要旨

- 動機
- 細分割曲面
  (Subdivision Surfaces)
  - Control pointの計算
  - Surfaceの生成
- ディスプレースメントマッピング
  (Displacement Mapping)
- コンテンツ作成

動機

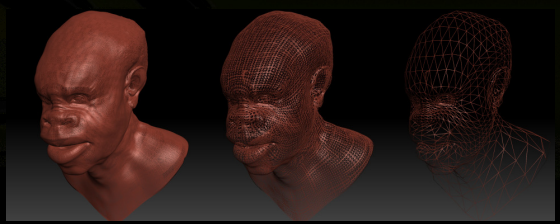
圧縮

- メモリ使用量とハンド幅の軽減
  - ハイポリモデルのレンダリングにおいて最もボトルネックになるのはメモリである

	Level 8	Level 16	Level 32	Level 64
標準トライアングルメッシュ	16MB	59MB	236MB	943MB
変位細分割曲面	1.9MB	7.5MB	30MB	118MB

スケーラビリティ

- 連続的LoD (Level of Detail)



アニメーションとシミュレーション

- 処理負荷の高い計算の頻度を減らす:

リアルなアニメーション:
ブレンドシェープ、 モーフィング**など.**

Physics、衝突判定、軟体力学計算など.

目標

- 今までにない映像を可能にする:
  - ハイポリモデル・キャラクター
  - リアルなアニメーション

細分割曲面 (Subdivision Surfaces)

細分割曲面

- 簡単にモデリングでき、柔軟なアニメーションができる
- 映画産業において幅広く使われている
- モデリングツールやスカルピングツールが使える

GPUでの実装

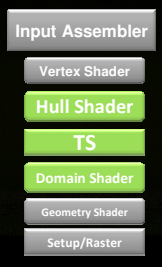
- 以前のGPUにおけるアプローチ:
  - "Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping", Michael Bunnell
  - ジオメトリシェーダーでの再帰的微細化
- 再帰的計算だった→ 直接計算が好ましい

Tessellationパイプライン

- Direct3D11はDirect3D10に対してプログラマブルテセレーションのサポート拡張をしている
- 二つの新しいシェーダー:
  - Hullシェーダ (HS)
  - Domainシェーダ (DS)
- 一つの固定機能ユニット:
  - Tessellator (TS)

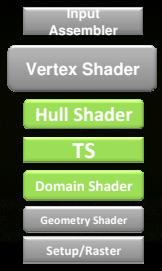
インプット・アセンブラ

- 新しいプリミティブタイプ : Patch
  - 任意の頂点数
  - 暗黙のトポロジーなし
  - Tessellationがイネーブルされている時のみ使うことができる



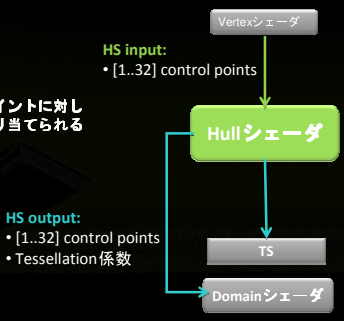
Vertexシェーダ

- パッチコントロールポイントの変換
- 通常以下の目的で使用:
  - アニメーション(スキニング、ブレンドシェイプ)
  - 物理演算
- より少ない頻度での高負荷なアニメーションを可能にする



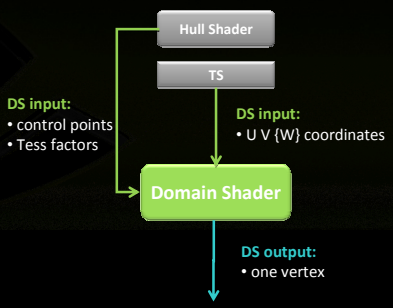
Hull Shader (HS)

- パッチ毎に起動
- 明示的な並列化
  - 一つのコントロールポイントに対して一つのスレッドが割り当てられる



Domain Shader (DS)

- 生成されるバーテックス毎に起動



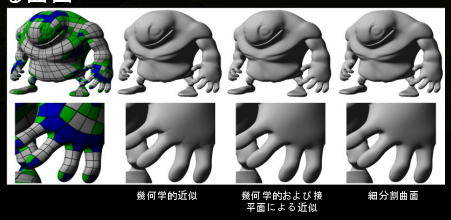
細分割曲面の計算

- Jos Stam: "Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values"
  - 非常に多くの頂点が必要
  - 計算コストが非常に高い
- Jeff Bolz and Peter Schroeder: "Evaluation of Subdivision Surfaces on Programmable Graphics Hardware"
  - 各構造およびテセレーションレベルに対して事前計算が必要

近似Catmull-Clark細分割曲面

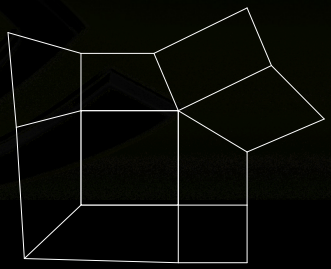
(Approximating Catmull-Clark Subdivision Surfaces - ACC)

- Charles LoopとScott Shaeferによって考案:
  http://research.microsoft.com/~cloop/
- ベジェパッチと独立タンジェントパッチで近似される曲面



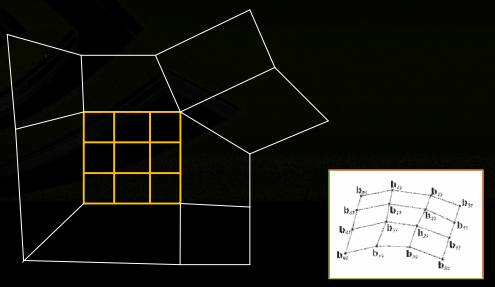
Control Point の計算

- HS への入力は面とその近傍:



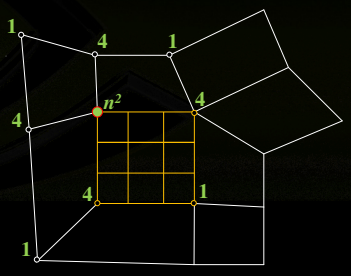
Control Point の計算

- HSからの出力は正則双三次ベジェパッチ:



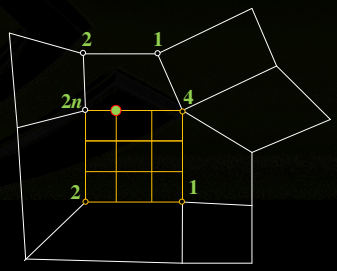
Control Point の計算

- 各コントロールポイントは近傍頂点の線形結合:



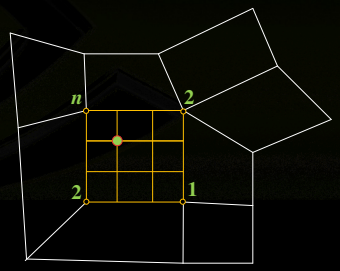
Control Point の計算

- 各コントロールポイントは近傍頂点の線形結合:



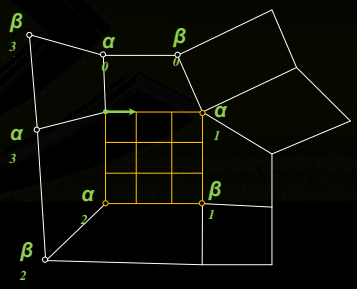
Control Point の計算

- 各コントロールポイントは近傍頂点の線形結合:



Control Point の計算

- コントロール接線も同様に計算できる:



Control Point の計算

- 全てのコントロールポイントは**加重和**: P_j = Sum(W_{ij} * V_i)で計算できる
- HSシェーダにおいてはコントロールポイント毎に１スレッドを割り当てて計算できる:

```
global float w[K][16];      } トポロジーコンビネーション係数
in float3 V[K];             } 入力頂点
out float3 pos[16];         } 出力コントロールポイント

void main() {
     float3 p = 0.0;
     for (int i = 0; i < K; i++) {
         p += V[i] * w[i][threadID];
     }
     pos[threadID] = p;
}
```

各入力頂点V_i対して

Control Point の計算

- **事前計算 を各トポロジーコンビネーションについて行っておく**
- **各コンビネーションはそれぞれ別のパスでレンダリングされる:**
  - **各トポロジーはそれぞれ違った頂点数で構成されている**
  - **ウェイト値のストア場所は共通の定数バッファである**
- **定数の総量はトポロジーコンビネーションの数に比例する**
- **トポロジーコンビネーションの数を最小限にする事が重要**

一貫性のあるコントロールポイント計算

- 共有されるコントロールポイントはメッシュ上の穴を回避するために一貫性のある計算が必要
  - 加算項は全て同じ次数で計算する
  - インデックス配列を使う

一貫性のあるコントロールポイント計算

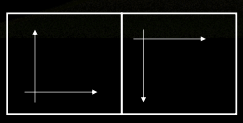
```
uniform int vertexIndex[K];
global float w[K][16];
in float3 V[K];
out float3 pos[16];

void main() {
     float3 p = 0.0;
     for (int i = 0; i < K; i++) {
         int idx = vertexIndex[i];
         p += V[i] * w[idx][ThreadID];
     }
     pos[ThreadID] = p;
}
```

浮動小数点の一貫性

- **浮動小数点数加算は可換でない:**
  - **A + B + C + D != D + C + B + A**
  - **(A + B) + (C + D) == (D + C) + (B + A)**
- **FMA – fused multiply add (~= shader mad)**
  - **FMA(A,B,C) -> A+B*C**
  - 分離した計算式(MUL + ADD)とは同等ではない
    - **A*a + B*b → FMA(A*a, B, b) != FMA(B*b, A, a)**
- **コンパイラの最適化に注意する**
  - **“precise”キーワードを使う**

曲面の生成

- ドメインシェーダでは双三次ベジェパッチと関連する接平面を計算できる
- 隙間のない(watertight) 結果を得ることに特に注意する必要がある（亀裂防止）
- 全ての計算はパッチエッジに沿って対称性を有する必要がある

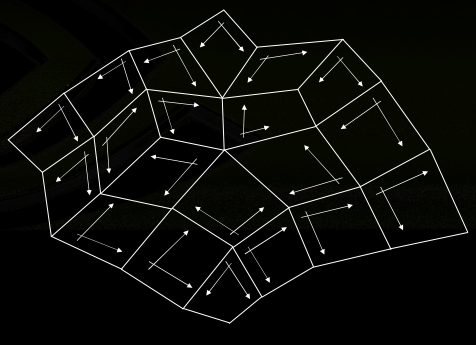


曲面の生成

- 二つの解決方法
  - 全ての面を一貫した同一の方向性を持つように並び替える
  - 境界において対称性のある演算をする

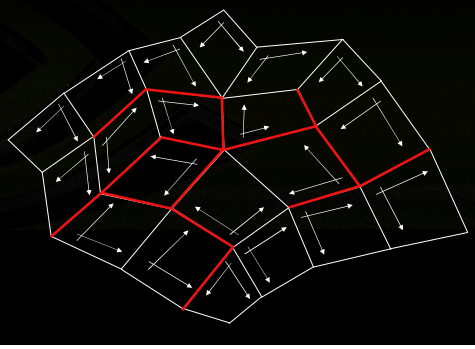
曲面の生成

- 各面は任意の方向性を持つのが一般的



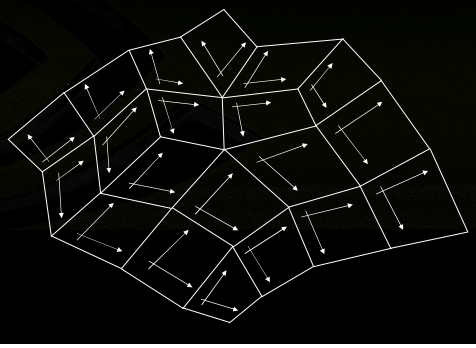
曲面の生成

- 共有エッジが反対方向を向いている



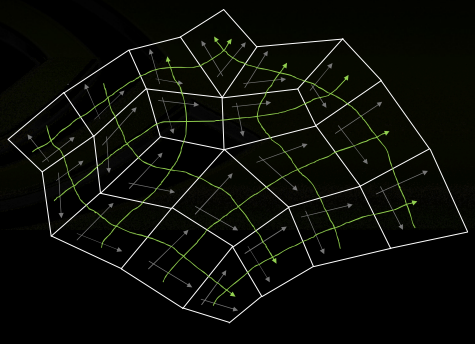
曲面の生成

- パッチを近接一貫性を持たせるために再整列



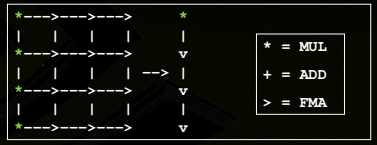
曲面の生成

- 下図は貪欲アルゴリズム



ベジェパッチの計算

- 非対称アプローチ:



- 計算順序はパラメータの並びに依存
- 60 Instructions

ベジェパッチの計算

- 非対称アプローチ:

- 計算順序はパラメータの並びに依存
- 60 Instructions

ベジェパッチの計算

- 非対称アプローチ:

- 計算順序はパラメータの並びに依存
- 60 Instructions

ベジェパッチの計算

- 非対称アプローチ:

- 計算順序はパラメータの並びに依存
- 60 Instructions

ベジェパッチの計算

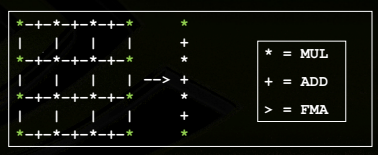
```
float3 bezierPosition(float2 uv, float3 p[16])
{
  float2 B0 =     (1 - uv) * (1 - uv) * (1 - uv);
  float2 B1 = 3 * (1 - uv) * (1 - uv) * (    uv);
  float2 B2 = 3 * (    uv) * (    uv) * (1 - uv);
  float2 B3 =     (    uv) * (    uv) * (    uv);

  return
    (B0.x * p[ 0] + B1.x * p[ 1] + B2.x * p[ 2] + B3.x * p[ 3]) * B0.y +
    (B0.x * p[ 4] + B1.x * p[ 5] + B2.x * p[ 6] + B3.x * p[ 7]) * B1.y +
    (B0.x * p[ 8] + B1.x * p[ 9] + B2.x * p[10] + B3.x * p[11]) * B2.y +
    (B0.x * p[12] + B1.x * p[13] + B2.x * p[14] + B3.x * p[15]) * B3.y;
}
```

ベジェパッチの計算

- このアプローチはクアッドメッシュのみに適用できる
- トポロジーを最小化するには面の方向性を制約
- 対称アプローチはより柔軟

対称性のあるベジェパッチの計算

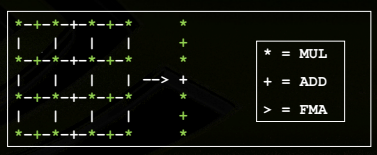
- FMAを使わない計算方法



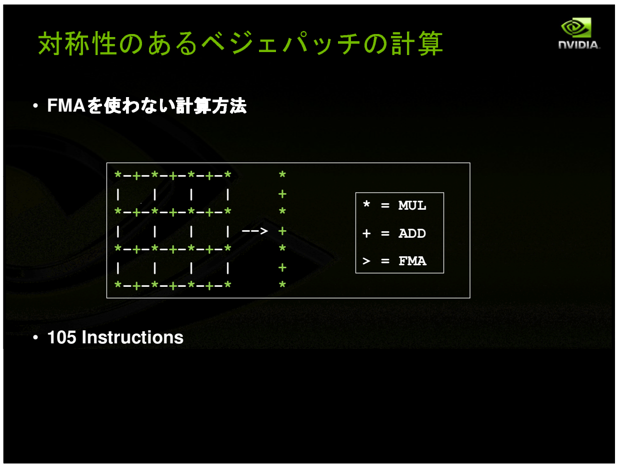
- 105 Instructions

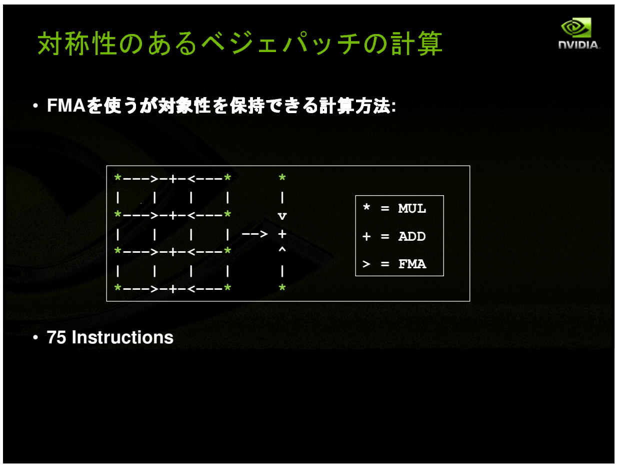
対称性のあるベジェパッチの計算

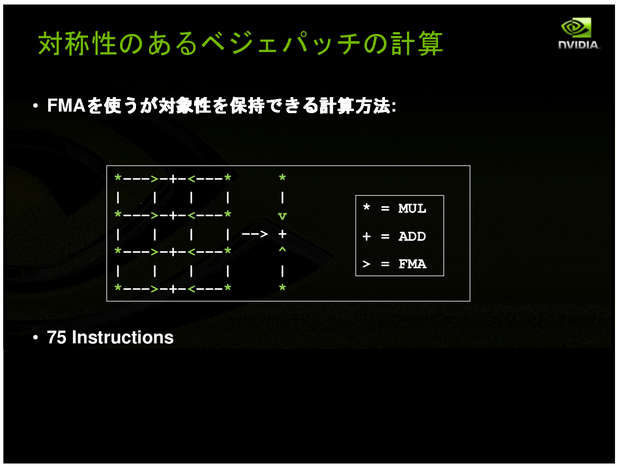
- FMAを使わない計算方法

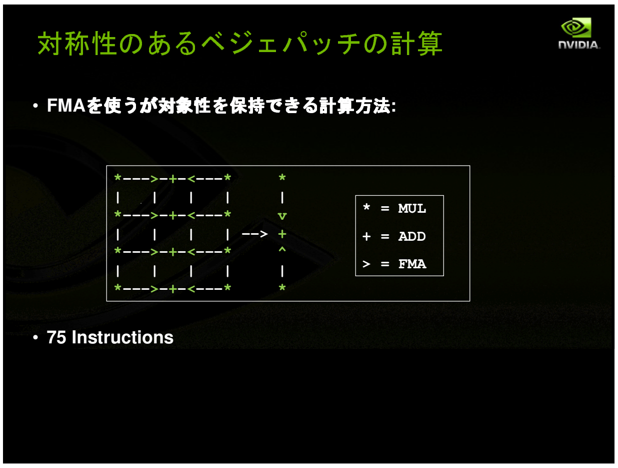


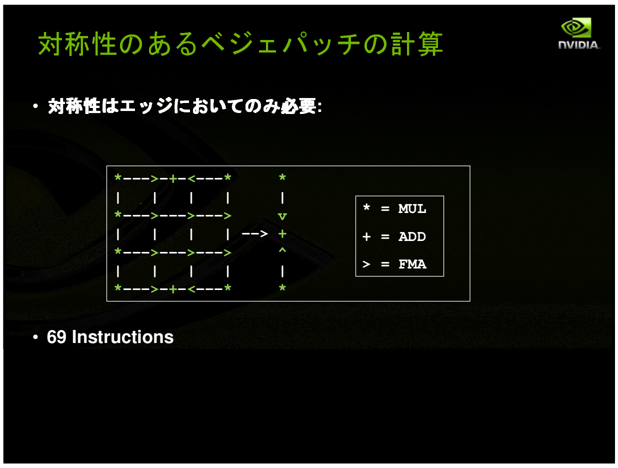
- 105 Instructions

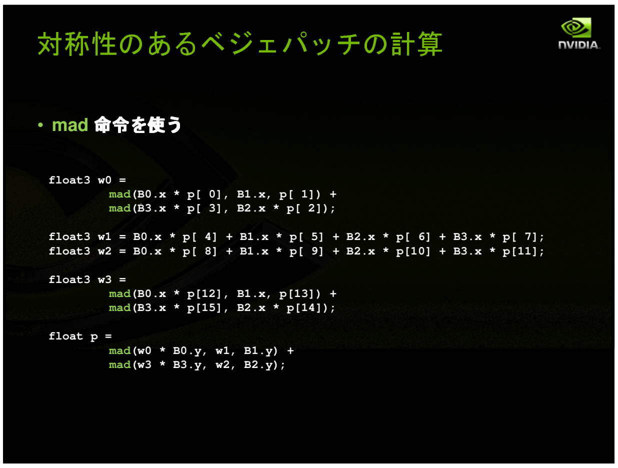












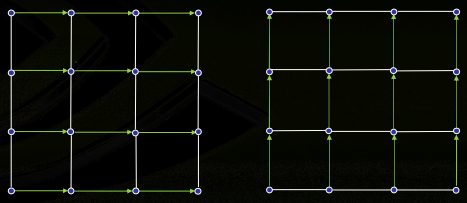
対称性のあるベジェパッチの計算

- precise キーワードをコンパイラ最適化による計算順序変更をなくすために使用する

```
[precise] float3 w0 =
    mad(B0.x * p[ 0], B1.x, p[ 1]) +
    mad(B3.x * p[ 3], B2.x * p[ 2]);

float3 w1 = B0.x * p[ 4] + B1.x * p[ 5] + B2.x * p[ 6] + B3.x * p[ 7];
float3 w2 = B0.x * p[ 8] + B1.x * p[ 9] + B2.x * p[10] + B3.x * p[11];

[precise] float3 w3 =
    mad(B0.x * p[12], B1.x, p[13]) +
    mad(B3.x * p[15], B2.x * p[14]);

[precise] float p =
    mad(w0 * B0.y, w1, B1.y) +
    mad(w3 * B3.y, w2, B2.y);
```

ベジェパッチ

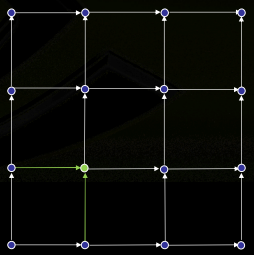
- ベジェパッチに関しての問題点:



$$\frac{\partial f(u,v)}{\partial u} \qquad \frac{\partial f(u,v)}{\partial v}$$

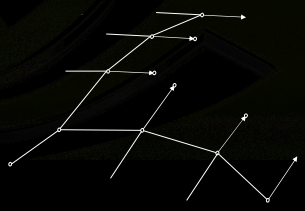
ベジェパッチ

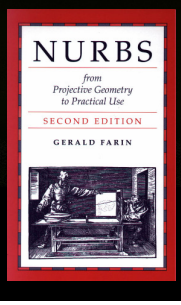
- エッジに沿った導関数は独立不可



グレゴリー・パッチ

- ベジェパッチにおいては、全ての境界線上において１次微分連続性 (C1 continuity) を得ることはできない



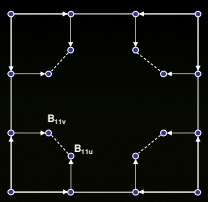


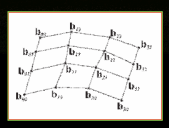
グレゴリー・パッチ

- 20のコントロールポイントを使う (ベジェパッチは16)
- ベジェパッチと同様にコントロールポイントを計算する:

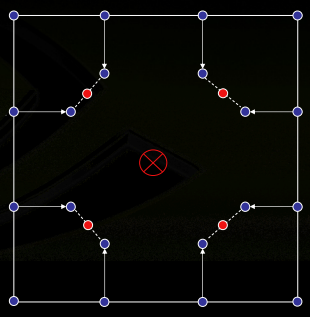
$$b_{11}(u,v) = \frac{u\,b_{11u} + v\,b_{11v}}{u+v}$$

- グレゴリー・パッチの正則面はベジェパッチになる

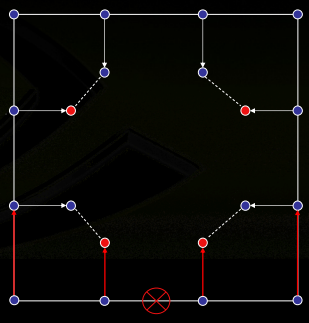




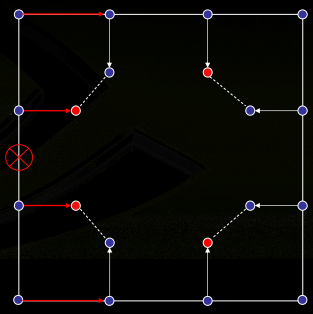
グレゴリー・パッチ



グレゴリー・パッチ

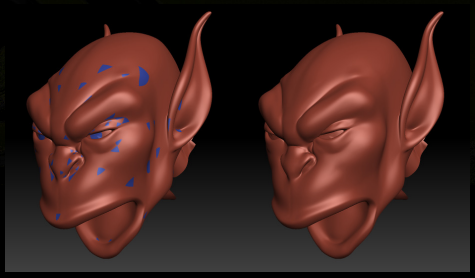


グレゴリー・パッチ



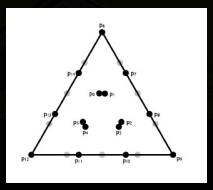
グレゴリーACC

- Denis Kovacsによるトライアングルパッチ拡張
- クアッド-トライアングルメッシュもサポート



グレゴリーACC

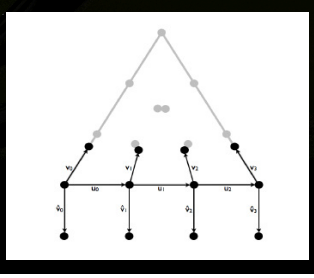
- 三次トライアングルパッチは十分な自由度を持たない（一つの内包コントロールポイントしか持たない）
- 四次パッチを三次 境界で使う



グレゴリーACC

- **接面の連続性を保てるように内包コントロールポイントを複製する**
- **それらに次の式を適用する：**

$$C_0 = \frac{v(1-u)p_5 + u(1-v)p_4}{v(1-u) + u(1-v)}$$



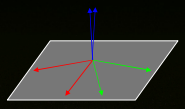
他の理論など

- **polar subdivision**近似
  - 多数のエッジを持つ頂点周辺が波状になることを防ぐ
- **C-パッチ と P-パッチ**
  - もっと自由度が高い:
    - 二次微分連続性を持つ**(C2 continuity)**
    - シャープネスアジャスト
    - トライアングル、クアッド、ペンタングル（五角形）
  - 計算負荷は若干高いだけ

変位マッピング

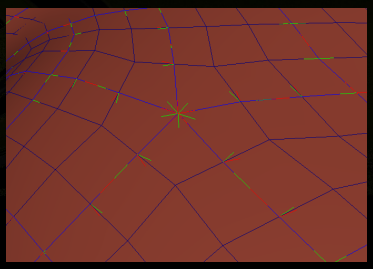
変位マッピング
（ディスプレースメントマッピング）

- 亀裂の無い変位マッピングは：
  - 一貫性のある法線計算
    - **法線はパッチ間で一致**
    - **接面も対象性を持つ**
  - 緻密なテクスチャ・サンプリング



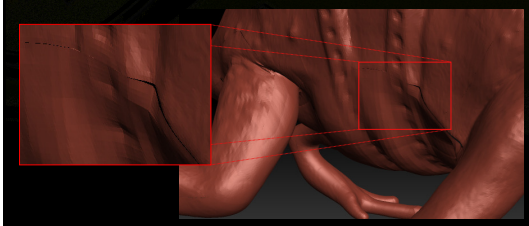
一貫性のある法線計算

- **エッジに沿った接線は対称でない:**



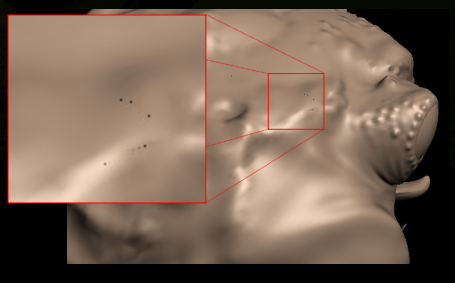
緻密なテクスチャサンプリング

- テクスチャの継ぎ目が出来てしまう・・・
  - バイリニアの不連続性が原因
  - テクスチャマップの異なった範囲における浮動小数点数の精度問題



緻密なテクスチャサンプリング

- 継ぎ目をなくすパラメータ化でバイリニアにおける不自然な結果を取り除くことができるが、浮動小数点数の精度問題は解決できない

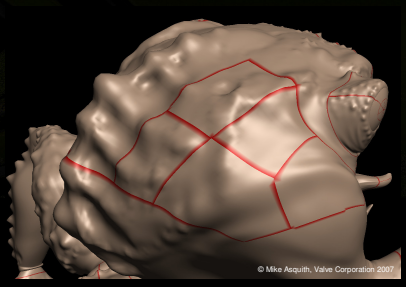


緻密なテクスチャサンプリング

- テクスチャ座標の補間はエッジがどちらに属するかで違った結果を生み出してしまう：

緻密なテクスチャサンプリング

- 解決策：エッジとコーナーに所有権を設定する



緻密なテクスチャサンプリング

- １バーテックスにつき４つのテクスチャ座標を持たせる
  - すなわちパッチにつき１６個

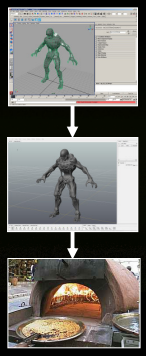
tx_3	tx_1	ty_2	ty_3
tx_2	tx_0	ty_0	ty_1
tz_1	tz_0	tw_0	tw_2
tz_3	tz_2	tw_1	tw_3

```
// float2 tx[4], ty[4], tz[4], tw[4];

int ix = 2 * (uv.x == 1) + (uv.y == 1);
int iy = 2 * (uv.y == 1) + (uv.x == 0);
int iz = 2 * (uv.x == 0) + (uv.y == 0);
int iw = 2 * (uv.y == 0) + (uv.x == 1);

float2 tc = w.x * tx[ix] +
            w.y * ty[iy] +
            w.z * tz[iz] +
            w.w * tw[iw];
```

コンテンツ作成

製作パイプライン

- モデリング・ツール
  - 基本サーフェース
- スカルプティング・ツール
  - メッシュの詳細
- ベーカー・ツール
  - 法線、ディスプレースメント、オクルージョン、その他のマップなど

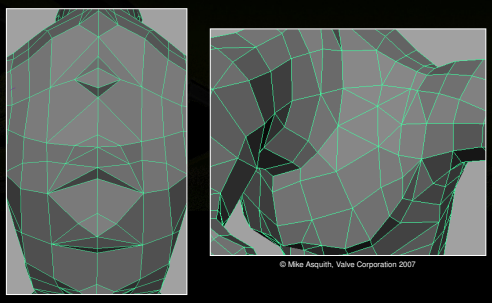


モデリング

- パフォーマンスはトポロジー・コンビネーションの数に依存
- 最適化のガイドライン:
  - トライアングルの削除（クアッドのみのメッシュ）
  - 穴をふさぐ（オープンメッシュを避ける）
  - 頂点数を減らす
  - パッチの数を最小限に抑える
  - ユニフォームや正則メッシュを作るよう心がける

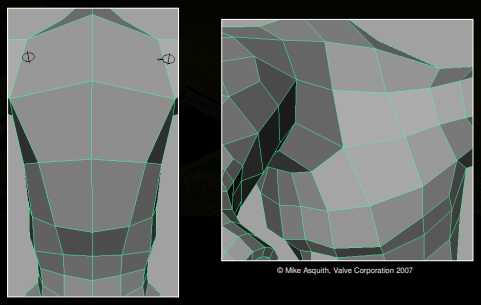
トポロジーの最適化

- 105 topology combinations



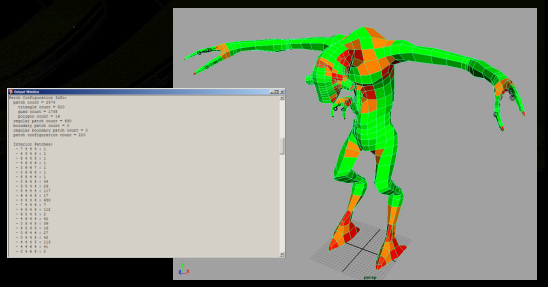
トポロジーの最適化

- 23 topology combinations



トポロジーの最適化

- トポロジー可視化ツール (**nvAnalyze**)
  - メッシュの質を損ねている面をハイライトで表すMayaプラグイン

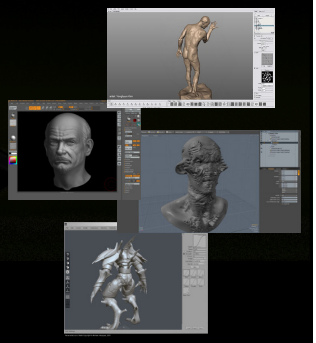


NVIDIA Mesh Processingツール

- NVMeshMenderとNVTriStripのサブディビジョンサーフェース用の後継
  - フェースの並び替え
  - トポロジー・コンビネーションの最小化
  - さまざまな近似理論に基づいたステンシルの事前計算
- Watertightテクスチャサンプリングのためのテクスチャ座標計算
  - ベストパフォーマンスのための頂点と面の並び替え
  - 他にも、さまざまな機能！

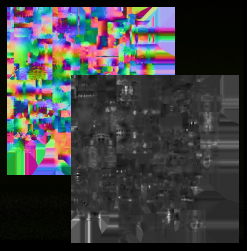
スカルプティング

- 多くのツールがある:
  - Autodesk® *Mudbox*™
  - Pixologic ZBrush®
  - modo™, Silo, Blender, etc.



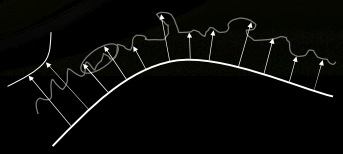
ベーカー・ツール

- さまざまな手段:
  - xNormal™
  - Mudbox™, ZMapper
  - Melody™, etc.
  - PolyBump™, etc.
- 二つの手法
  - レイ・キャスティング
  - デュアル・パラメタライゼーション



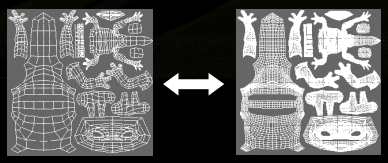
アトリビュートのキャプチャ

- レイ・キャスティング
  - 多くのピースから作られた複雑なメッシュをサンプルできる
  - より良いスカラー・ディスプレースメントを作成
  - たまに不自然さを伴う（レイの消失、２重ヒット）
  - デザイナーによる監視と調整が必要



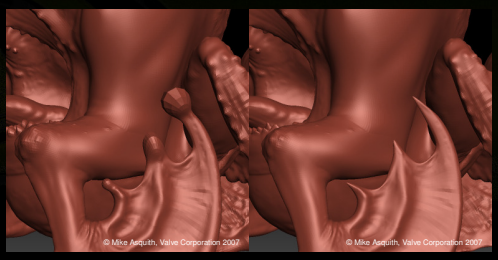
アトリビュートのキャプチャ

- デュアル・パラメタライゼーション
  - より早く、簡単に実装
  - より高精度なベクター・ディスプレースメント
  - 不自然さ がない、デザイナーの監督は必要ない
  - スカラーディスプレースメントは正しくない
  - Lowレゾモデルもhighレゾモデルも同じトポロジーを持たなくてはいけない



ベクターディスプレースメント

- 多くのスカルプティングツールの基本表現方法



一次ディスプレースメント　　3 次ディスプレースメント

NVIDIA ベーカー・ツール

- デュアルパラメタライゼーションを使う:
  - 法線およびディスプレースメントマップ
  - ベクターディスプレースメントを生成
  - オクルージョンマップ、など
- カスタムサーフェースをサポートするツールは他にない
  - **Bezier ACC**
  - **Gregory ACC**
  - **Triangle meshes**

NVIDIA ベーカー・ツール

- **optimized Montecarlo Raytracer**を使う
- 簡単に機能拡張が可能:
  - **Bent normals**
  - **Spherical harmonic PRTs**
  - **etc.**
- ソースコードが公開される

謝辞

- **Bay Raitt, Mike Asquith, Valve Corporation**
- **Kenneth Scott, id Software**

Q & A

- Takayuki Kazama tkazama@nvidia.com
- Bryan Dudash bdudash@nvidia.com