Lost Planet 2:DirectX11 Features

Lost Planet 2について

- サードパーソンアクションシューティング
- MT-Framework 2.0を採用
  - マルチプラットフォーム対応の内製エンジン
    - PS3,X360,PC(DX9,DX10,DX11),Wii,3DS …
- PC版はDirectX11に対応し、10月発売予定

アジェンダ

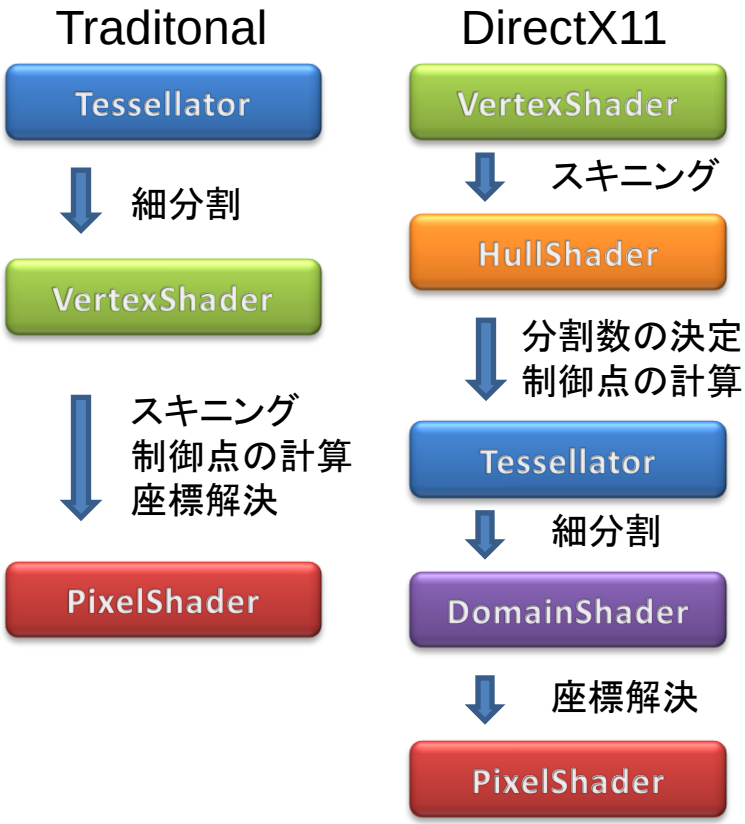
- テッセレーションの活用
- コンピュートシェーダーの活用

Tessellation

テッセレーションの活用

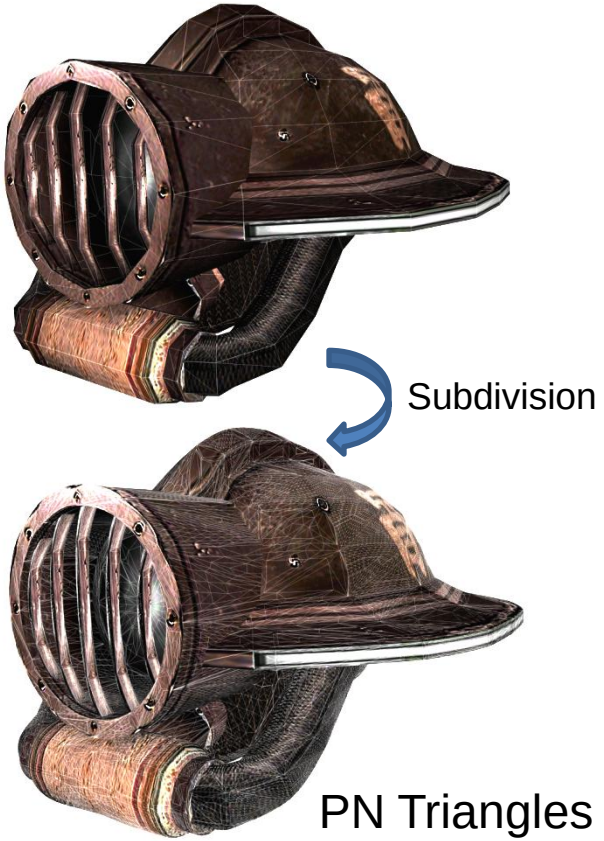
テッセレーション

- ポリゴンの細分割
  - ジオメトリ品質の改善
  - 柔軟なLOD
  - データ量とバンド幅の削減
- 従来のテッセレーション
  - N-Patchか、ベンダー独自のAPI
  - 分割数に応じて頂点処理の負荷が増大
- DirectX11のテッセレーション
  - 追加のシェーダーステージ
    - HullShader/Tessellator/DomainShader
  - より効率的なテッセレーション
    - 分割レベルに依存しないスキニング
    - 適応的な分割レベルの決定



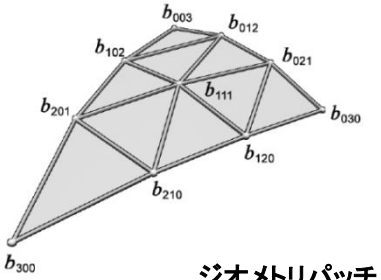
サブディビジョンサーフェイス

- 曲面を滑らかにする
- よく知られたアルゴリズム
  - PN Triangles
    (N-Patch)
  - Phong Tessellation
  - Approximating Catmull-Clark
    (ACC)

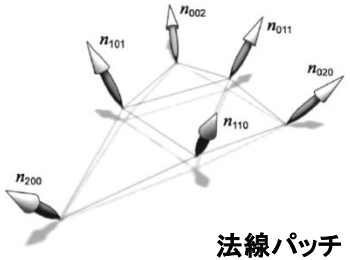


PN Triangles

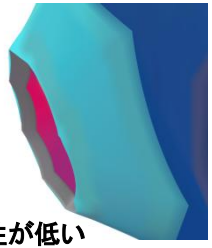
- Curved PN Triangles
  - by Alex Vlachos, JörgPeters, Chas Boyd, and Jason Mitchell, I3D 2001
- 座標と法線から制御点を生成
  - ジオメトリパッチ用の追加の7点
  - 法線パッチ用の追加の3点
- 利点
  - 既存のデータ構造がそのまま利用可能
  - エッジ単位の制御が可能
- 欠点
  - DCCツールのサポートがない
  - サーフェイス間の連続性が低い



ジオメトリパッチ



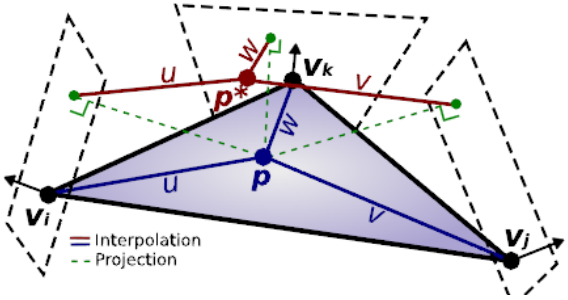
法線パッチ

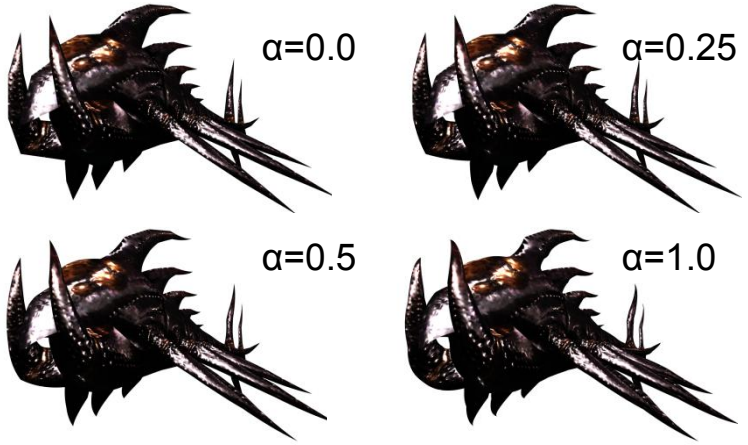


サーフェイス間の連続性が低い

Phong Tessellation

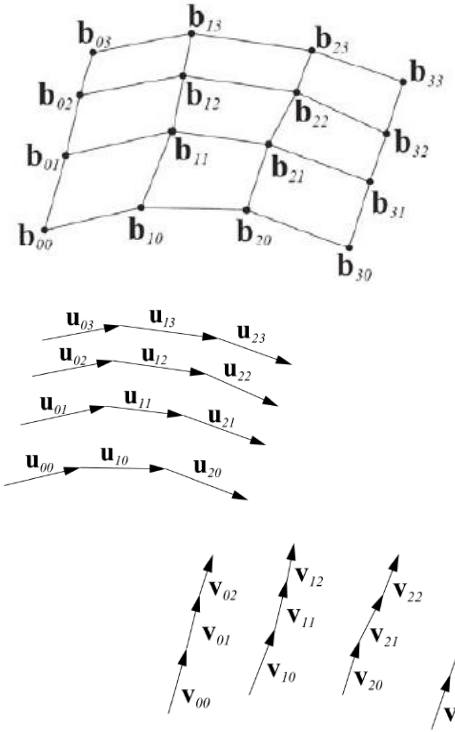
- SIGGRAPH2008 Asia paper
  - by TamyBoubekeurand Marc Alexa
- 座標、法線、重心座標を元に直接補間
  - 追加の制御点が不要
  - α定数で丸みを制御
- 利点
  - 既存のデータ構造がそのまま利用可能
  - シンプルで高速
- 欠点
  - PN-Trianglesよりも曲面の連続性が低い
  - エッジ単位の制御ができない





Approximating Catmull-Clark

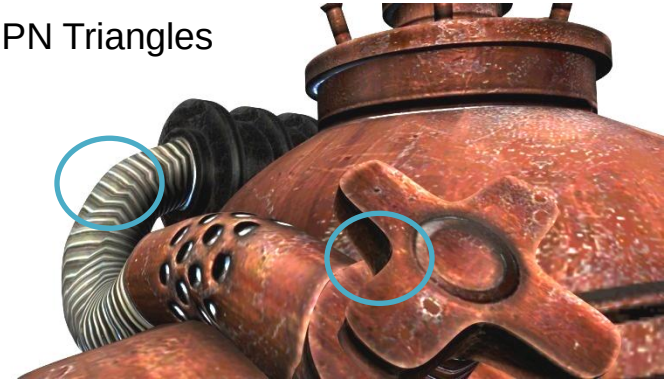
- Approximating Catmull-Clark Subdivision Surface with BicubicPatches
  - *by Charles Loop and Scott Schaefer*
- Catmull-Clark曲面をベジェパッチで近似
  - 16のジオメトリパッチ用制御点
  - 24の法線パッチ用制御点
- 利点
  - 多くのDCCツールが標準的にサポート
  - 連続性の高い高品質な曲面
- 欠点
  - 専用のデータ構造が必要
    - 4点と近傍の可変個の点
  - 計算負荷が高い
    - 細分割を前提とした少ないポリゴンでのモデリングが必要



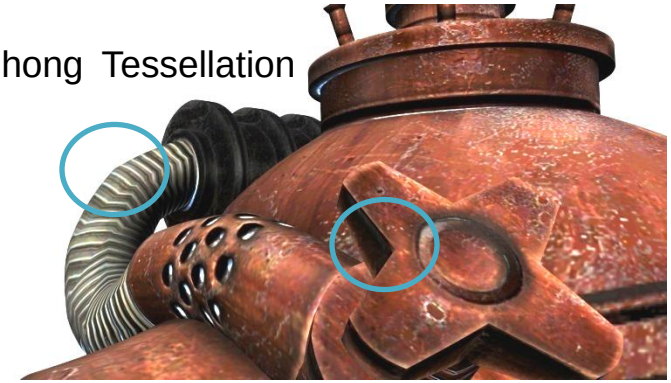
Lost Planet2での選択

- Catmull-Crarkは現実的ではない
  - リソースをDirectX9版と共通にしたい
    - Catmull-Crarkは専用のデータ構造が必要
- PN-Trianglesを採用
  - リソースをDirectX9版と共通にできる
    - トポロジをTriangleListにするだけで良い
  - エッジ単位での制御が容易
  - Phong Tessellationよりも高品質

PN Triangles

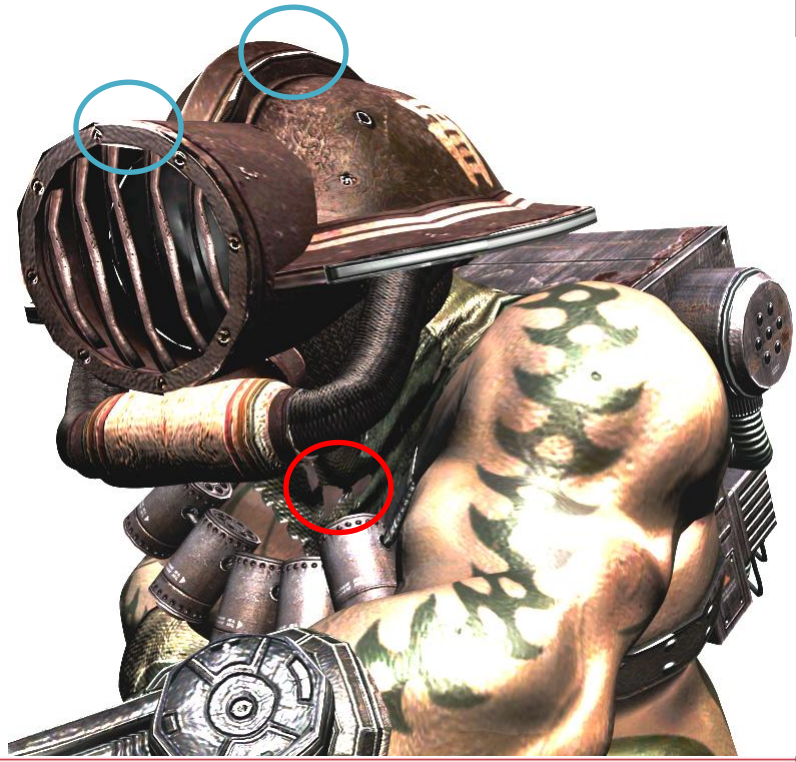


Phong Tessellation



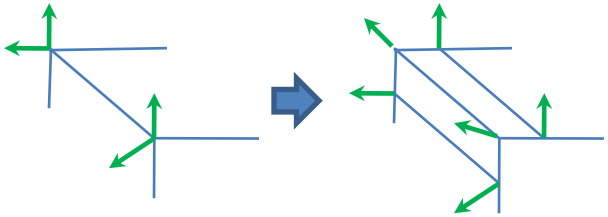
PN-Trianglesの問題

- クラック(穴が空く)
  - 同じ座標の頂点で法線ベクトルが異なる
  - 重なったエッジで頂点座標が異なる
    - Tジャンクション
- デザイン的な問題
  - 太く/丸くなりすぎる
  - 重なったモデルが膨らんでめり込む

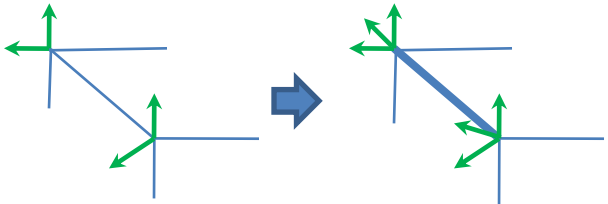


クラック対策

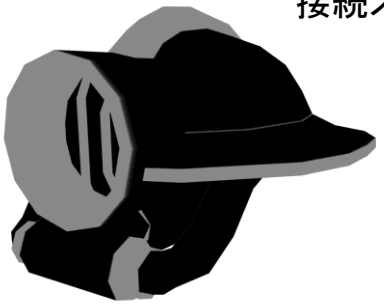
- いくつかのアプローチ
  - トランジショナルポリゴン
    - クラックするエッジ境界を事前に分割する
      - ポリゴン数が増加
      - 共通リソースではDirectX9版のパフォーマンスに影響
  - 接続ストリップ
    - クラックするエッジ境界をストリップポリゴンで接続する
      - ポリゴン数の増加、境界のテクスチャが伸びる
  - テッセレーションマスク
    - クラックするエッジ部分はテッセレーションしない
      - 多くのハードエッジでテッセレーションの効果が得られない
  - ジオメトリ法線
    - ジオメトリパッチ専用の法線を用いる
      - 詳細を後述



トランジショナルポリゴン



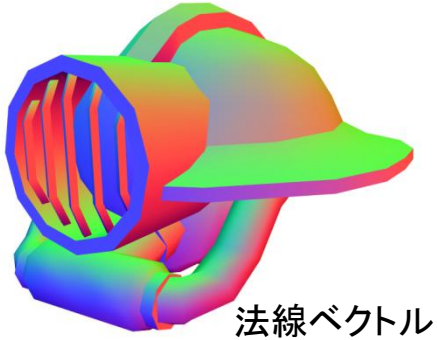
接続ストリップ



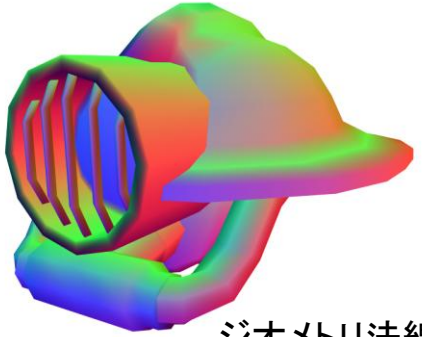
テッセレーションマスク

ジオメトリ法線

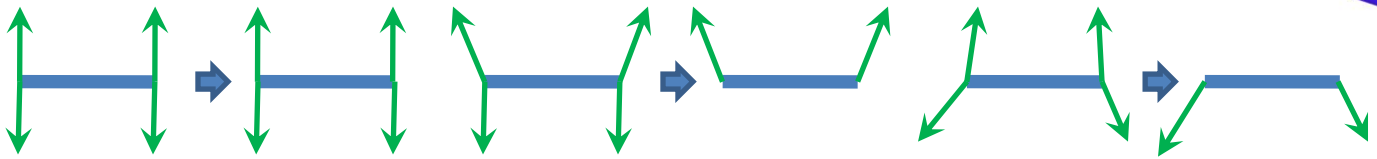
- ジオメトリパッチ計算用の法線ベクトル
  - 同一座標での法線の同一性を保証
  - 法線パッチは通常の法線ベクトルで計算
- ジオメトリ法線の生成
  - 同一座標で異なる法線を持つエッジを統合
    - エッジを構成する2つの法線ベクトルの内積が小さい方の法線に統合
    - 両方の内積が1.0の場合はマージ対象に加えない
  - 同一座標の法線を再平均化



法線ベクトル



ジオメトリ法線



ジオメトリ法線の問題

- テクスチャの歪み
- サーフェイスが丸くなりすぎる

オリジナル

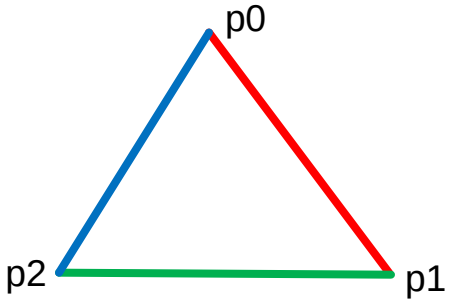
通常法線を利用

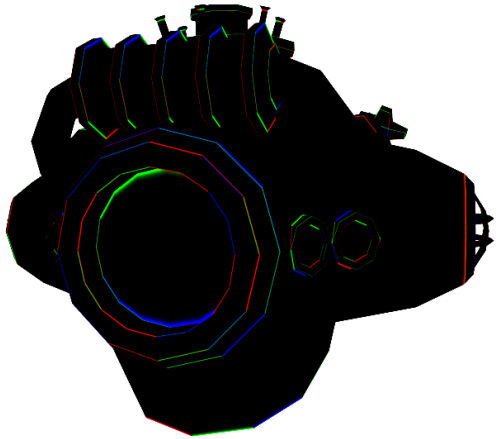
ジオメトリ法線を利用

エッジ単位の制御

- 頂点毎にエッジ判定用のフラグを追加
  - 三角形の各頂点毎に3bitのフラグ
    - p0-p1エッジがクラックするか否か
    - p1-p2エッジがクラックするか否か
    - p2-p0エッジがクラックするか否か

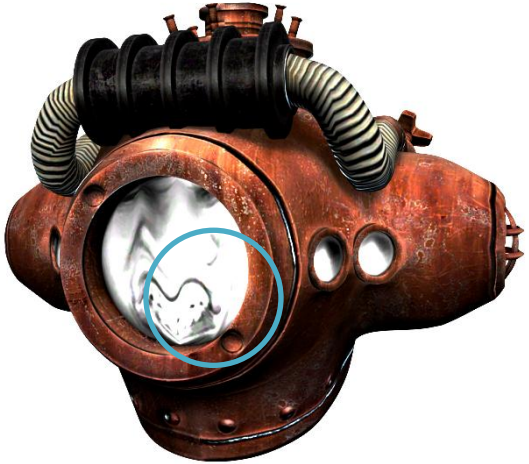
- エッジ単位に利用する法線を決定
  - エッジを構成する2頂点のフラグの論理積が真の時、対象のエッジ
  - 対象エッジの制御点の計算のみ、ジオメトリ法線を適用



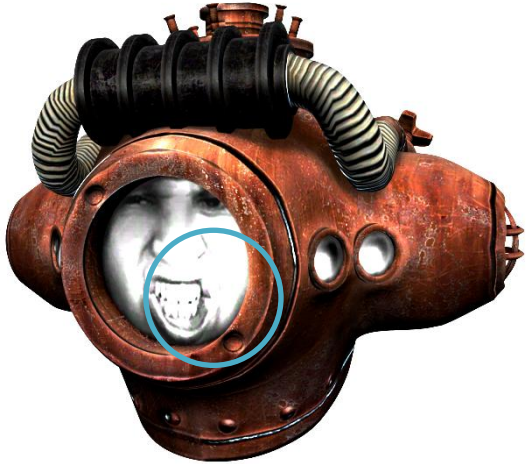


ジオメトリ法線の使用フラグ

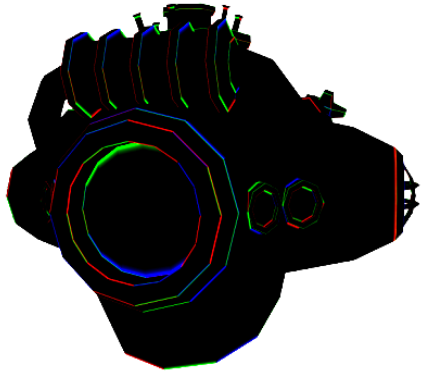
エッジ単位の制御



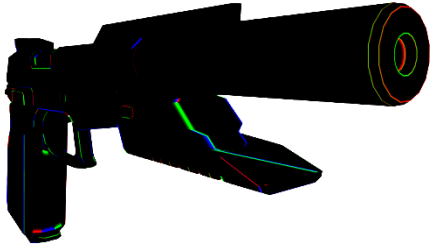
ジオメトリ法線を利用



エッジ単位の制御

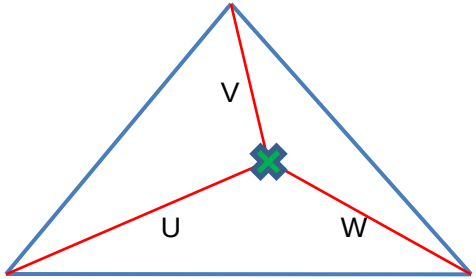


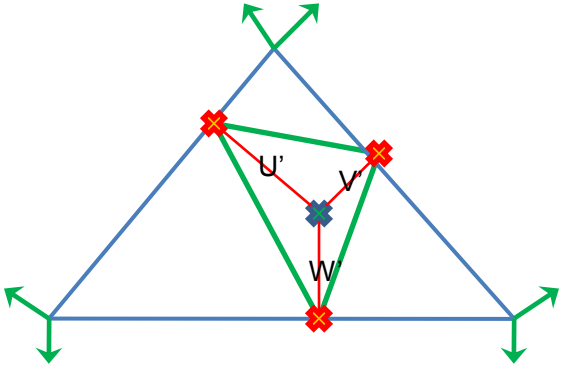
ジオメトリ法線フラグ



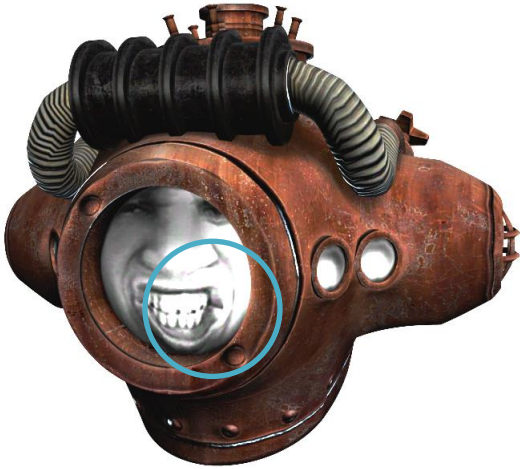
Phong Tessellationへの適用

- エッジ単位の制御が困難
  - 3頂点を接空間に投影した座標の線形補間
- アルゴリズムをエッジ基準に拡張
  - 重心座標と各エッジとの最近点から構成される、内接三角形を定義
  - 内接三角形からの重心座標(U' V' W')を計算
  - 内接三角形の各点の投影座標をエッジごとに線形補間して求める
  - 重心座標(U' V' W')から目的の座標を求める
- ジオメトリ法線フラグが無効の場合は通常計算





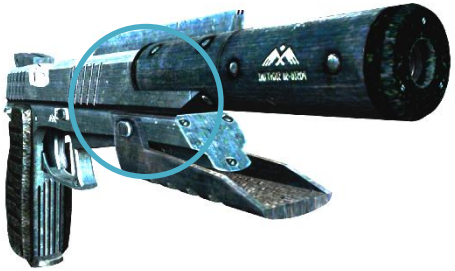
Phong Tessellationへの適用



ジオメトリ法線を利用

エッジ単位の制御

PN Triangles



ジオメトリ法線

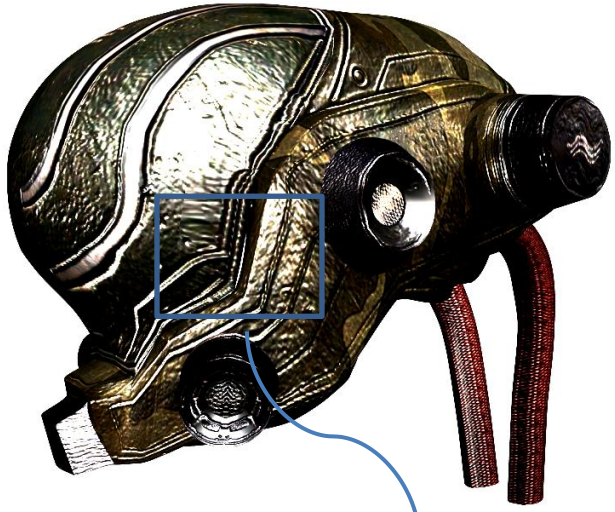
- 利点
  - ポリゴン数の増加がない
    - 頂点毎に4バイト程度の追加情報のみ
      - DirectX9版への影響が少ない
  - 典型的なエッジではうまく動作
- 欠点
  - シェーダー命令数の増加

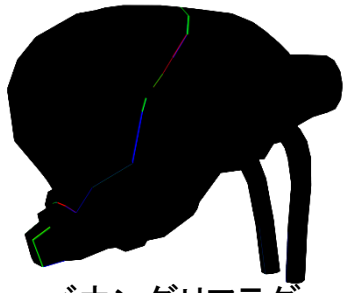
通常法線

ジオメトリ法線

その他のクラッキング

- Tジャンクション
  - 重なったエッジを構成する頂点座標が異なる
- 対策
  - 共有されない境界エッジは線形補間した座標を使用
  - 境界エッジ識別用のバウンダリフラグを追加
    - 頂点毎に3bit





バウンダリフラグ

デザイン的な問題の対応

- モデルが太る/丸くなりすぎる
  - 本来フラットな面なのに、法線がフラットでない
    - アーティストがいい感じに調整した結果
- 調整パラメータを導入
  - エッジの長さと丸さに基づいた係数
    - 長くて丸いエッジほど変化を緩やかにする
    - (1.0 – エッジ法線の内積) × 定数 / エッジの長さ
  - エッジ単位にジオメトリパッチの制御点を補正
  - マテリアル単位にアーティストが調整

オリジナル

PN Triangles

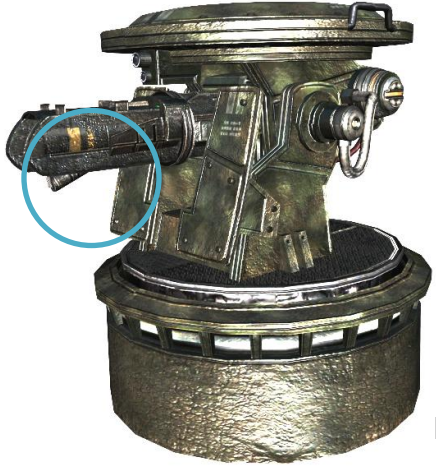
デザイン的な問題の対応

F=0

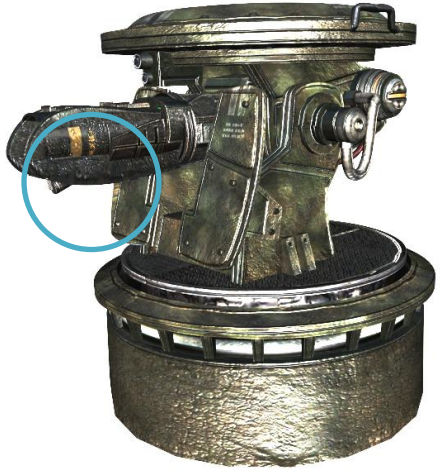
F=10

F=100

F=0



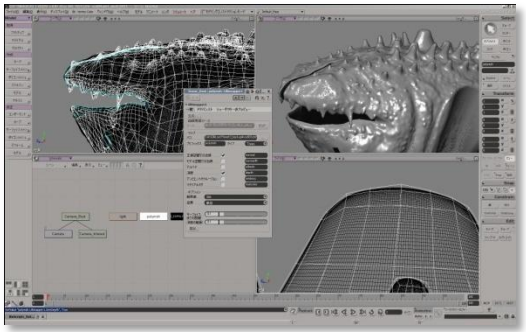
F=100

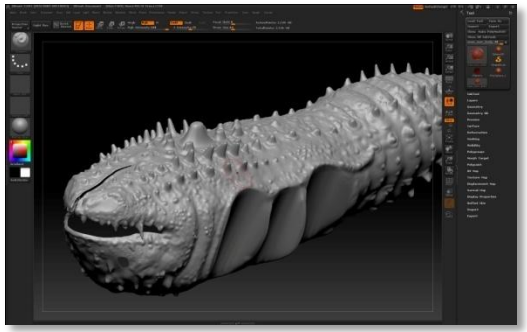


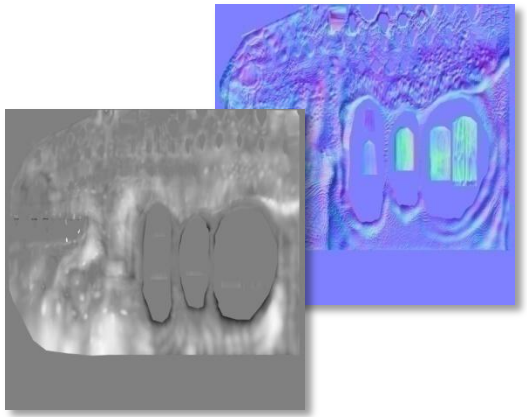
F=10000

ディスプレースメントマップの作成

- コンテンツパイプライン
  - XSI
    - ベースモデル作成
  - Z-Brush
    - ハイモデル作成
  - XSI(AltiMapper)
    - ハイトマップ、法線マップ作成
- LostPlanet2のモデル
  - 正確なハイポリゴンモデルが存在しない
    - CrazyBumpなどで作成した2Dベースの法線マップと合成
    - 部分ごとにZ-Brushの使用/未使用が異なる
  - Z-Brushでハイモデルを作成しなおし
    - 一部のボスモデルのみの対応に

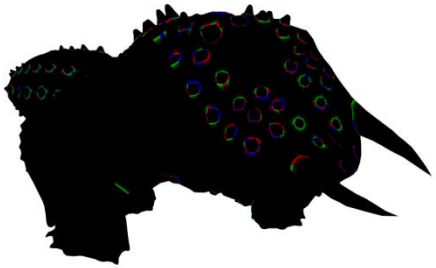






ディスプレースメントマッピングの問題

- クラッキング
  - 同一座標のエッジが参照するテクスチャ座標が異なる
  - 同一座標のエッジが参照するテクスチャが異なる
- クラッキング対策
  - 複数のUVを持つアプローチは困難
    - LostPlanet2のモデルは複数枚のテクスチャから構成される
    - DirectX9版のデータが増える
  - 非連続なUVを持つエッジはディスプレースしない
    - バウンダリフラグに統合
    - 品質的には望ましくない



バウンダリフラグ

非連続なUV境界

ディスプレースメントマッピング

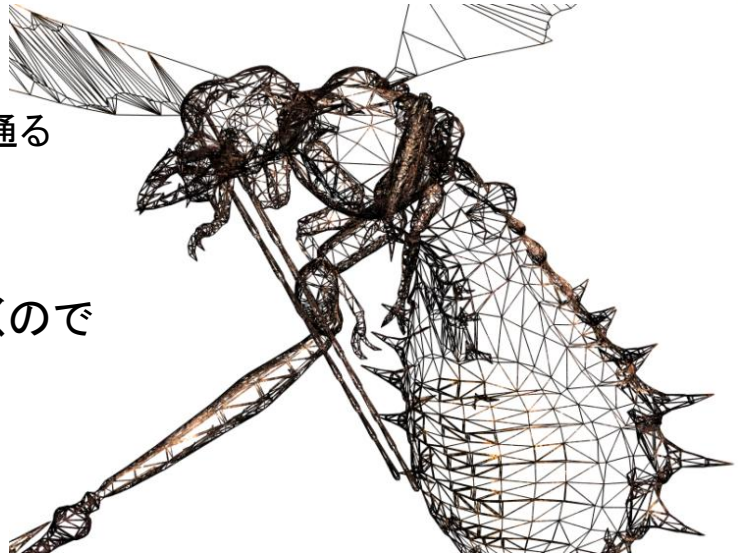
- 違いが分かりにくい
  - 重要なシルエットは既にポリゴン化されている
    - 細かなディティールは法線マップ化されている
  - ディスプレースメントを前提としたモデリングが必要

オリジナル

ディスプレースメントマップ適用

テッセレーションの最適化

- アダプティブテッセレーション
  - 最適な分割数の決定
    - 見た目に影響を与えない範囲で分割数を少なくする
  - 不要なパッチの排除
    - 見えないパッチはカリングする
      - 画面外や裏面でもドメインシェーダーまで全て通る
- シェーダーユニットの負荷バランス
  - 可能な限り上位のシェーダーで処理
    - 特に頂点シェーダーは頂点キャッシュが効くので高速

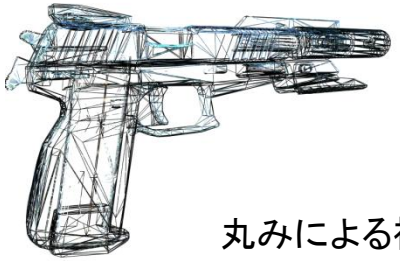


最適な分割数の決定

- エッジ単位に分割数を決定
  - TessFactor = Max(L × R × C / D0 , L × R × C / D1)
  - L … エッジの長さ
  - R … 画面解像度(ピクセル数)
  - C … ユーザー定義の分割定数
  - D0,D1 … 各点までのワールド空間の距離
- シルエット以外の分割数を下げる
  - 法線と視線ベクトルとの内積に基づく
    - 簡易版
- 丸みの少ないエッジは分割数を下げる
  - エッジを構成する2頂点の法線の内積に基づく
    - PN Triangles/ Phong Tessellation のみ

基準分割

シルエット補正

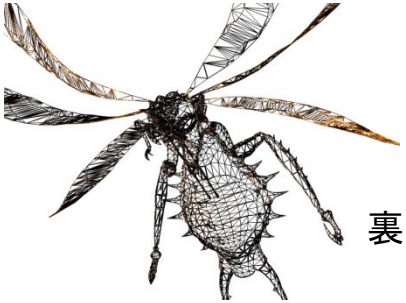


丸みによる補正

不要なパッチの排除

- 裏面のパッチの消去
  - 3頂点全ての法線ベクトルと視線ベクトルの内積が負の場合
- 画面外のパッチの消去
  - 3頂点全てが視錐台外の場合
- サーフェイスの膨らみを考慮
  - PN Trianglesやディスプレースメントマップでは実際のサーフェイスよりも外側/内側に膨らむ
  - 少し大きめに判定マージンを持つ
    - 正確には、頂点を共有するポリゴンの最大変位度を数値化し、頂点単位に保持して判定時に考慮すべき。
- 頂点シェーダーでクリップフラグを計算
  - 頂点単位に視錐台6面の内外判定と裏面判定を行う
  - ハルシェーダーで3頂点のフラグの論理積をとる

基準分割



裏面カリング

視錐台カリング

Demo

まとめ

- テッセレーション
  - ついにDirectX11で標準化
    - より柔軟で効率的な実装が可能
  - PN Triangles
    - 既存のデータでも互換性を維持したまま恩恵が受けられる
    - エッジ単位の柔軟な制御が可能
  - HDゲームでは効果は微妙
    - 既に十分ハイポリゴン
      - わかる人にしか分らない
- 次世代機に期待!!
  - テッセレーション前提のコンテンツ制作が重要
    - Catmull-Clark曲面
    - ベクターディスプレースメント

CAPCOM

CEDEC 2010

ComputeShader

コンピュートシェーダーの活用

コンピュートシェーダー

- Wave Particlesによる
  インタラクティブな水面表現
- Soft Bodyによる
  巨大ボスの表皮表現

DirectX11の水面

- 全体的な波
  - ディスプレースメントマップを多重合成して表現
    - 従来のプロシージャル手法と同じ

      ex) 大理石の模様や、雲のテクスチャなど

- 細かな波（波紋）
  - Wave Particlesの簡易版で表現
    - キャラクターの動きや銃撃による波紋の発生
    - 壁などの障害物で波が反射

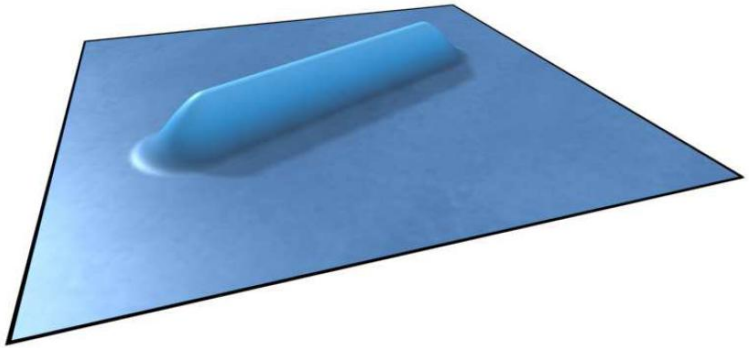
DirectX9の水面

DirectX11の水面

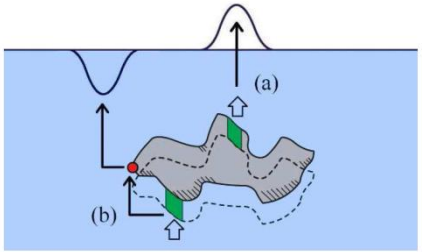
Wave Particles

Cem Yuksel, Donald H. House, and John Keyse. SIGGRAPH 2007 (TOG)

- 波を直進する粒子として表現
  - 粒子間相互作用はしない
  - ある程度進んだら粒子を分割
  - 壁に当ったら反射
  - 時間とともに強度が減衰し消滅



- 粒子をポイントレンダリング、フィルタリングして
  ハイトフィールド水面を作成



- 波の発生
  - 物体の形状や動きを考慮して発生

ロストプラネット2への適用

- 簡易版Wave Particles
  - 粒子の分割を行わない
    - バッファオーバーラン対策（発生する個数が予測できない）
  - 波の発生は指定した点から放射状にのみ
    - 当初は作成済みのエフェクトに連動させて自動生成をもくろむ
    - 結局は足元や着弾時にプログラム的に生成
- 制作済みのDX9版のリソースを有効活用
  - 適用対象は爆風エフェクトで反応するように作られた水面
  - シェーダだけで実装できるので適用コストは軽い
    背景モデルの修正は必要ない

CS実装：粒子バッファ

粒子は生成および時限消滅するので増減処理が必要

- Append Structured Buffer(ASB)
  - Unordered Access View(UAV)を持てる（ランダムread/write可能）バッファ
  - 要素として構造体が使える
  - バッファの最後に指定した要素を追加（Append）できる
- 粒子の増減管理に2枚のASBを使用
  - ピンポン式に処理

粒子の構造体

変数	型
発生点	float2
進行方向	float2
経過時間	float
生存限界時間	float
強さ	float

CS実装：粒子の更新

1. 各粒子の時間を進ませる
2. 移動量から壁との衝突判定
   - 事前に水面と背景の境界線を抽出
   - スタックレスのAABB-Treeにして衝突判定
3. 経過時間が生存限界まで達すると消滅
   - 生存している粒子のみ粒子バッファにAppendで書き出し

CS実装：CS起動（Dispatch）

- **ID3D11DeviceContext::Dispatch(nX, nY, nZ)**
  nX, nY, nZ: 起動するスレッド（グループ）数

- 粒子数は増減するので起動するスレッド（グループ）数が不定

  ID3D11DeviceContext::DispatchIndirectを使用して
  更新CSを起動
  - ディスパッチ数をGPU側で指定できる
    - **ID3D11DeviceContext::CopyStructureCount**
  - CPUで粒子バッファの粒子数を取得（リードバック）する必要がない

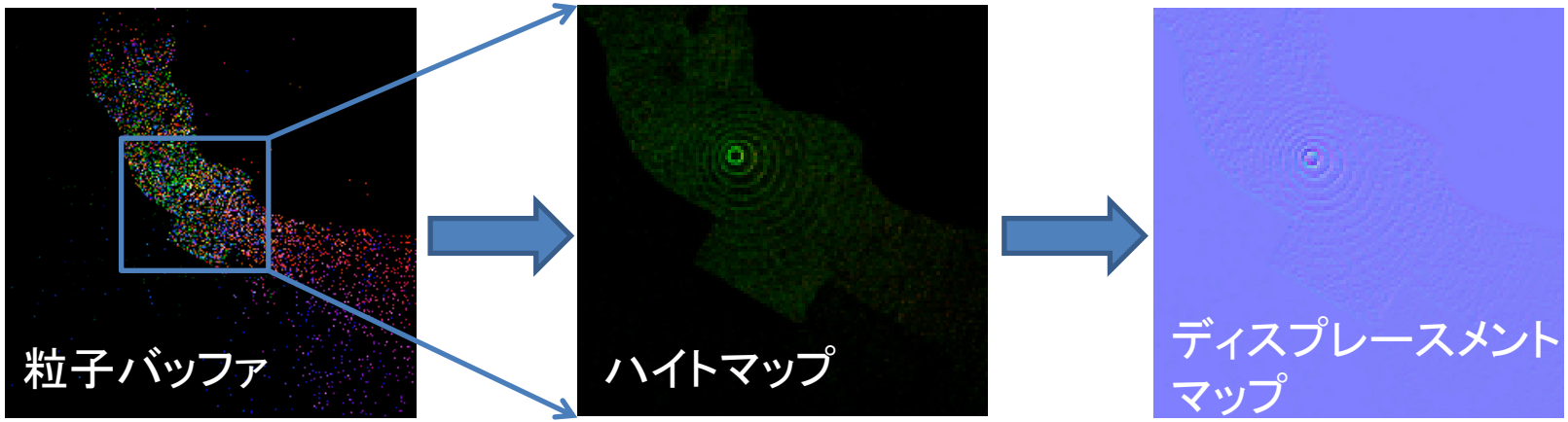
CS実装：頂点バッファ更新

ポイントレンダリングのために粒子バッファ（ASB）を頂点バッファ化

- Byte Address Buffer(BAB)
  - Unordered Access View(UAV)を持てる（ランダムread/write可能）バッファ
  - バイト列として扱え、指定したアドレスに値を書き出すことが可能
  - VertexBufferをByteAddressBufferとして使用
    - D3D11_BIND_VERTEX_BUFFER | D3D11_BIND_UNORDERED_ACCESS
- コンピュートシェーダーを使って粒子バッファから頂点バッファへコピー

ディスプレースメントマップ作成

- **ID3D11DeviceContext::DrawInstancedIndirect**を使いポイントレンダリング
  - 粒子バッファ（ASB）から描画する点数を取得（CPUリードバックなし）
- フィルターをかけてハイトマップを作成
- ハイトマップからディスプレースメントマップを作成
  - 法線マップ(RGB) +ハイトマップ(A)



Wave Particles ビデオ

Wave Particles ビデオ

ソフトボディ

- 巨大ボスの表皮に「よりリアル」な動きをもたせる
  - 自らの動きに対する反応（足踏みによる表皮揺れ）
  - 爆風に対する反応（ロケットランチャーなど）
- 「重さ、重量感」をより感じさせ、その迫力を増す
- 既存モデルにアーティストが表皮（表面）の「柔らかさ」をペイントするだけで適用

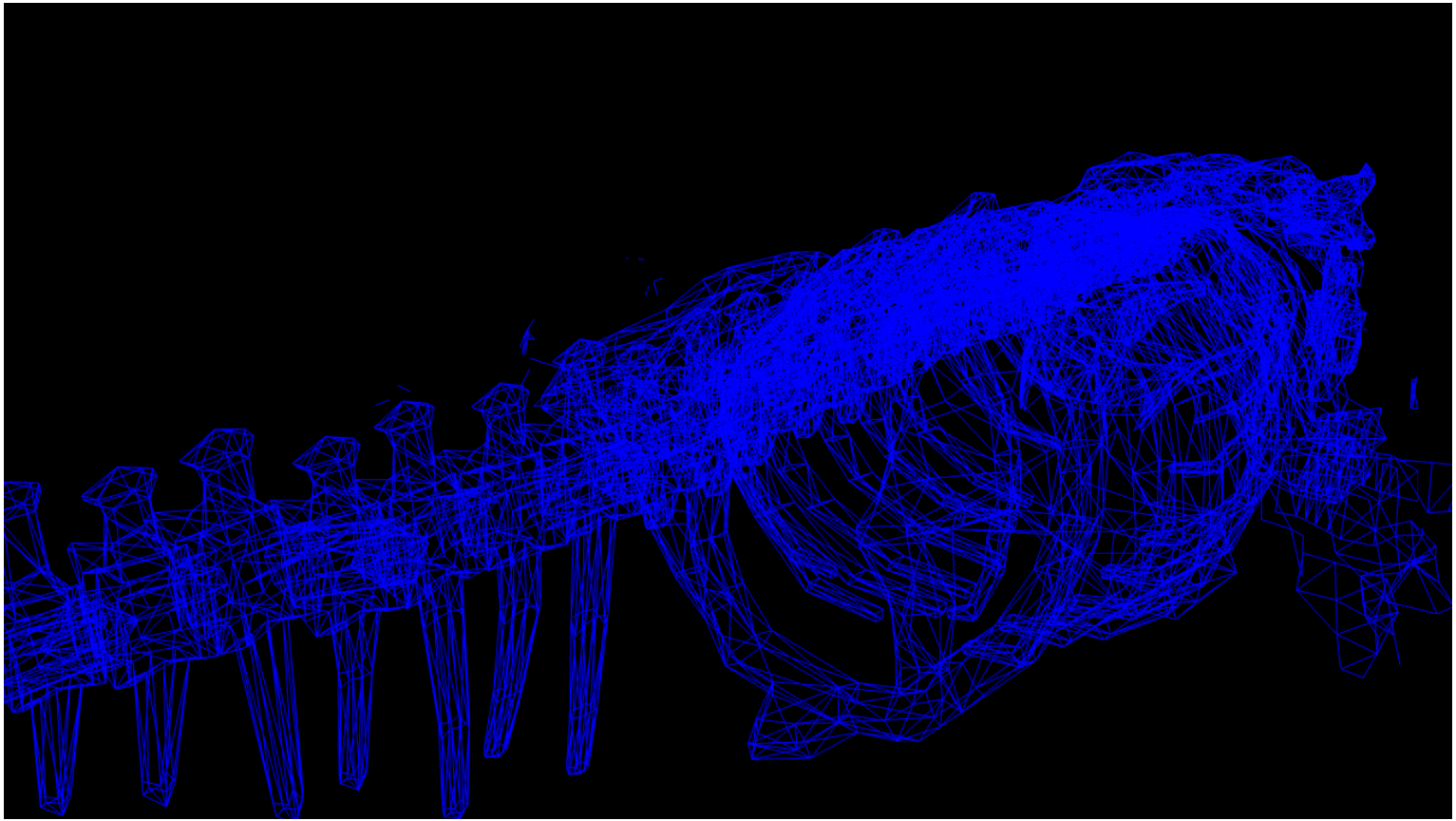
ソフトボディOFF（ビデオ）

ソフトボディON（ビデオ）

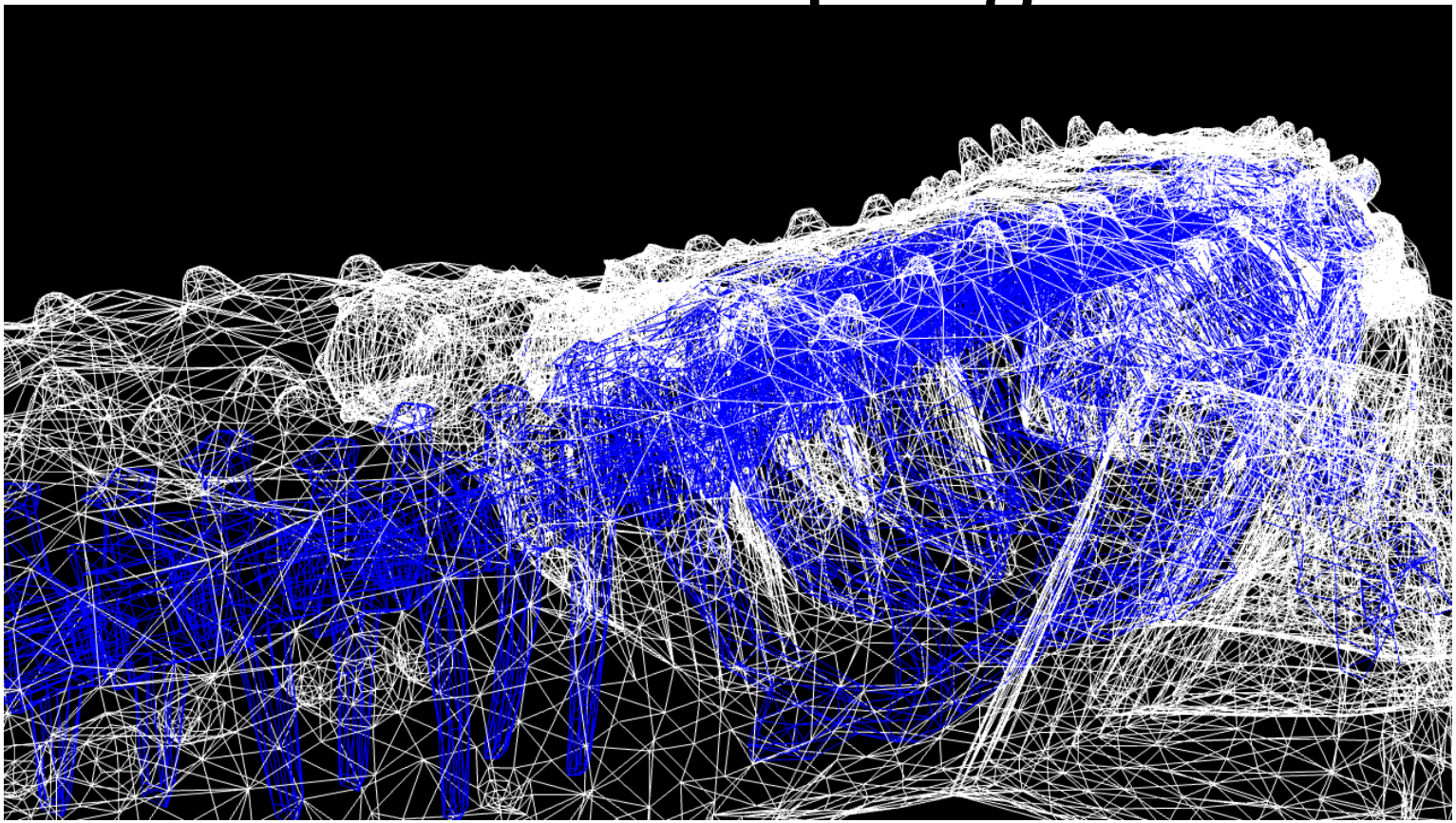
質点バネモデル (Mass-Spring Model)

- 3種類のバネで表皮の動きを作る
  - 表皮面を形作るSurface Springs
  - 骨と表皮をつなぐInner Springs
  - 質点（シミュレーション頂点）とスキニング頂点をつなぐGoal Springs
    - 頂点ごとに「柔らかさ」をペイント
- Connected Surfaces Structureがベース
  - Jesper Mosegaard: “Realtime Cardiac Surgery Simulation”, Master’s thesis, 2003

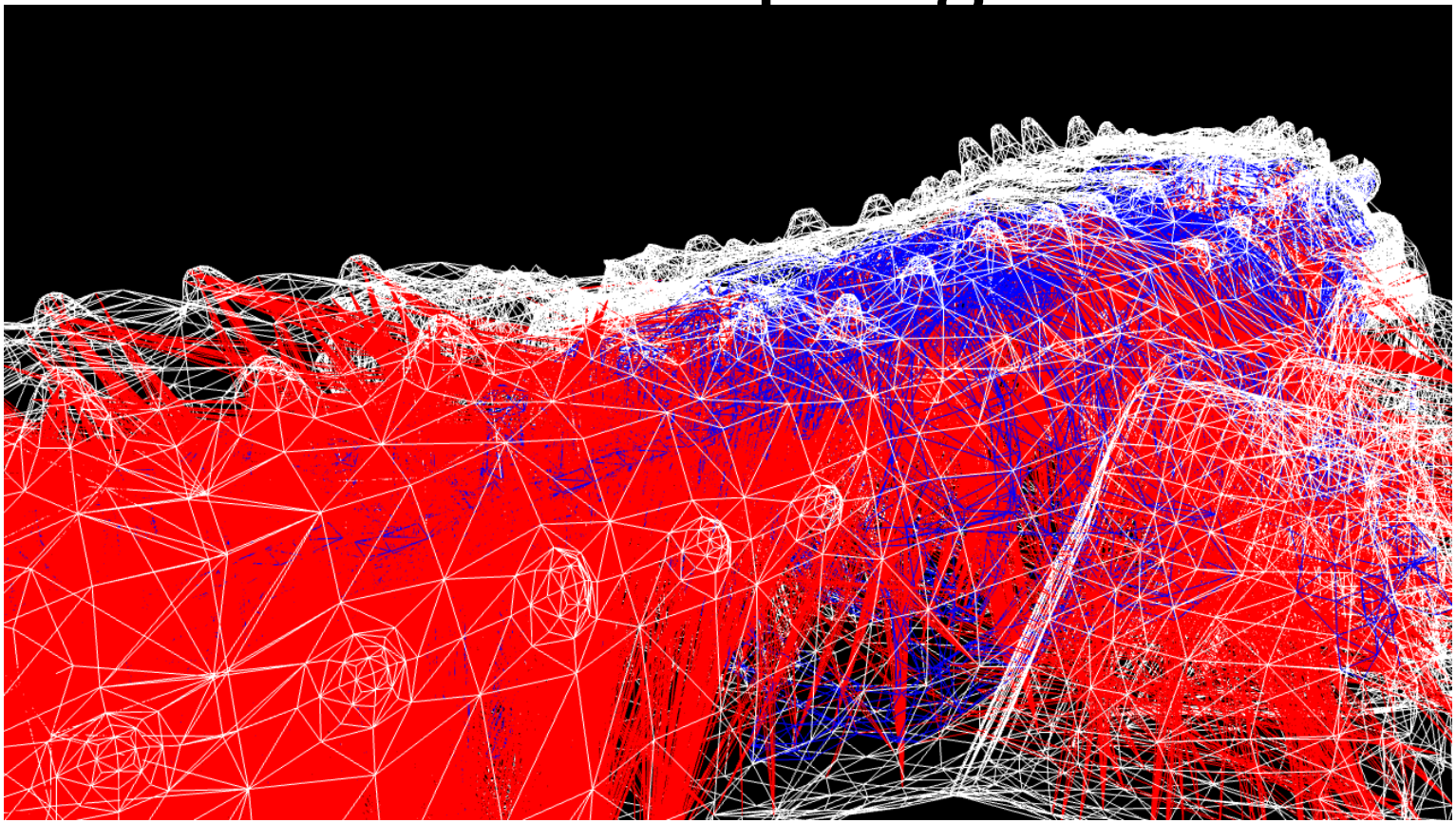
Bone Surfaces



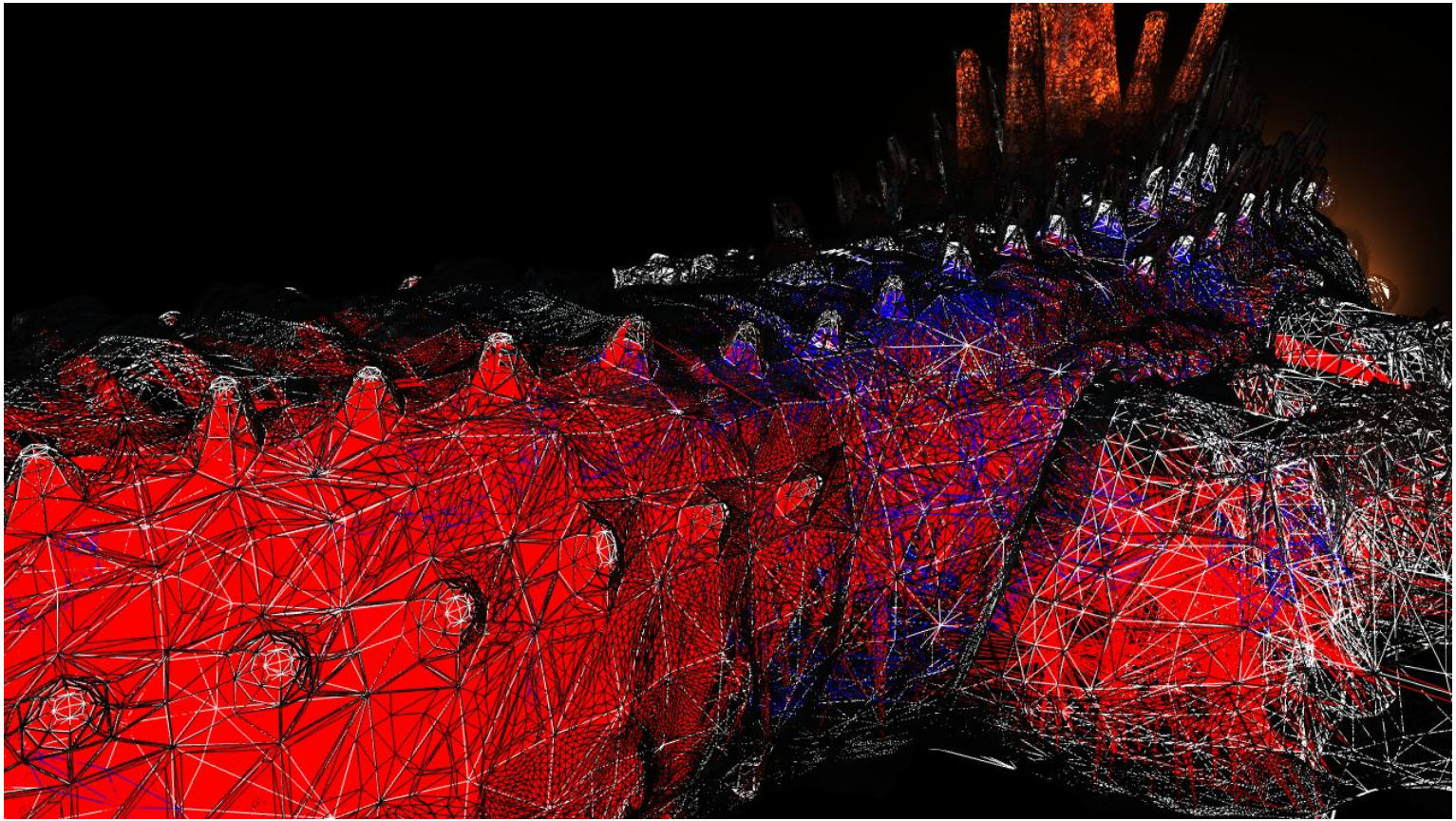
Surface Springs



Inner Springs



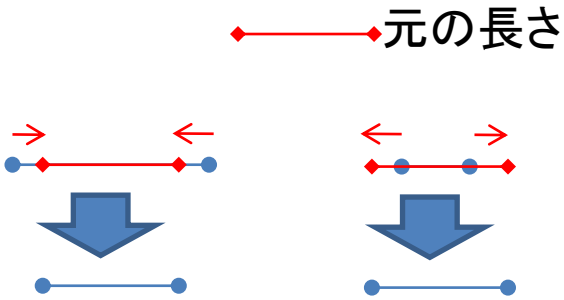
Tessellated Model



Verlet（ベルレ）物理

- 分子動力学の分野で古くからある一般的手法
  - SHAKE法など
- 質点の運動
  - Verlet積分： $\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + (\mathbf{x}(t) - \mathbf{x}(t-\Delta t)) + \mathbf{a}(t) \cdot \Delta t^2$
  - 位置を直接動かしても安定（位置と速度がずれない）
- バネ（拘束）

```
delta = x2-x1;
deltalength = sqrt(delta*delta);
diff = (deltalength-restlength)/deltalength;
x1 -= delta*0.5*diff;
x2 += delta*0.5*diff;
```

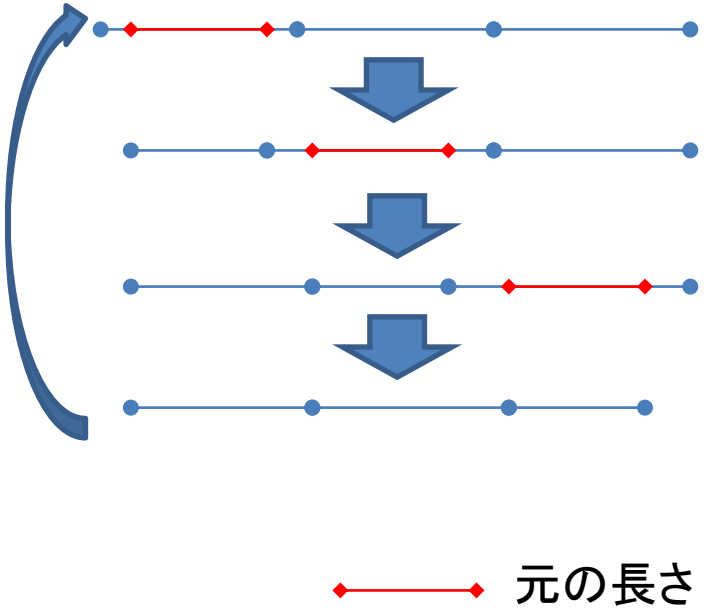


ゲームにおけるVerlet物理

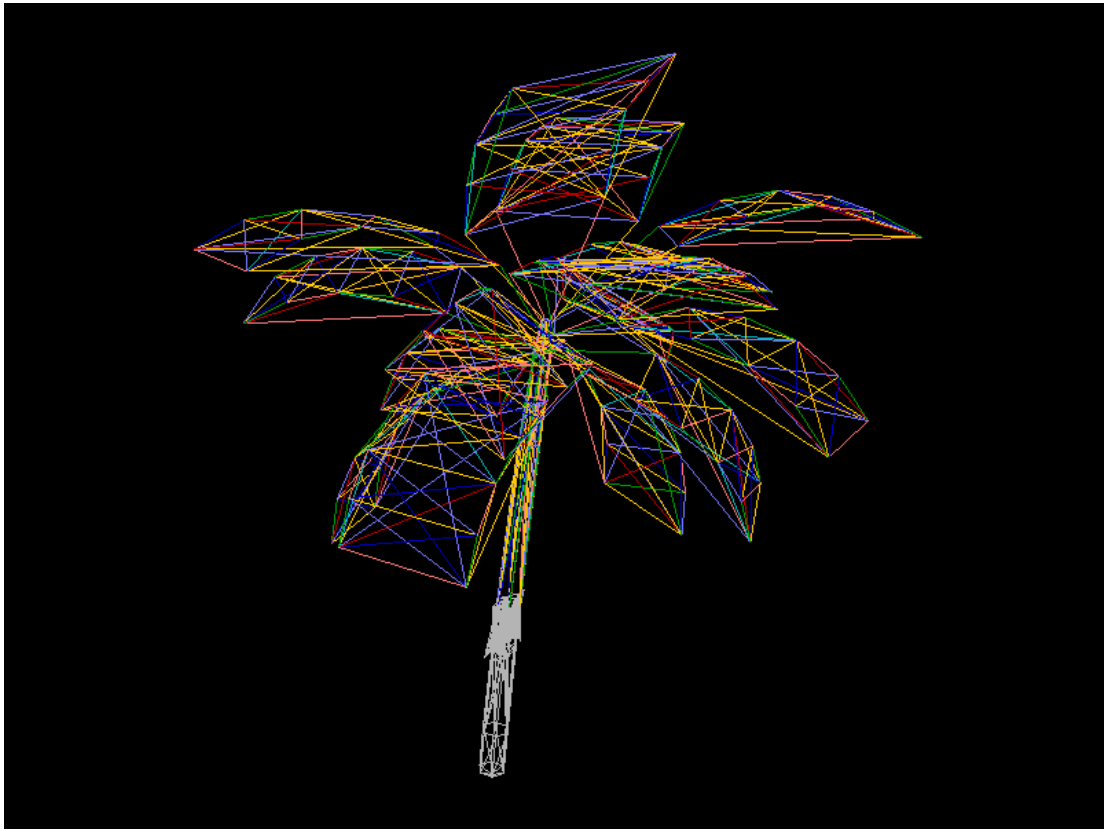
- ゲーム分野でも広く使われている
（IK／剛体／水面／ソフトボディ）
  - AB法（AnimeBody）
  - *fysix* engine(Hitman)
  - NVIDIAサンプルデモ／Position Based Dynamics
  - PixelJunk EDEN
  - Particle IK Solver（Spore）
  - Cloth（Alan Wake）

ソルバ：Gauss-Seidel(GS)

- Verlet物理で一般的に使われているソルバ
- 前ステップの結果を利用（収束を速める）
  - 計算依存→並列に処理できない
- バッチング
  - 独立したリンク集合（バッチ）に分ける
  - バッチ単位で並列処理
- PC/XBOX360のPixel Shader (DX9)版で使用
  - Surface Springs + Goal Springsのみ
  - バッチ数は最高8



バッチング

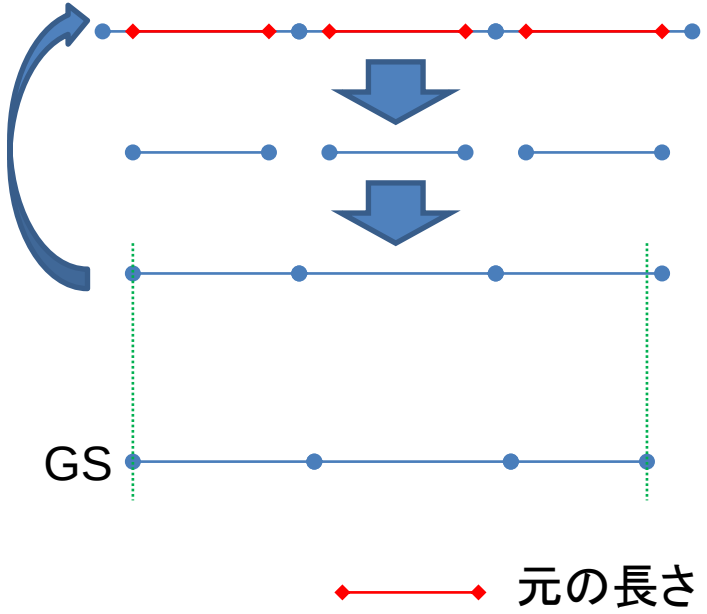


DX9 SoftBodyビデオ

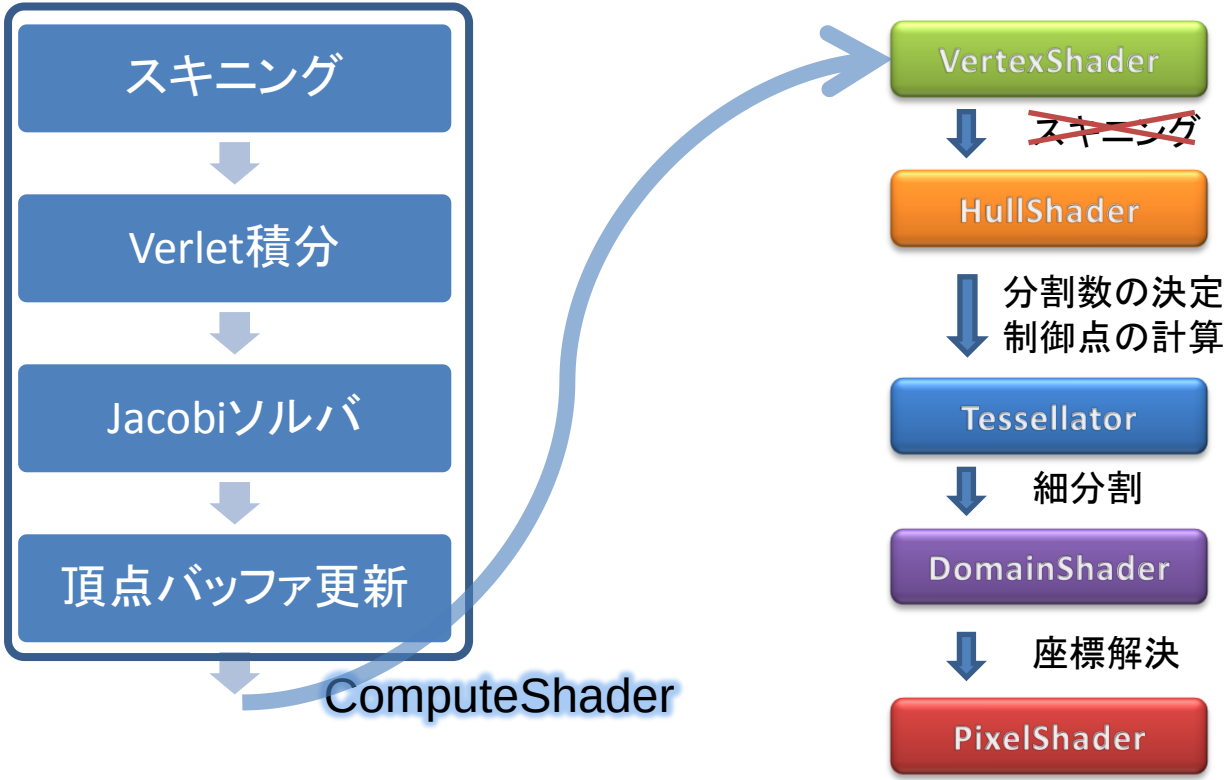
ソルバ：Jacobi

Inner Spring によるバッチ数（Dispatch数）増加への対策

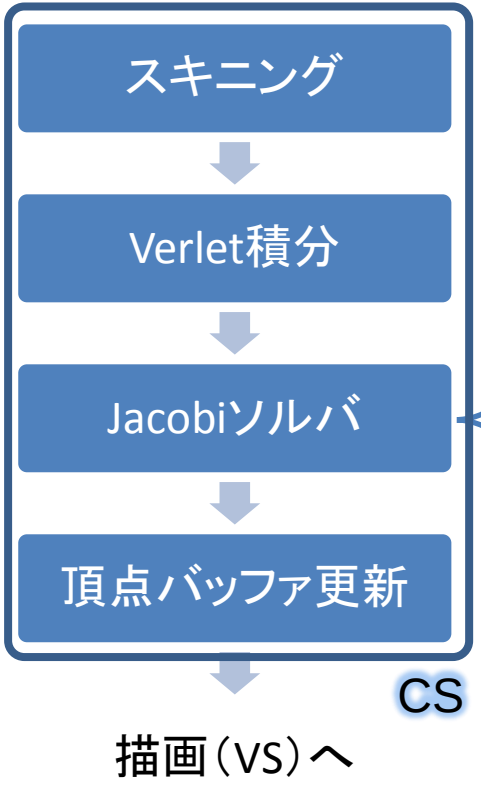
- 計算依存なし→並列処理むき
- Dispatchが1回
- 不規則メッシュや接続数が多いとき有効
- 前処理（バッチング）が不要
- 収束はGSより遅い
  - Goal Springsを使う場合あまり影響がない



CS実装：処理の流れ

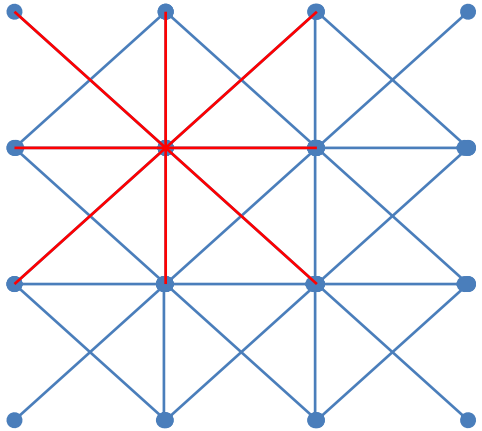

スキニング
Verlet積分
Jacobiソルバ
頂点バッファ更新
ComputeShader
VertexShader
スキニング
HullShader
分割数の決定
制御点の計算
Tessellator
細分割
DomainShader
座標解決
PixelShader

CS実装:Jacobiソルバ

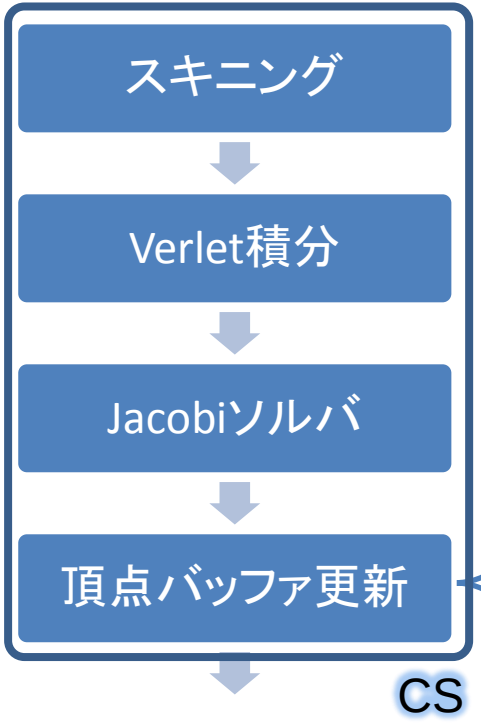


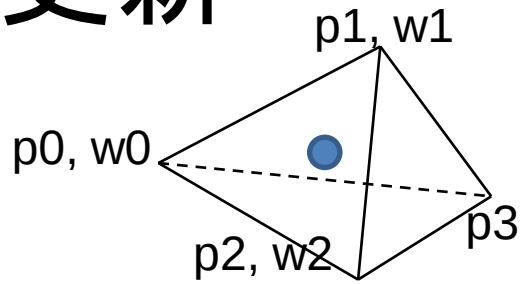
- 頂点中心(ギャザー)
  - 頂点ごとにリンク先のリストを持たせる
- シェアードメモリ
  - メッシュのタイリング
  - タイル内で頂点データをグローバル(遅い)からシェアード(速い)にコピー
  - シェアードメモリを参照して計算

各点は何度も参照される



CS実装：頂点バッファ更新





- Vertex BufferをCSで更新
  - Wave Particlesと同様の方法
  - DX9ではVTF
  - スキニング、シミュレーション、頂点位置計算の頂点処理がCSによって一度で済む
- 描画頂点や法線は四面体の重心座標系で算出
  - 三角形メッシュの場合は、体積一定の仮想四面体
  - 描画メッシュとシミュレーションメッシュが異なってもよい
  - 処理の統一

```
DrawVertex = p0 * w0 + p1 * w1 + p2 * w2 +
  p3 * (1.0f – w0 – w1 – w2);
```

Demo

まとめ

- コンピュートシェーダーは使える！
  - GPUで完結した処理、CPUリードバックなし
  - 実装のしやすさ
    - 記述が容易
    - 描画パイプラインと独立
  - 描画パイプラインとの連動
    - 頂点バッファを直接いじれる
    - テッセレーション／ディスプレースメントマップ
  - 最適化には従来のGPGPU的発想が必要
- デバッグ環境の問題
  - GPUベンダごとに異なる
  - PIX？

ご質問は？