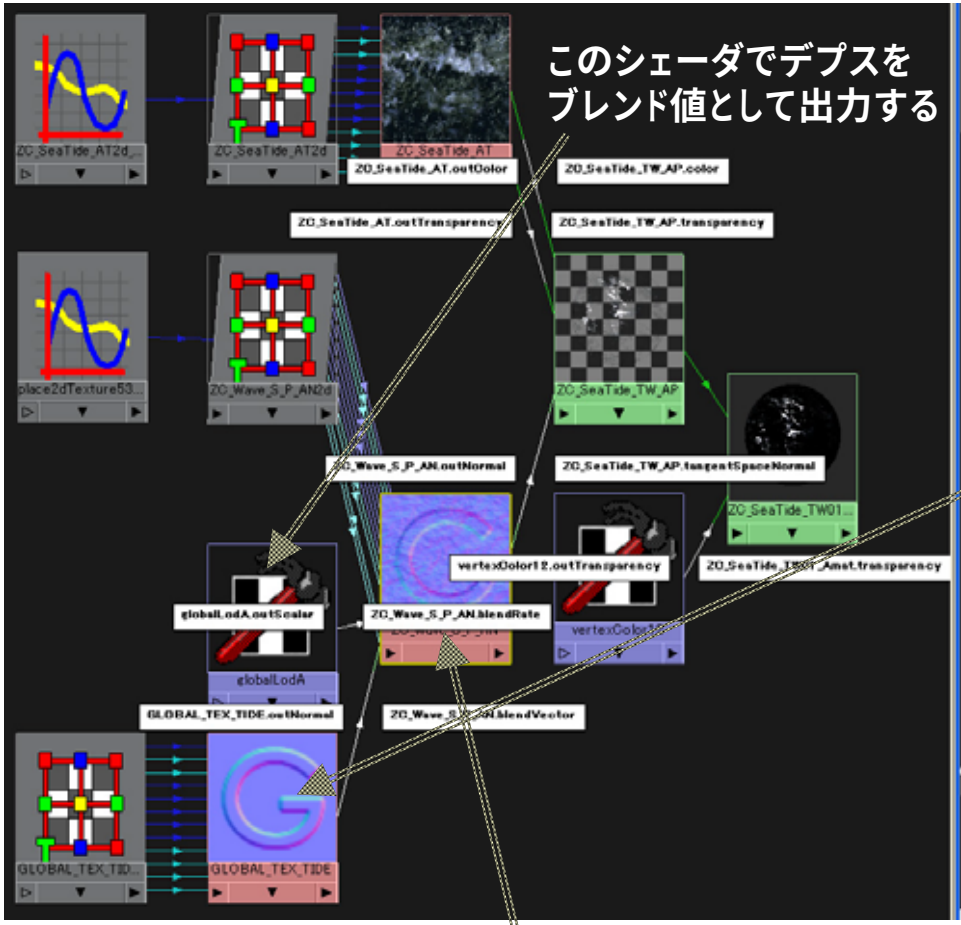


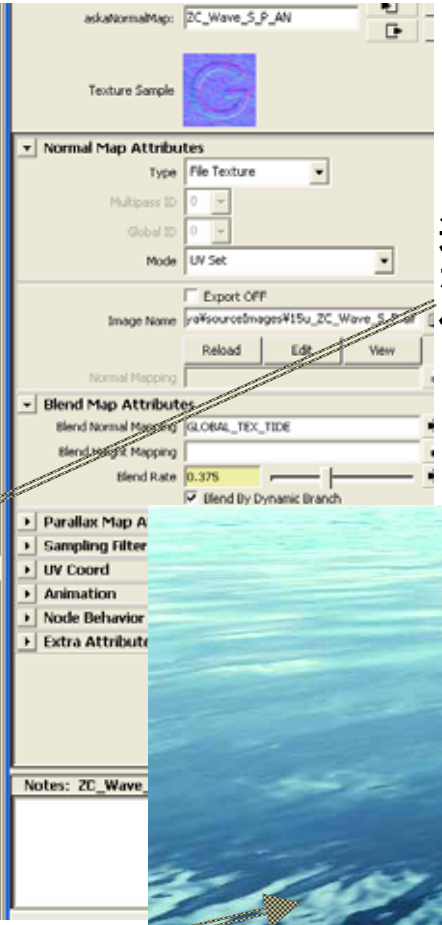
ライティングを行わないシェーダ





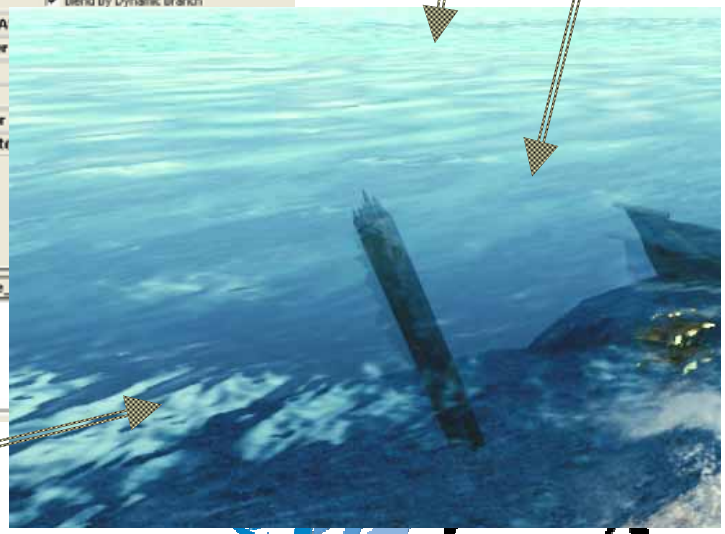
距離で処理を変えるシェーダ





中間ではブレンド

遠景ではダイナミック
なノーマルマップが
使用される



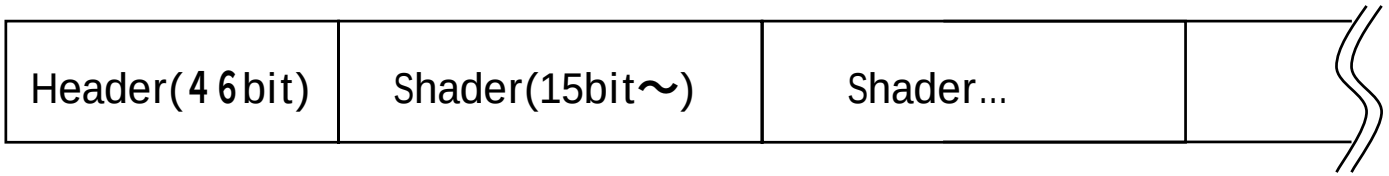
近景ではスタティックな細かいノー
マルマップが使用される

ランタイム実装のキーコンセプト

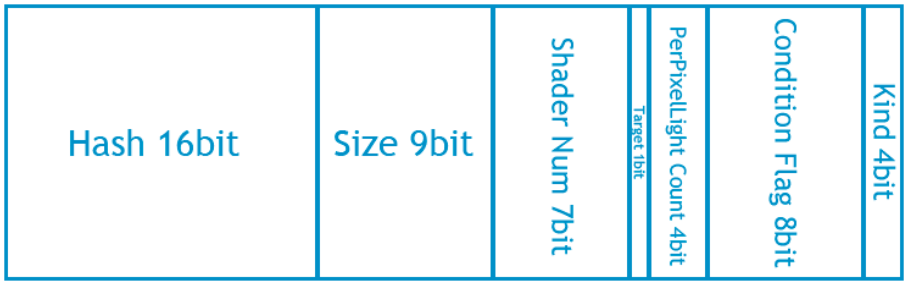
- フレキシブルなシェーダの組合せの実現
  - 可変長ビット列でシェーダの組合せを表現する
    以後これを ”ShaderKey” と呼ぶ
- hlslファイルを適時生成
  - 個々の小さなシェーダソースを統合して
    hlslまたはCgファイルを生成する
- シェーダキャッシュシステム
  - ShaderKeyをキーとしてデータベースよりシェーダ
    インスタンスを生成、取得する

ShaderKey

- 可変長のビット配列
- 大まかにはヘッダとシェーダチャンクからなる
  - ヘッダ
    - ShaderKeyのサイズ
    - シェーダの数
    - など…
  - シェーダチャンク
    - シェーダIDと引数
    - チャンク自体も可変長



ShaderKey:ヘッダ

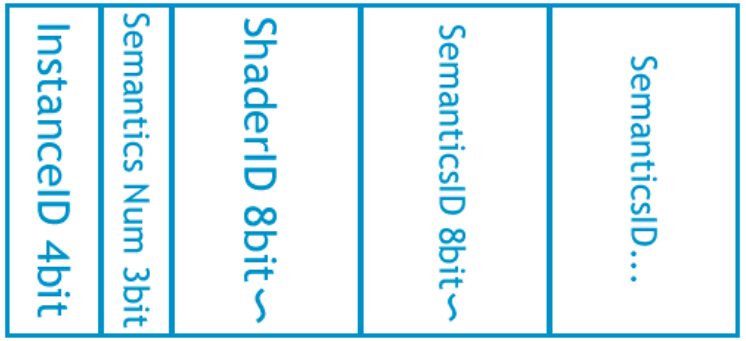


- **より詳細な情報**
  - Hash
    - ShaderKey**バイト列の**CRC
    - **キー検索のハッシュやバリデーションチェックに使用**
  - ShaderKey**サイズ**
  - **シェーダ数**
  - **ターゲット**
    - VertexShader**や** PixelShader、GeometryShader**など**
  - **コンディションフラグ**
    - **法線のありなし、頂点カラーのありなしなど**
    - **主に**VertexShader**と**PixelShader**入出力レジスタのすり合わせに使用**
  - **シェーダ種別**
    - **通常モデル用、パーティクル用、ポストフィルタ用など**

ShaderKey:シェーダチャンク

- ヘッダで宣言された数のチャンクが連続する
- シェーダチャンク情報
  - シェーダID
  - 引数の数
  - 引数(セマンティクスID)
  - インスタンスID

    シェーダIDとセマンティクスIDは可変長



シェーダID

- 個々のシェーダを識別するユニークなID
  - 一つのIDがシェーダソースコードに対応している
    - ツール(Maya)上で見えているノード一つが複数のシェーダIDに分割されることもある

セマンティクスID

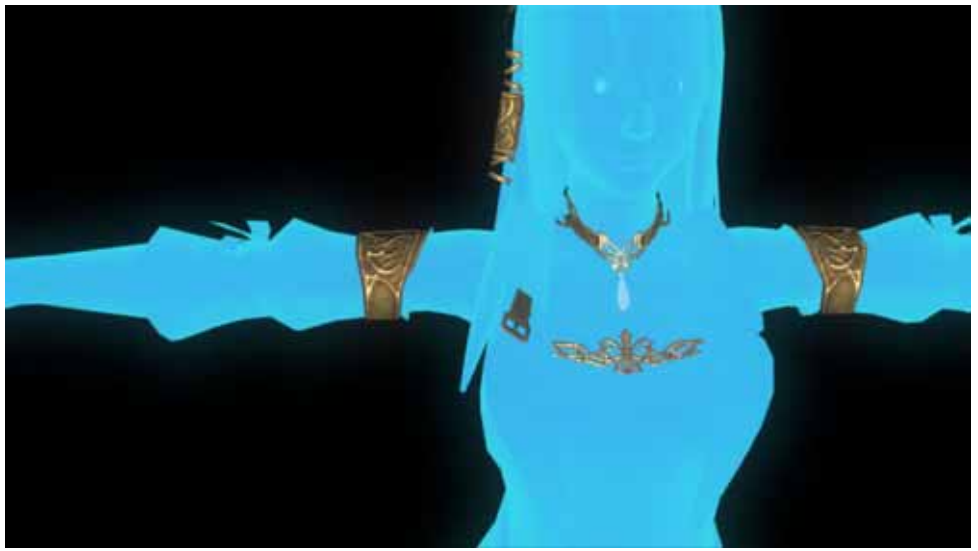
- Pixel Shaderでの変数を識別するID
  - 頂点データ
    - Vertex Shaderから受け渡されるデータの識別
      - Vertex Shaderでデータは処理されることもある
      - 位置ベクター、法線ベクターなど
  - テンポラリレジスタ
    - シェーダノード間で受け渡しされる変数
      - 関数の引数と戻り値
  - システムレジスタ
    - システムが暗黙に計算、設定する変数
      - Eyeベクター、反射ベクター
        - » 毎回作るコードを書いておいても、使わなければコンパイル時の最適化で勝手に消える

インスタンスID

- 同名のShader Constantを区別するID
  - 複数のシェーダ間で同名のシェーダ定数がある場合に割り振られるインデックス
    - 同じシェーダノードを複数使用する場合にも結果的に同じ名前の定数が使われることになる
      - このような場合にもインスタンスIDで区別される

ソースファイルの生成

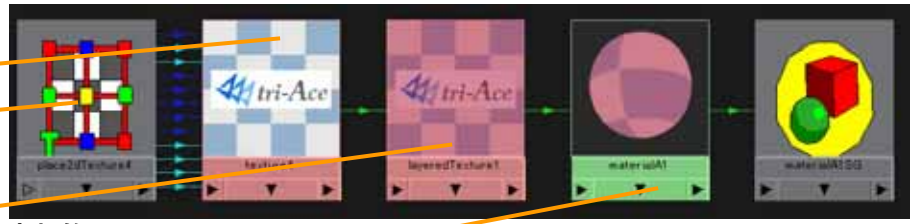
- ランタイムでソースファイルをコンパイル
  - ソースコードはShaderKeyを利用して各シェーダファイルからhlsl(Cg)ファイルとして生成される



具体的なコードは？

- 汎用変数(テンポラリレジスタ)を介して、関数(各シェーダ)の列挙で記述できる形にする
  - 例.テクスチャとカラーを乗算するだけのシェーダコード

```
A = FetchTex2D( stage, uv );
B = ConstColor();
C = Blend( A, B );
FinalCombiner( C );
```



このノードは内部的には2つのノードとして出力される

A,B,Cがテンポラリレジスタ
stageやuvがテクスチャ用セマンティクス

コード生成

- Main.ahsl
  - 最も基本になるソースコード
    - シェーダのエントリポイントを含む
  - 実際のコード生成としては main.ahsl内でincludeされる3つのヘッダファイルを動的に作成する
- 各ヘッダファイル
  - PreCondition.h
    - 主にdefineを記述する
      - 各種機能のenable
  - Function.h
    - 各関数(FectchTex2D()等)の本体を記述する
  - Body.h
    - 各関数のコールを列挙する部分

Main.ahslソースイメージ

```
#include      <PreCondition.h>

float4x4      cmWVS                    : register(c000);
float3x4      cmWorld                  : register(c004);
      ...
float4        cvReserve[96]            : register(c000);

#include      <Functions.h>

PS_OUT  AHSLMainPS( const PIPE Pipe )
{
      PS_OUT psout;
      float4 vPos = Pipe.vPosThru;
      float4 vNormal = float4(0,0,0,0);
      float3 vEyeDir = CalcEyeDirection( vPos );
#ifdef USING_VERTEXNORMAL
      vNormal =  float4(normalize(Pipe.vNormal.xyz),0.0);
#endif
      ...

      #include      <Body.h>

      psout.vCol = vFinalColor;
      return psout;
}
```

システムが提供する定型constant

レジスタの宣言や初期化

defineに従い必要な定数も生成する

ahslファイルサンプル1

- FetchTex2Dシェーダ

#〜#がセマンティクス文字列に置換される

```
#out# = FetchTex2D( #stage#, #uv# )
```

Body.hにコピーされる

```
@pre
@func
#ifndef decl_FetchTex2D
#define decl_FetchTex2D
float4 FetchTex2D( int s, float2 uv )
{
    return tex2D( asTexStage[s], uv );
}
#endif
@end
```

@funcなどがセクション記号

ここはFunction.hにコピーされる

ahslファイルサンプル2

- MakeEyeFresnelシェーダ

```
#fEyeFresnel# = MakeEyeFresnel( vNormal, vEyeDir, #fFresnel# )

@pre
#define USING_VERTEXNORMAL
@func
#ifndef decl_MakeEyeFresnel
#define decl_MakeEyeFresnel
float MakeEyeFresnel( float3 vNormal, float3 vEye, float coef )
{
    float f = pow( 1.0-saturate(dot(vNormal, vEye)), 5.0 );
    return lerp( 1.0, f, coef );
}
#endif
@end
```

PreCondition.hにコピーされる

ahslファイルサンプル3

- ConstColorシェーダ

$はインスタンスIDに置換される

```
#dst# = eConstColor_Color_Color$

@pre
@func
float4        eConstColor_Color_Color$;
@end
```

ConstantColorなど同時に複数使われるようなシェーダは
インスタンスIDによってShaderConstantを管理する

ShaderKeyの生成

- Body.hをIDリスト(enum)に従ってShaderKeyに変換する

```
A = FetchTex2D( stage, uv );
B = ConstColor();
C = Multiply( A, B );
FinalCombiner( C );
```

このコードは、以下の様な5つのチャンクのShaderKeyになる

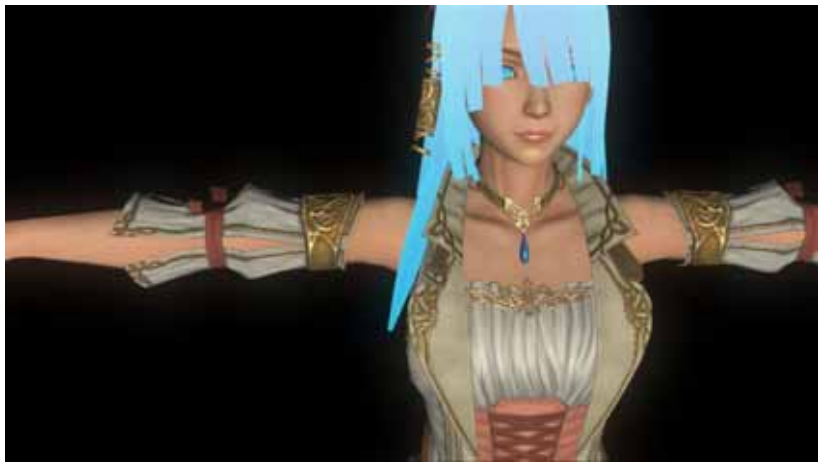
Header:4	FT:A,stage,uv	CC: B	M: C,A,B	FC: C

ShaderImmediateConstant

- **高速化のために一定の値を定数化**
  - **特定の**constant**ベクトルを**constant register**を使わずに即値としてソースを生成**
    - (0,0,0,0)
    - (0,0,0,1)
    - (1,1,1,0)
    - (1,1,1,1)

シェーダキャッシュシステム

- コンパイルされたシェーダをキャッシュファイルとして保存する
  - シェーダインスタンスの生成、管理
  - シェーダのコンパイル、記録
    - ShaderKeyをキーとする



シェーダキャッシュ

- コンパイルされたシェーダは開発機HDDのシェーダキャッシュファイルに記録される
  - キャッシュの要素
    - ShaderKey
    - コンスタントテーブル情報
    - シェーダオブジェクトバイナリ
  - このファイルは最終的にゲームで使われる全てのシェーダの組合せを格納する事が前提

シェーダデータベース

- ランタイムのシェーダを管理する
  - Classとして実装
  - HDD上のキャッシュファイルからのロード
    - 開発ビルドでは書き込みも行う
  - ShaderKeyによるクエリーに対して
    シェーダのインスタンス生成、管理を行う

シェーダプロファイルデータ

- 開発補助用のデータ
  - キャッシュファイルとは別のファイルに記録する
  - 各ShaderKeyに対応する各種情報を保持
    - 累計使用時間、最終使用日時
    - クエリー回数、生成日時
    - ビルドした開発機名
    - リージョン情報
    - マージ回数
      - など…

No.	生成回数	アクティブ時…	実効…	キャッ…	VS…	シェ…	キー…	マ…	作成者	作成日時	最終アクセス時間	ライ…	条件	即…	アダ…	主要シェーダ
173	146444	585928	0	2952	PS	Obj	157	37	takejiro	2008/04/25 1…	2008/05/09 1…	2	0	4	0	FetchTex2D/Fet
174	146378	585724	0	2836	PS	Obj	142	37	takejiro	2008/04/25 1…	2008/05/09 1…	2	0	4	0	FetchTex2D/Fet
175	771505	3089321	0	2416	PS	Obj	140	37	takejiro	2008/04/25 1…	2008/05/09 1…	2	0	5	0	FetchTex2D/Fet
176	146780	587390	0	2840	PS	Obj	146	37	takejiro	2008/04/25 1…	2008/05/09 1…	2	0	5	0	FetchTex2D/Fet
177	149788	599342	0	2324	PS	Obj	141	37	takejiro	2008/04/25 1…	2008/05/09 1…	2	0	6	0	FetchTex2D/Fet

その他

- Version管理
  - Versionが更新されるとシェーダキャッシュの自動リビルドを行う
    - 更新できない場合は最初から作り直し
- ShaderNode
  - ShaderKeyをラップする抽象レイヤー
    - Struct ShaderNodeの片方向リンク構造
  - Maya ShadingNodeからの中間形式的な位置付け
    - マテリアルデータのエクスポートもこの形式
  - 動的シャドウプロジェクターなどの後付もこのレイヤーで行う
    - ゲームプログラマによるフルスクラッチシェーダ
    - 既存のノードへのモディファイア

シェーダ管理の問題

- この実装はシェーダの生成が必要
  - 基本は実行時にコンパイル
    - 生成したシェーダはキャッシュファイルに保存
    - バージョンアップ時はリビルド
      - またはキャッシュファイルを削除
  - キャッシュファイルの生成およびマネージメントが問題点としてエンジン設計時点から予想されていた
    - 実際にマスターアップが近づいてきて問題が顕在化

キャッシュサイズの問題

- リリース時にはシェーダをコンパイルしない
  - 基本的にシェーダバイナリはQA時に生成することが前提
    - 設計的にはリリース時でもシェーダコンパイルは可能
      - しかし見苦しいので実際には採用されていない
  - マスターアップが近づいてくると実際のキャッシュサイズが増大した
    - 最初は10Mぐらいを予想していたがそれを超える雰囲気だった

キャッシュサイズ対策 – 圧縮

- キャッシュファイルを圧縮
  - ファイルサイズ問題に対応するためメモリ上のキャッシュデータを辞書式で圧縮して保存するようにした
    - 23%ほどデータが小さくなった
  - 使用するときはこれを展開しながら利用
    - マルチスレッド化していたのでパフォーマンス的な問題は発生しなかった

さらなるファイルの増大

- 圧縮によりファイルサイズは減ったが
  - 実際に通しプレイなどをするとキャッシュサイズはさらに増加
    - この時点ですでに20Mを楽に超えていた
      - 最終的な(メモリ上の)キャッシュサイズを20M強に設定
      - この時点でもすべてのシチュエーションを網羅したわけではない
        - » さらに増大されることが予想されていた

キャッシュシステムの拡張

- キャッシュシステムをL1/L2キャッシュに分離
  - ここでいわゆるキャッシュ的なアルゴリズムを導入
    - L1はシェーダのインスタンス
      - 展開されCreateXXXShader()された状態
    - L2は圧縮された状態のデータ
  - これにより、展開された状態のバッファを最小限にすることが可能になった

キャッシュエラーの検出

- 壊れたキャッシュファイルが散見してきた
  - キャッシュファイルを本格的生成するように
    - バグなどでキャッシュファイル書き出し中に停止
    - 壊れたキャッシュファイルが出来てしまう
  - CRCなどを導入して壊れたキャッシュエントリを検出するようにした
    - 壊れたエントリは検出して破棄

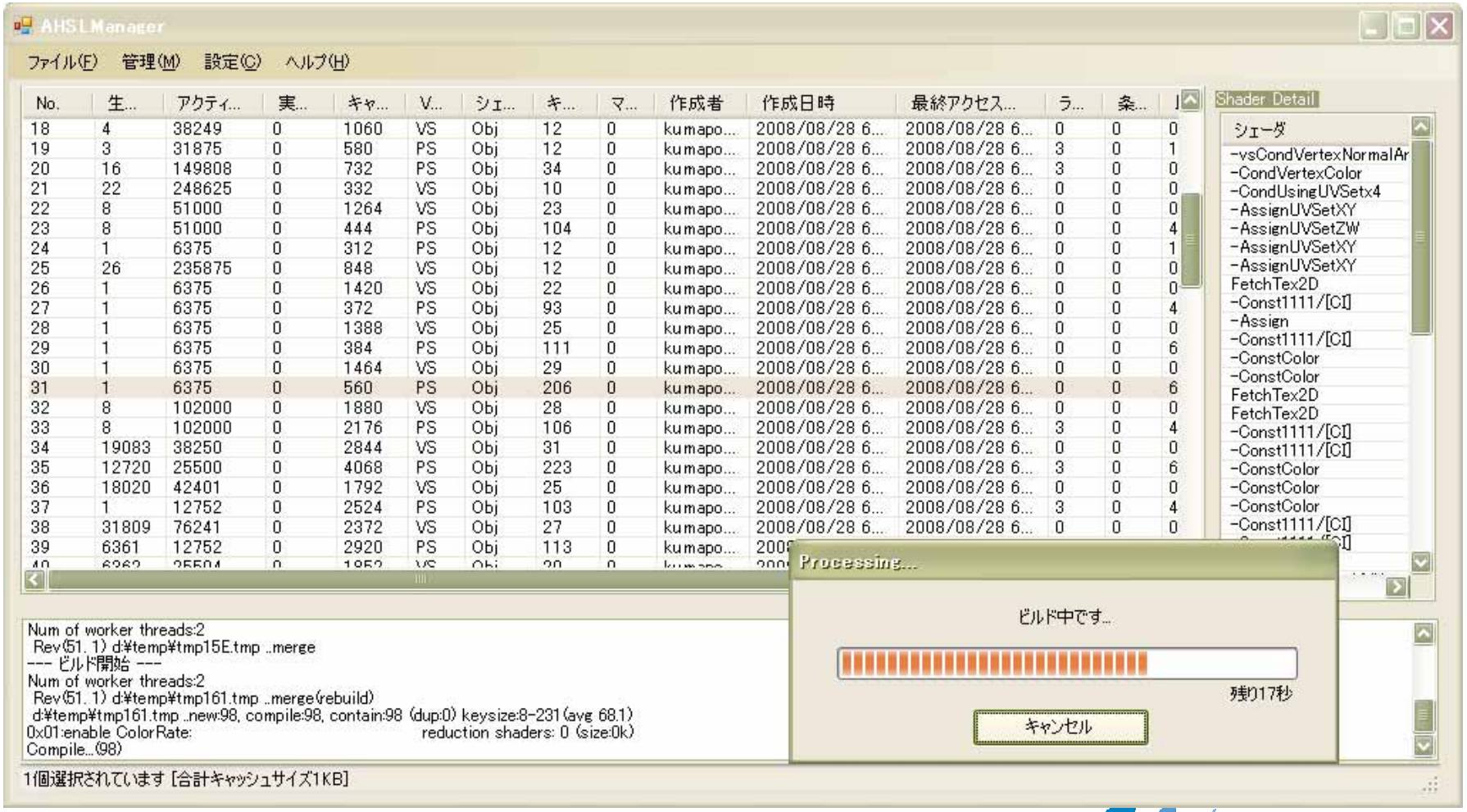
オフラインコンパイルのサポート

- キャッシュサイズの増大によりキャッシュ生成の時間が問題に
  - AHSLManagerというWindows上のツールでコンパイルをサポート
    - PCの性能に応じてコンパイル速度が数倍以上高速化

AHSLManager

- シェーダキャッシュファイルおよびプロファイルデータを管理、生成するツール
  - Windowsツール
  - シェーダキャッシュの管理を行う
    - シェーダのコンパイルが可能
    - 複数の開発機に対してダウンロード、アップロードができる
    - キャッシュファイルのマージや分割も行う
  - プロファイル情報の表示

AHSLManager



圧縮の効率化

- データサイズがさらに増加
  - 圧縮を効率化するために初期辞書を追加
    - さらに15%ほど圧縮率が増加
  - しかしそれでもキャッシュサイズは増加し続けた

キャッシュ分割対応

- これまで一つだったキャッシュファイルを複数に分割できるようにした
  - リージョンという概念を導入
    - 複数のリージョンを選択して使用する
    - 生成時にリージョンを指定して記録する
  - 同時にファイルへの書き出しのサイズ制限も撤廃
    - 開発時にはファイルをL3キャッシュとして扱う
    - メモリ上にキャッシュエントリが無い場合はそのエントリをファイルからロードするように変更
      - 開発時には若干の遅延が発生する

リージョン

- キャッシュの分割のための付加情報
  - ゲームプロジェクト側で設定する

Area1 Region:1

Area2 Region:2

共通キャッシュ
Region:1&2

Area1用キャッシュ
Region:1

Area2用キャッシュ
Region:2

効率とサイズの選択

- それでもサイズはまだまだ増大
  - レンダリングパフォーマンスとキャッシュサイズの調整をプロジェクトでできるようにした
    - ColorRateシェーダをエイリアス化
      - この値が1.0かそれ以外でシェーダを分けるかどうか?
      - 有意なパフォーマンス低下は見られなかった
        - » デフォルトでON
    - PerPixelLightカウントをエイリアス化
      - PerPixelLightの最大数に応じてシェーダを分けるかどうか?
      - 2-3%程度のパフォーマンス低下が見られた
    - Shadowのdensity設定をエイリアス化
      - この値が1.0かそれ以外でシェーダを分けるかどうか?
      - 2-3%程度のパフォーマンス低下が見られた

それでもシェーダは増える

- しかし健闘空しくキャッシュサイズは50Mを超える
  - 30,000を超えるキャッシュエントリ数
  - キャッシュファイルを分けて対応したがそれでも規定サイズを超えていた
    - キャッシュの内容を分析

ShaderAdaptorの問題

- ShaderAdaptorによるバリエーションが多数をしめる
  - ShaderAdaptorとはランタイムでマテリアルに追加されるタイプのシェーダ
    - Shadow, Projectorなど…
    - これによるシェーダバリエーションが8割ぐらいを占めていた
      - 特にShadow
      - 一つのオブジェクトに5つのシャドウマップがアサインされていることもあった

ShaderAdaptorの問題

5つものアダプタを使用して
いるエントリがたくさん

このシェーダではすべての
アダプタがシャドウマップ

AHSLManager

ファイル(F)　管理(M)　設定(C)　ヘルプ(H)

No.	生成...	アクティブ...	実効...	キャッ...	VS...	シェ...	キー...	マ...	作成者	作成日時	最終アクセス時間	ライ...	条件	即...	アダ...	主要シェーダ
1451	218	69724	0	3296	PS	Obj	97	34	90	2008/04/08 1...	2008/05/09 1...	3	0	1	5	DiffLambertShade
3463	169	4020	0	3928	PS	Obj	170	36	86	2008/04/04 1...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
3466	716	103280	0	3928	PS	Obj	170	36	86	2008/04/04 1...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
3467	1057	356584	0	3880	PS	Obj	170	37	67	2008/04/04 1...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
3472	703	77561	0	3928	PS	Obj	170	36	86	2008/04/04 1...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
5536	32	2874	0	3996	PS	Obj	156	35	86	2005/11/22 2...	2008/04/25 1...	3	0	3	5	FetchTex2D/Diff
6626	17	2119	0	4184	PS	Obj	164	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	3	5	FetchTex2D/Fet
6629	6	1426	0	3884	PS	Obj	163	23	takejiro...	2000/00/00 0...	2008/04/28 0...	3	0	3	5	FetchTex2D/Fet
6638	11	1450	0	4172	PS	Obj	164	23	takejiro...	2000/00/00 0...	2008/04/28 0...	3	0	3	5	FetchTex2D/Fet
6674	3	10	0	3996	PS	Obj	179	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	5	5	DiffLambertShade
6675	5	20	0	4048	PS	Obj	198	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	5	5	FetchTex2D/Fet
6682	17	80	0	4060	PS	Obj	197	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	5	5	FetchTex2D/Fet
6683	7	30	0	4044	PS	Obj	184	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	3	5	FetchTex2D/Fet
6684	3	10	0	3876	PS	Obj	180	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	6	5	FetchTex2D/Diff
6685	113	30192	0	4284	PS	Obj	198	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	5	5	FetchTex2D/Fet
6686	43	10742	0	4284	PS	Obj	185	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	3	5	FetchTex2D/Fet
6689	3	1304	0	4212	PS	Obj	180	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	5	5	DiffLambertShade
6697	29	7216	0	4248	PS	Obj	199	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	5	5	FetchTex2D/Fet
6707	15	3194	0	4092	PS	Obj	181	23	takejiro...	2000/00/00 0...	2008/04/30 2...	3	0	6	5	FetchTex2D/Diff
6745	3602	289506	0	3860	PS	Obj	185	31	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
6746	1776	86860	0	4080	PS	Obj	224	31	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	6	5	FetchTex2D/Fet
6750	1812	120312	0	3836	PS	Obj	185	30	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
6755	1668	394526	0	3812	PS	Obj	185	31	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
6756	698	214250	0	4044	PS	Obj	224	31	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	6	5	FetchTex2D/Fet
6790	875	38044	0	3956	PS	Obj	185	28	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
6791	907	59094	0	3852	PS	Obj	189	31	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
6820	488	16696	0	3852	PS	Obj	189	31	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
6821	340	14300	0	4020	PS	Obj	224	31	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	6	5	FetchTex2D/Fet
6824	164	8088	0	3960	PS	Obj	189	28	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
6825	212	7602	0	4200	PS	Obj	224	28	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	6	5	FetchTex2D/Fet
6829	162	30060	0	3816	PS	Obj	189	27	takejiro...	2000/00/00 0...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
7472	14	4314	0	4996	PS	Obj	208	36	77	2005/11/22 2...	2008/05/09 1...	3	0	5	5	FetchTex2D/Fet
8416	4	800	0	4160	PS	Obj	164	23	takejiro...	2008/04/28 1...	2008/04/30 2...	3	0	3	5	FetchTex2D/Fet
9934	55	5031	0	4168	PS	Obj	235	32	takejiro...	2008/04/28 1...	2008/05/09 2...	3	1	8	5	FetchTex2D/Fet
10457	1	0	0	3920	PS	Obj	163	21	takejiro	2008/04/28 1...	2008/04/28 1...	3	0	3	5	FetchTex2D/Fet
10458	1	0	0	3920	PS	Obj	176	21	takejiro	2008/04/28 1...	2008/04/28 1...	3	0	5	5	FetchTex2D/Fet
10459	1	0	0	4180	PS	Obj	215	21	takejiro	2008/04/28 1...	2008/04/28 1...	3	0	6	5	FetchTex2D/Fet
10460	1	0	0	4108	PS	Obj	228	21	takejiro	2008/04/28 1...	2008/04/28 1...	3	0	6	5	FetchTex2D/Fet
10616	779	277897	0	3432	PS	Obj	185	35	86	2008/04/18 1...	2008/05/09 1...	3	1	6	5	FetchTex2D/Fet
10619	363	119173	0	3428	PS	Obj	194	35	86	2008/04/18 1...	2008/05/09 1...	3	1	8	5	FetchTex2D/Fet

Shader Detail

シェーダ
-vsCondVertexNormalAndBinormal
-CondExVertexColor
-CondUsingUVSetx4
-CondVertexLighting
-ShadowDepthProjector/[ADP]
-AssignRtoRGBA
-MultiplyG
-ShadowDepthProjector/[ADP]
-AssignRtoRGBA
-MultiplyG
-ShadowDepthProjector/[ADP]
-AssignRtoRGBA
-MultiplyG
-ShadowDepthProjector/[ADP]
-AssignRtoRGBA
-MultiplyG
-ShadowDepthProjector/[ADP]
-AssignRtoRGBA
-MultiplyR
-AssignUVSetXY
-AssignUVSetXY
-AssignUVSetXY
FetchTex2D
-Const1111/[C1]
FetchTex2D
-Const1111/[C1]
-Const1111/[C1]
-Const1111/[C1]
FetchTex2D
-Const1111/[C1]
FetchTexDualParaboloid
-Const1111/[C1]
-SetNormal
-AssignRtoRGBA
-LightMaskMixer
-AssignGtoRGBA
-LightMaskMixer
-PackVector
-AssignAtoRGBA
-AssignAtoRGBA
DiffLambertShader
-AssignAtoRGBA

ShaderAdaptor制限

- 強制的にShaderAdaptor数(shadow)を制限できるように
  - 生成時やツール上でShaderAdaptor数を制限できるように
    - これによりシェーダファイルサイズを大幅に減らすことが可能に
      - 一部ではビジュアルに問題が出てしまったので手動での調整を必要とした

未生成シェーダへの対応

- ShaderAdaptorなどバリエーションがあるシェーダが未生成の場合
  - リリース時にはバリエーションの基底となるシェーダを利用するように対応
    - 少しでも表示がおかしくならないように

キャッシュの生成

- キャッシュの生成はQAチームで行っていた
  - ある程度シェーダに関わる仕様やリソースが固まった時点でシェーダの生成を行った
    - デバッグ機能を使用
    - 実際にゲームをプレイして生成
    - 複数の担当者で作ったファイルをツールでマージ
  - この生成が予想以上に大変だった
    - 数十人のスタッフで2週間以上
    - また最初は試行錯誤が多く何度も作り直した

QAの誤算

- 生成にかかる期間の見積もりができていなかった
  - 予想外の手間
    - ある程度の問題点は想定していたが予想外の問題も発生した
  - ゲームやリソースの作り方にもよる
    - 今後はこの点にも留意してリソースを作成する必要がある

最終的なデータ

- ある作品ではファイルは4つに分割した

領域名	サイズ	エントリ数
Boot	18k	26
Common	9.8M	9158
Area1	9.0M	6804
Area2	14.8M	11120

その他のデータ

- 別のプロジェクトでの現時点でのデータ

領域名	サイズ	エントリ数
Boot	25k	26
Area1	29.4M	22822
Area2	30.2M	23842
Area3	33.2M	25799

キャッシュ増大要因のまとめ

- 初期プロジェクトはエンジン開発とゲーム開発が同時進行
  - ShaderAdaptorによるバリエーション
    - たとえばShadowの方式もたくさん実装された
  - ShaderImmediateConstantによるバリエーション
    - 対策を検討はしていた
      - 今回は対策を行わなかった
  - 新しい機能が実装されることによるシェーダ数の増加
    - 初期のデータと末期のデータで使用しているシェーダが異なる
    - デザイナーにとっても初めてのシェーダシステムでいろいろ試行錯誤した結果

まとめ - デメリット

- キャッシュの生成コスト
  - 生成自動化に限界がある
  - ファイルサイズの問題
    - 現時点でもシェーダバイナリとしては大きすぎる
      - 次世代では問題ないかも?
  - 結果的にリソース作成時にもある程度意識する必要
    - もうちょっとエントリ数を整理したい
- シェーダを作る難易度
  - ある程度デザイナーがシェーダの仕組みを知らなければいけない

まとめ - メリット

- 自由度の高さ
  - シェーダノードを作ればプログラマに頼らずともいろいろなシェーダをデザイナーだけで作ることが可能
    - へんな組合わせのシェーダを作ることも出来る
- パフォーマンスの最適化
  - ShaderImmediateConstant機能など
  - シェーダコンパイラの性能を生かして最適なシェーダを自動生成できる

今後の課題

- シェーダ生成の問題解決
  - シェーダ生成自動化を推し進める
    - オフラインである程度予想してシェーダを作成しておく?
    - ランタイムでのシーン情報やデータベースを元にシェーダを作成?
  - シェーダ生成の高速化
    - とくにリビルド
    - 分散シェーダ生成
  - エントリ数の削減
  - GPU側でなんか対応?

質問は?

- スライドは以下のページでもダウンロードできます
  - http://research.tri-ace.com/