NVIDIA
デモ開発チームの秘密

GeForce 6800

NVIDIAデモ開発チーム

“Nalu”の制作

Hubert Nguyen
William Donnelly
NVIDIA

ブロンド長髪の描画

ブロンド長髪の描画

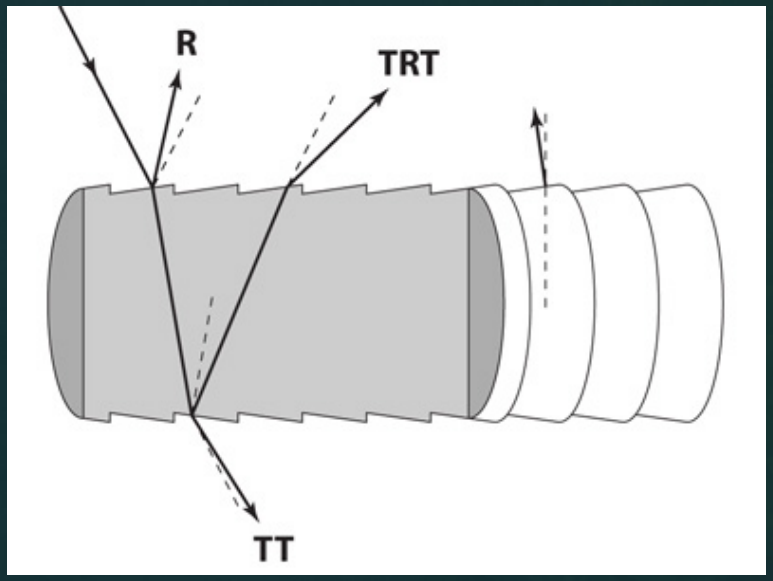
- 長髪
  - 動的なアニメーションが必要
    - ライティングを先に計算しておくことはできない
  - たくさん必要
    - そのためシェーディングが高速でなければならない
- ブロンド
  - みっつの可視ハイライト(黒髪ではひとつ)
  - 影が非常に際立つ

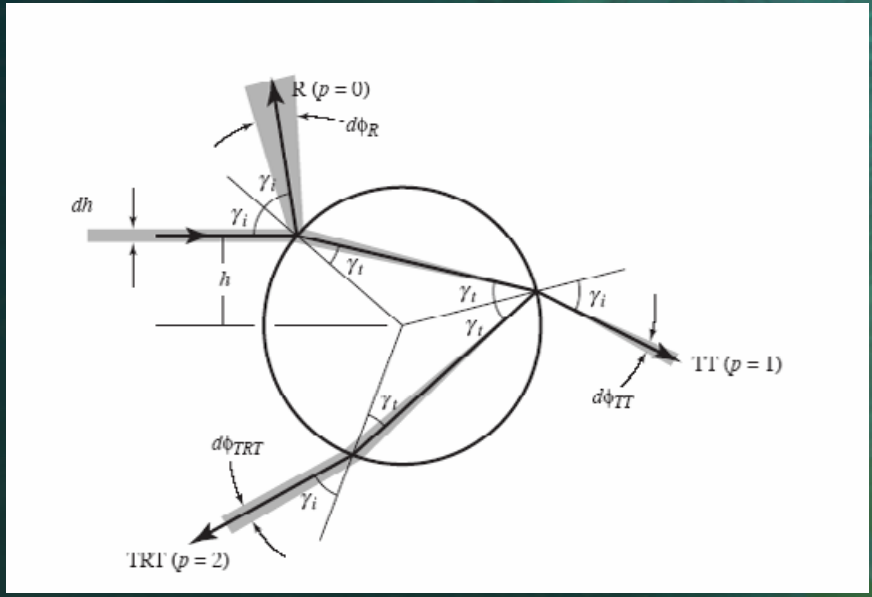
基礎となった論文

- “Light Scattering from Human Hair Fibers”
- By Steve Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan
- SIGGRAPH 2003

論文ではみっつのハイライトを考慮

- みっつの最重要項を考慮
  - R, TT, TRT





- 光の経路を書式に
  - Rは反射
  - T伝達

TTハイライト

- TT – 強力な前方拡散要素
  - 水中の髪には重要

反射モデル

- 髪は4次元関数
  - 2つの光源角度 + 2つの視点角度
- 低次元項へ織り込む
  - ```
    M_R (thetaH) * N_R  (thetaD, phiD)
    + M_TT (thetaH) * N_TT (thetaD, phiD)
    + M_TRT(thetaH) * N_TRT(thetaD, phiD)
    ```
- 2次元関数をテクスチャに保存
  - 複雑な数値計算より、テクスチャのほうが高速
  - ミップマップを使うことにより、シェーダーのエイリアスを除去
    ```

影

- “オパシティ・シャドウ・マップ Opacity Shadow Map” (OSM)ベース
- By Tae-Yong Kim and Ulrich Neumann SIGGRAPH 2001

なぜオパシティ・シャドウ・マップなのか?

- シャドウ・マップに対するオパシティ・シャドウ・マップ

  “光がここから遮られるか?”

  に対して

  “何パーセントの光がここから遮られるか?”

- そのため境界のアンチエイリアスやボリューム描画をサポートできる

- 普通のシャドウ・マップでは境界でエイリアス
  - 髪は100%境界!

KimとNeumannの結果

影無し

15層

オパシティ等式

$$T(z) = e^{-\int_0^z k(z')\,dz'}$$

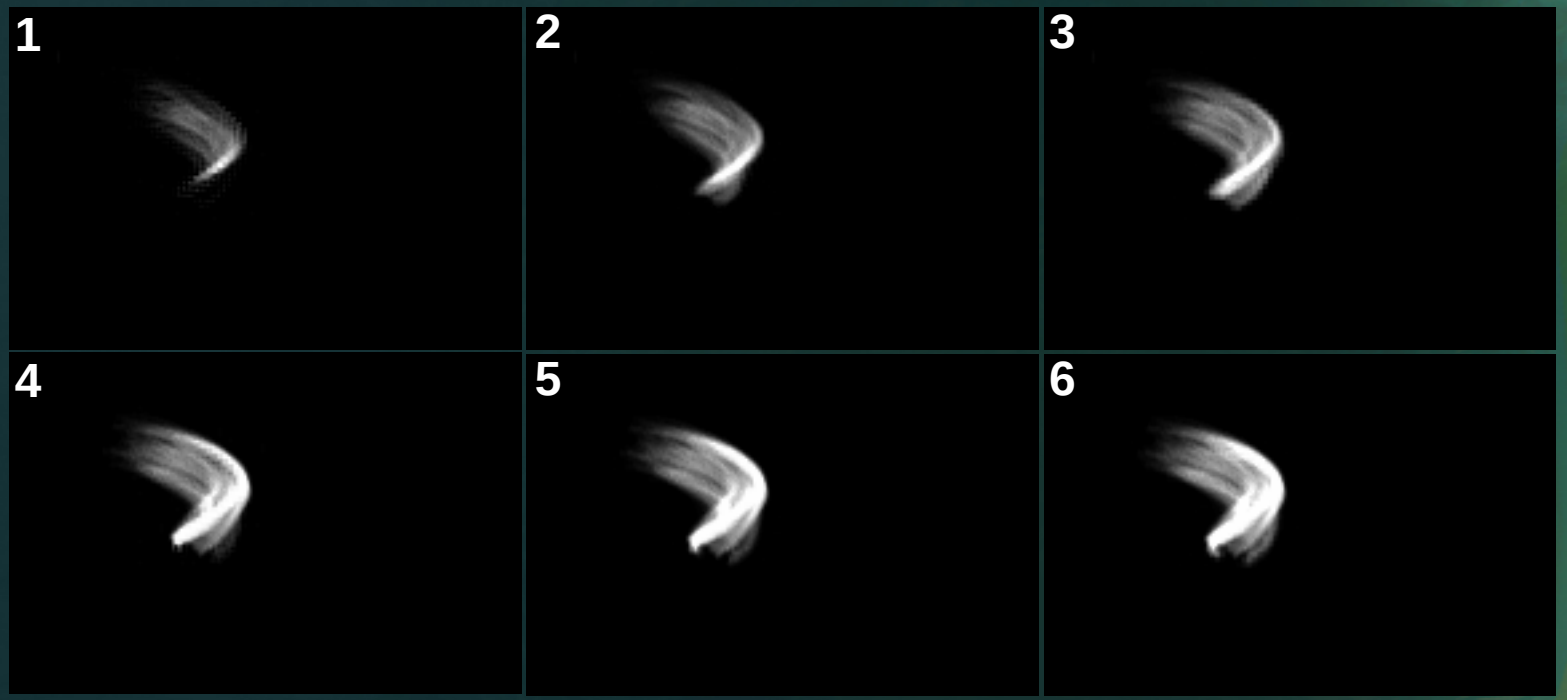
- **T(z):** 深さZまで到達する光の量
- 不連続な物質の場合 **(髪):**
  - 積分は、光源と影の投影される部位までの間の全ての髪の総和
- 加算ブレンドで総和を計算
  - “消失係数” Kで影の暗さを調整

オパシティ・マップを作成

- 髪を分断する**16**平面を選択
  - 髪の包囲球を使って等分に配置
- 各平面でそれぞれの髪ピクセルに対して
  - その髪ピクセルは平面より光源に近いか？
    - Yes: 髪を平面に加算
    - No: 何もしない

OSM作成

- 髪を**16**平面に描画
  - もともとの実装：16描画パス(RP)
- 髪の低い**LOD**を使用することができる



… up to 16

頂点シェーダーでの実装

- 光空間でも髪の位置を計算
  - Zバイアスを足して限られたZ解像度を補う
- 光空間での髪ピクセルの位置:
  - どのオパシティ・マップを使うか(z)
  - オパシティ・マップでのテクスチャ座標(x,y)

ピクセル・シェーダーでの実装

$$T(z) = e^{-\int_0^z k(z')dz'}$$

各平面での積分の値は分かっている

- 線形補間によって、間の値を計算
- 平面での値を線形補間して求める

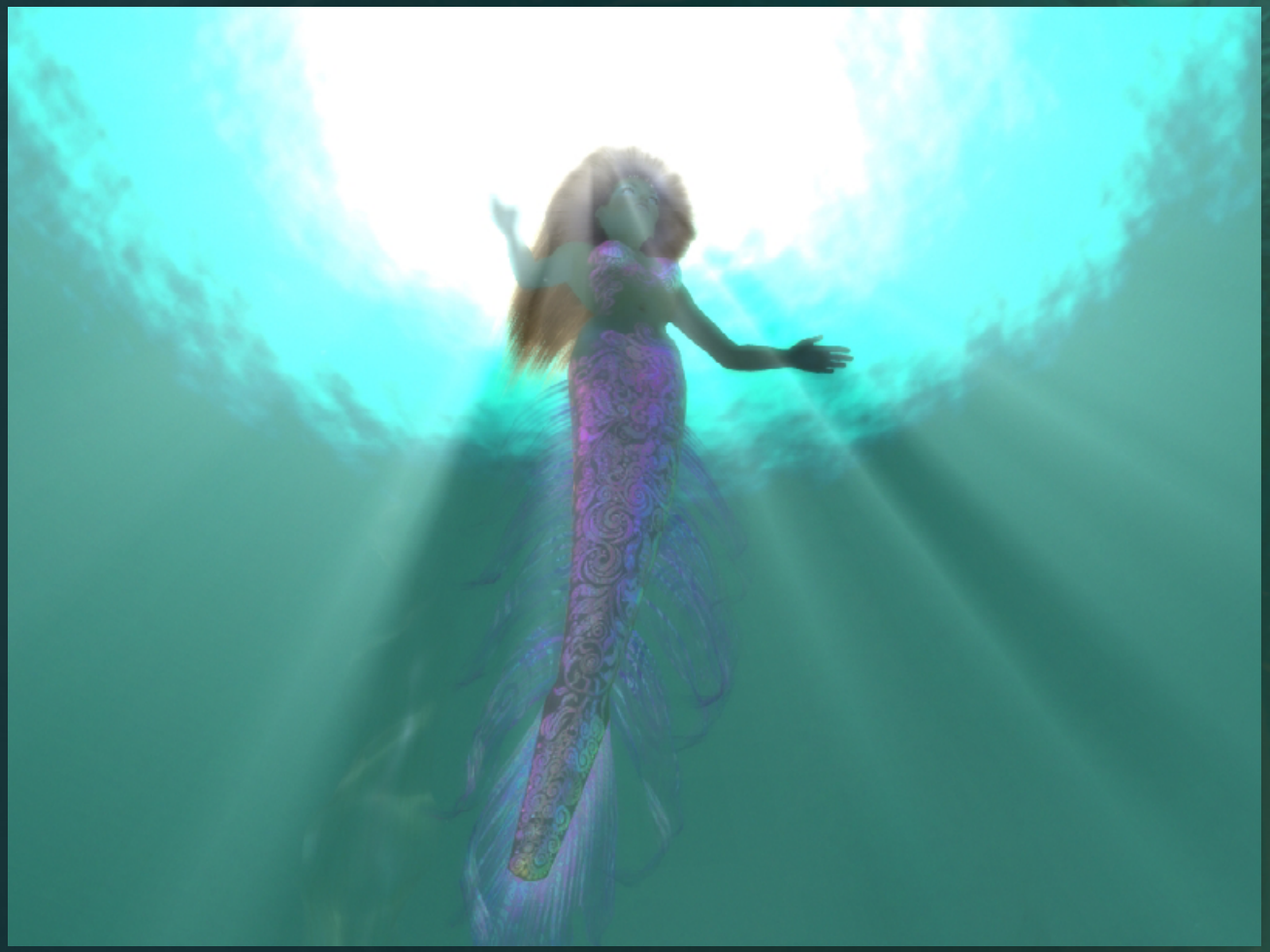
- オパシティは指数計算して求める

デモでは...

影無しの髪

影付の髪

光のシャフト：“ゴッド・レイ”



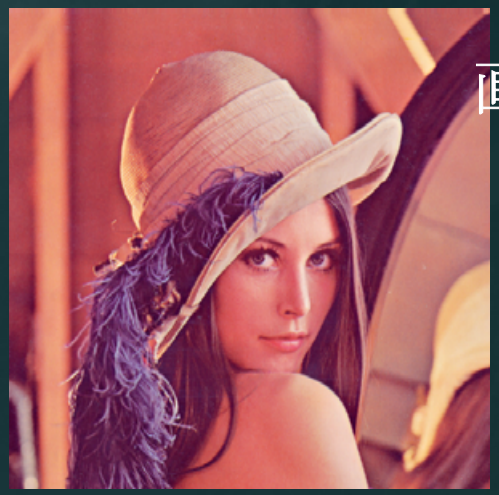
シャフトは放射状ブラー効果をもとにしている

放射状ブラー効果

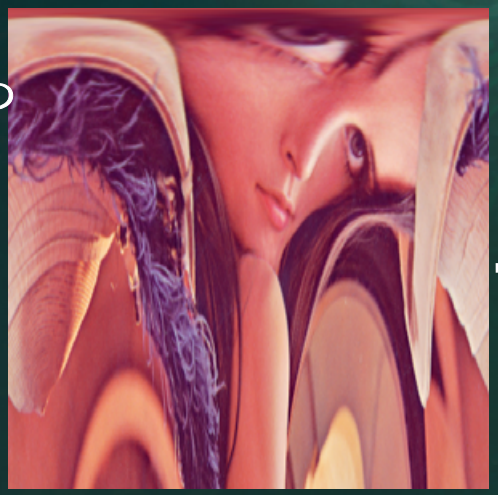
- 極座標に変換 (x,y)->(r,theta)
  - 位置 = (r,theta) でテクスチャ座標 = (x,y) になるようなグリッドを使用
- 垂直にブラーをかける
- 直交座標に戻す
  - 位置とテクスチャ座標を入れ替えただけの同じ変換を使用

直交座標から極座標へ

放射状ブラー効果: ビジュアル



直交座標から
極座標

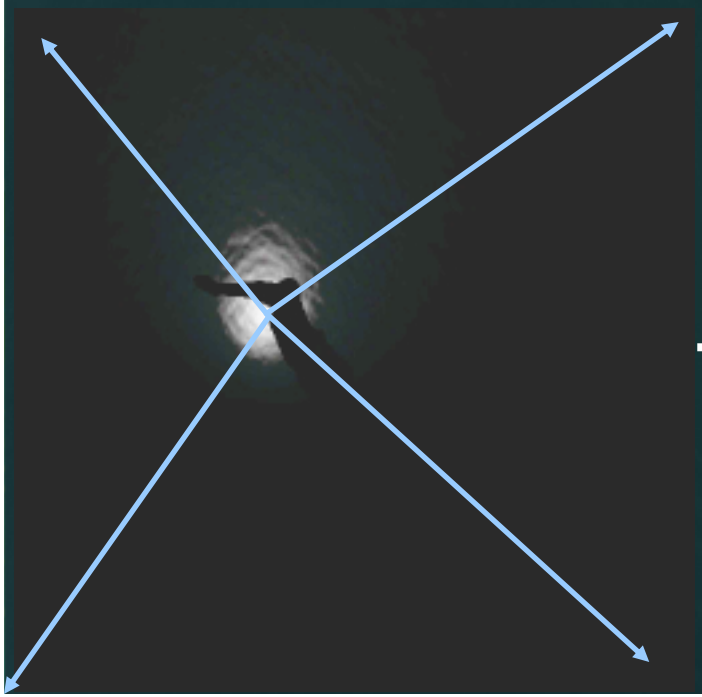


垂直
ブラー

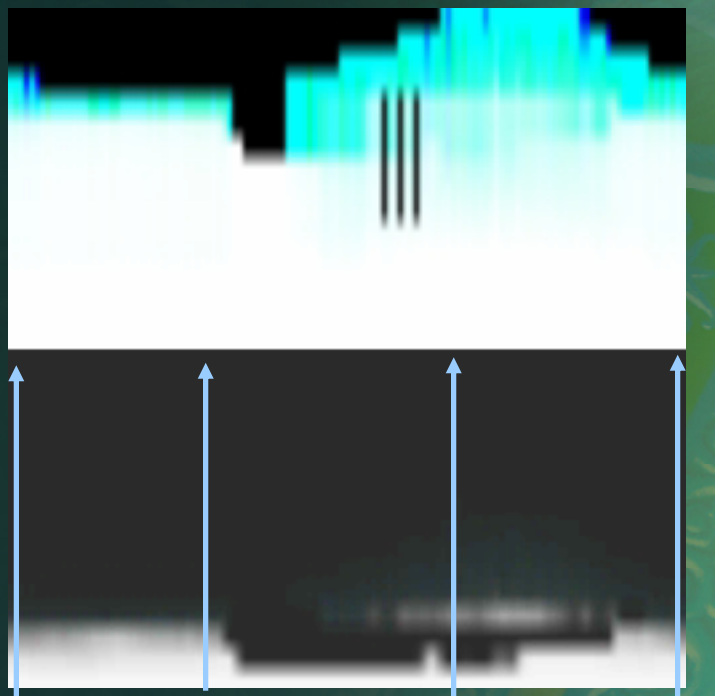
極座標から
直交座標

放射状ブラー効果: デモでは

- 明るい領域を描画
- 遮蔽物をアルファに描画
- 放射状にブラーしアルファから遮蔽物の分を引く



To Polar Coordinates



デモ中での“ゴッド・レイ”

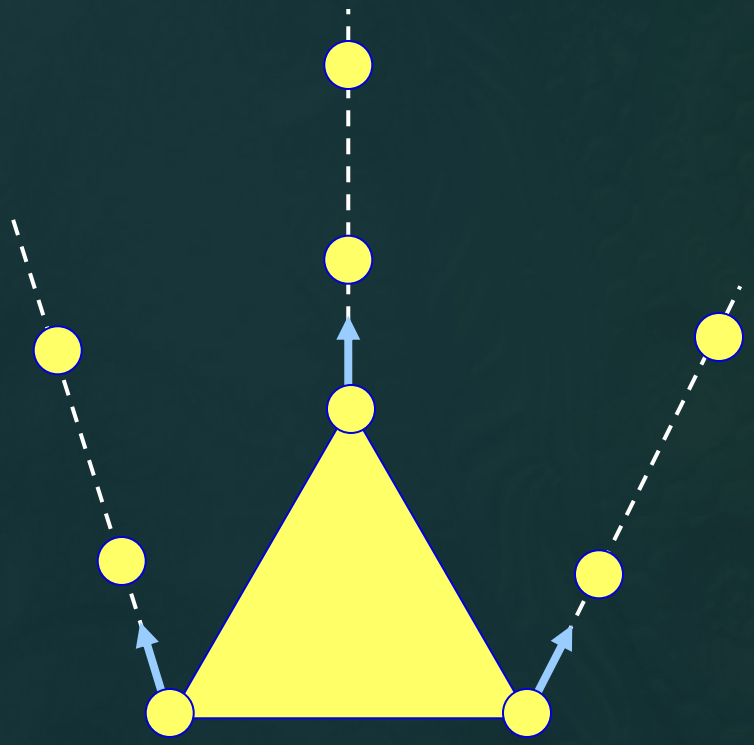
髪のジオメトリと動力学

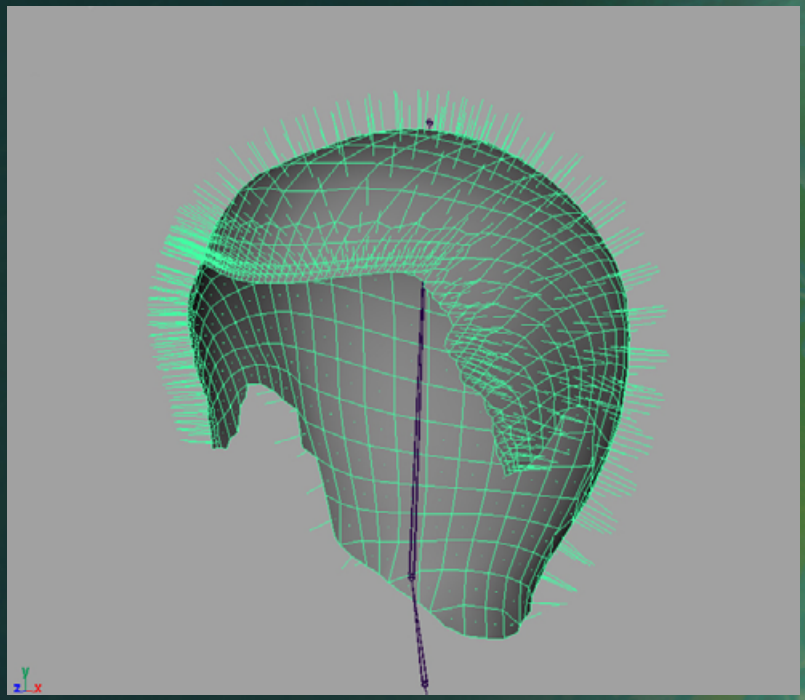
髪のジオメトリ：概略

- 762の制御髪によって4095の個々の髪が動作する
- “制御髪“
  - 動力学と衝突判定によって実際に動作する髪の集合
  - パーティクル同士が距離の制限を受けるパーティクル・システムをもとにしている
  - 参照になるジオメトリから作成
- “緻密な髪”のジオメトリは制御髪をならして補完することにより作成

髪ジオメトリ：配置と作成

- “制御髪” は専用のジオメトリから発生させる



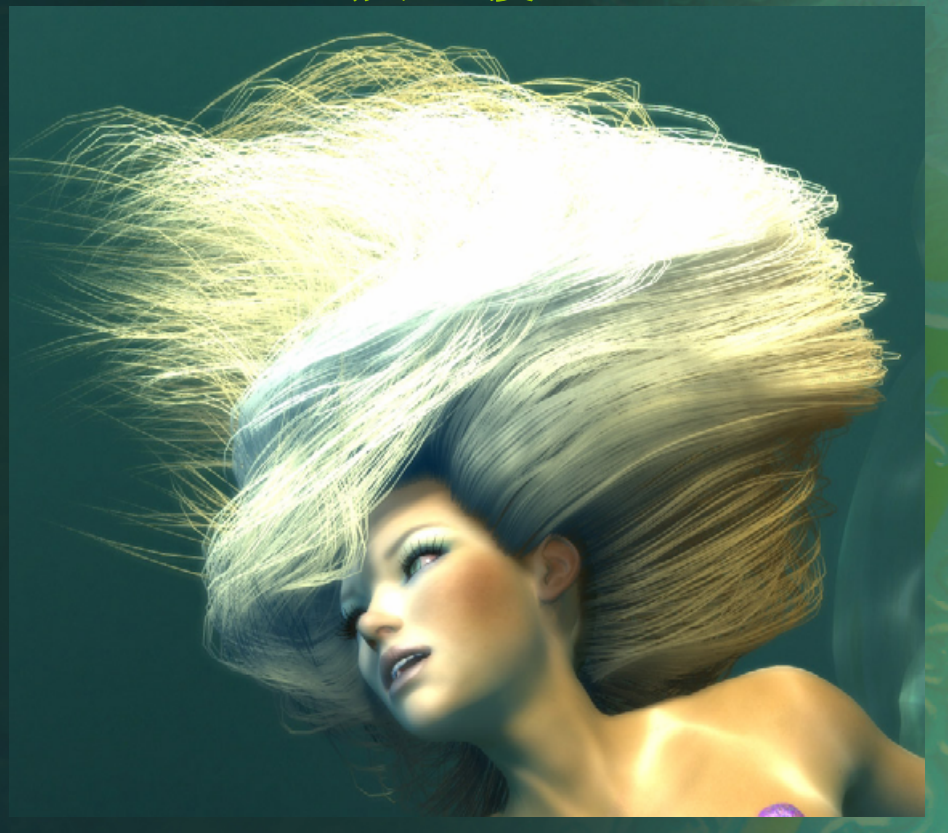


髪ジオメトリ: 制御髪 (左図)

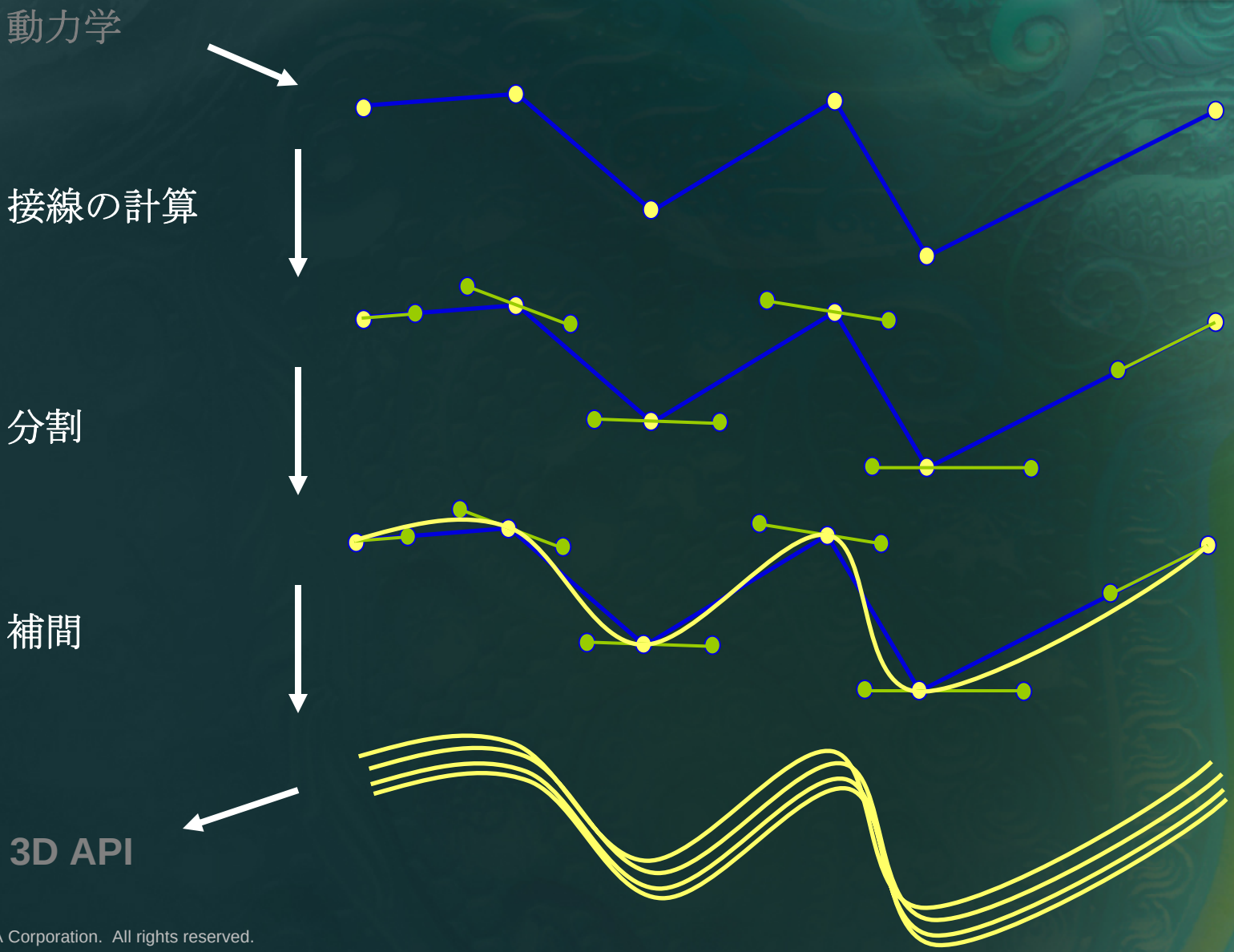
- 物理シミュレーション、動力学、衝突判定は制御髪に対して行われる

制御髪

緻密な髪



髪ジオメトリ: ライフサイクル (毎フレーム)



髪の動力学

- パーティクル・システムをもとにしている
- “Verlet”積分を使用
  - 前フレームの位置を利用して速度を計算

$$\mathbf{x}' = 2\mathbf{x} - \mathbf{x}^* + \mathbf{a} \cdot \Delta t^2$$

$$\mathbf{x}^* = \mathbf{x}$$

Reference: “**Advanced Character Physics**”
Thomas Jakobsen, IO Interactive, Denmark.

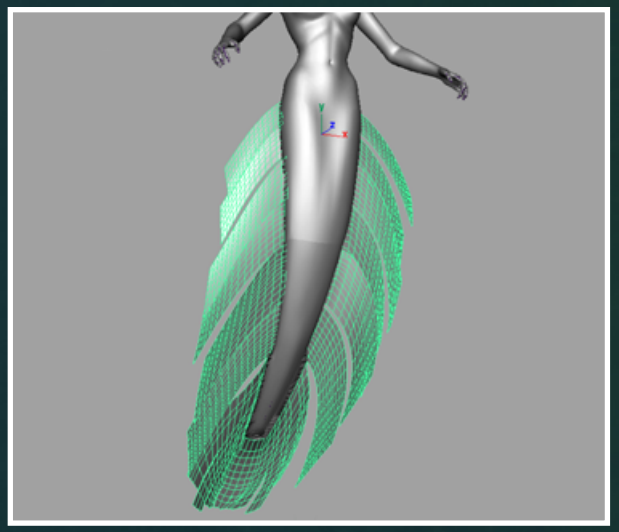
髪の動力学: 制限

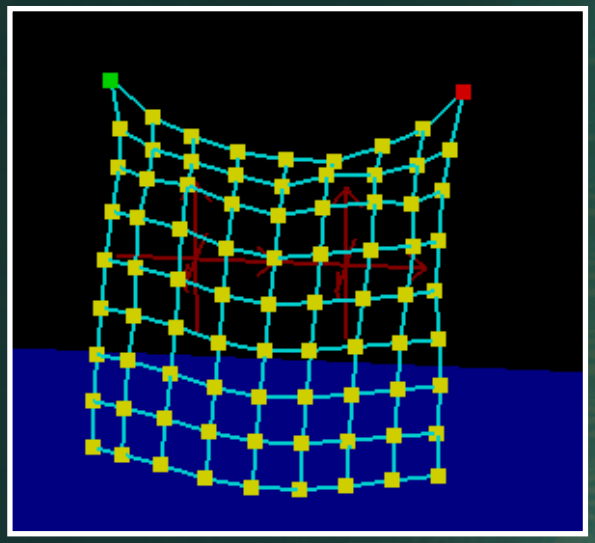
- 二種類の制限:
  - 無限の重量: “髪の根元”にあるパーティクルに作用する. これにより髪が頭皮に引っ張られる
  - 距離の制限: “制御髪”の各節の長さを一定に保つ

- この処理を繰り返し行うと、パーティクルは全体として要求どおりの長さに収束する

ヒレ

- ヒレは布のシミュレーション
  - 三角形のエッジに制限を設けることにより、どんなメッシュでも布に変えることができる





ソフト・シャドウ

- “テクスチャ空間での拡散 Texture Space Diffusion” (詳しくは “GPU Gems” に)がベース：
  - まず普通のシャドウ・マップの計算を行う
  - しかし、テクスチャ空間に描く
    - UV座標を頂点シェーダー出力の座標として使用する
  - テクスチャ空間での白黒シャドウ出力をブラー
  - キャラクタを描画する際、ブラーした結果をシャドウ比較に使用

セルフ・シャドウ: 視覚化

UV空間の描画

セルフ・シャドウ: 問題点

- キャラクタをUV空間に開く
- UVが連続でないところに継ぎ目が見える

将来の課題

- GPUでもっと多くの処理を
  - 物理シミュレーション
  - 当たり判定 (布のシミュレーションのように)
  - 曲線で分割
  - 法線と / 接線の計算
  - 髪の補間
  - その他なんでも :)

“Timbury”の制作

Ryan Geiss

NVIDIA

氏名: Timbury Entonin Mudgett

出生地: Cleveland, England, 1896

職業: 昆虫学者

生き方: “科学”を追及

使用された技術と効果

- ハイダイナミック・レンジ(HDR)ライティング
  - 非常に高解像度のライティング計算ができる
  - 自動ゲイン調整: 光の量によってカメラの絞りが変化
    - 明るい光源を直視 -> 普通の物体はシルエットに
    - “普通の” 白いトーン(例えばシャツ)が暗くなり、飽和していた白 (例えば太陽)は明るいまま
- fp16キューブマップの環境ライティング
- 後処理で画像を ”ソフトに” している
- アニメーション: スキニングとシェイプ・ブレンド
- 適応型のサブディビジョン・サーフェイス
- “ソフト” (複数タップの) シャドウ
- 光の屈折があるメガネ

フレームの描画

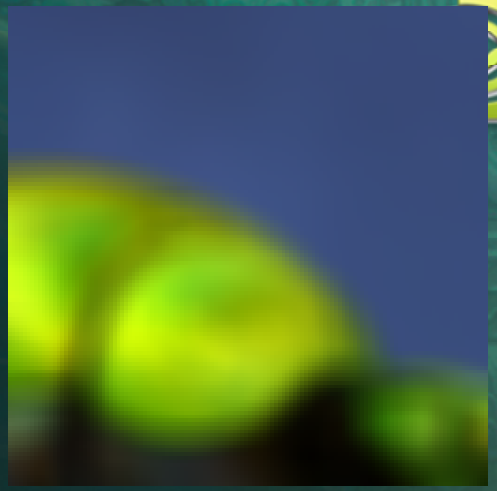
- シーンをテクスチャに描画
- 明るさを解析
- コピーを残して、ブラーをふんだんにかける
- ブラーしたものと、もとのシャープなものを合成(画像をソフトにするため)
- 明るさ解析の結果を使って暗くする
- 最終画像を画面に出力

描画 (詳細)

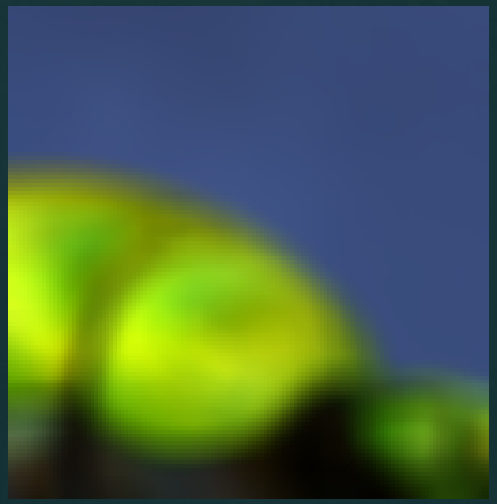
- まずシーン全体をfp16の大きなテクスチャに描画
- それを3x3のサンプルを用い、256x128のテクスチャに描画
- 256x128 fp16のテクスチャで、さらに4パス描画:
  - x方向ブラー
  - y方向ブラー
  - x方向ブラー
  - y方向ブラー
- これできれいな擬似ガウシアンの画像ができる(しかし高速!)
- 最後のパスで、ブラーされた画像をもとのシャープな画像と合成し、やわらかい映像の効果を出す

- オリジナル

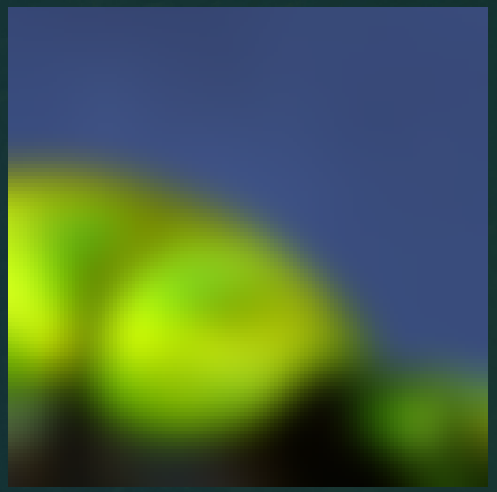
xブラー



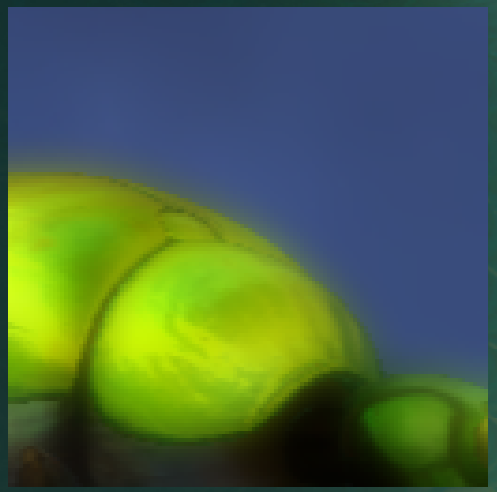
yブラー



- **x**ブラー **(2)**



yブラー **(2)**



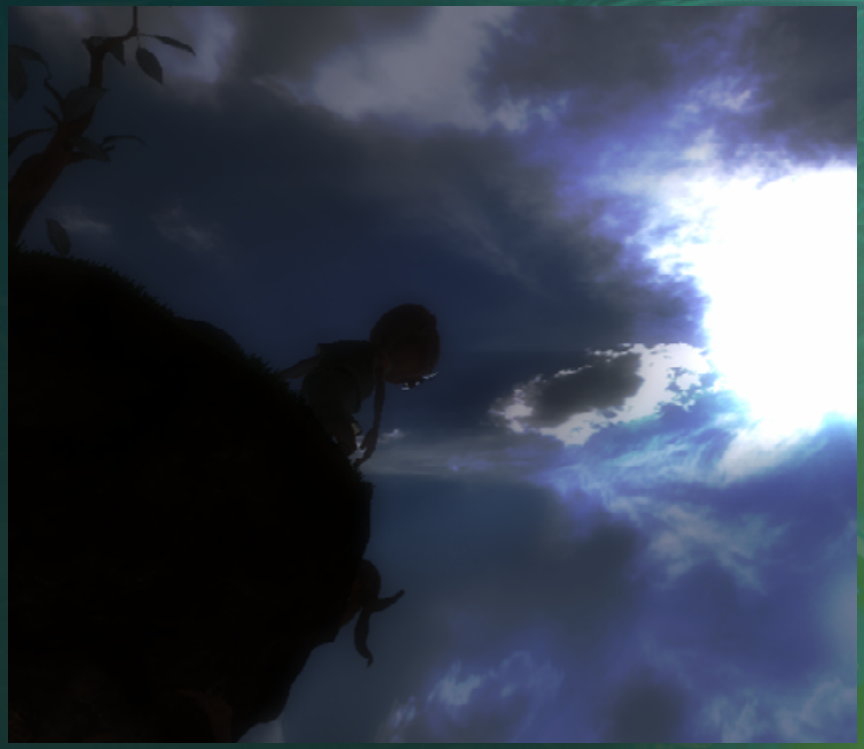
合成

自動ゲイン調整 (AGC)

- 家庭用ビデオカメラや人間の目が多すぎる光に対して行う動作の再現 – レンズを絞る
- シーンの中でとても明るいのは太陽
  - シーン内のほかのものの色より40倍程度明るい
- シーンを見回すと、流入する光の量によって絞りが調整されるのがわかる
- 太陽を直視 → 絞りが小さく、ほとんどのオブジェクトがシルエットに

AGC例

普通のライト



太陽を見る

fp16 HDRテクスチャの使用

- fp16入力テクスチャ:
  - 空のキューブマップ
  - 環境ライティングに使用する全てのキューブマップ(ディヒューズ1、スペキュラー 8)
- fp16描画ターゲット:
  - 主描画ターゲット
  - 全ての後処理用描画ターゲット
- fp16はハードディスク内ではIndustrial Light & Magicの**OpenEXR**フォーマットで保存されている
  - フリーなソース!: **http://www.openexr.org**
- 利点:
  - 高質なライティング計算と後処理の効果
  - AGC効果を可能にする

描画の流れ: AGCを追加

- カメラに流入する光の量を決定する:
- ブラー処理の途中から低解像度(8x8)の画像を作る
- 更に1x1の大きさのイメージにダウンサンプル
  - この1ピクセルだけに対して複雑なシェーダーを走らせる
  - 16のサンプルを使用し、線形補間(バイリニア・フィルター)で64テクセルからのサンプリング
  - 中心近くのサンプルに多めの重みをつける
  - それぞれのサンプルに対して輝度を求める:

    lum = dot(float3(0.3,0.48,0.22));
  - このフレームの平均光量:

    L = sum(lum values) / sum(weights)
  - float3(1/L, 1/L, 1/L)をシェーダーの出力として書き出す

描画の流れ: AGCの追加 (つづき)

- 最後の合成パスのシェーダー:
  - シャープな(もとの)イメージとブラーしたものの合成 (だいたい半々)
  - この結果を1x1テクスチャからの結果で掛ける(1/L).

線形と非線形のトーン・マップ

- このAGC実装では全てのピクセルの輝度の平均を見つけ、その逆数で画面全体の光量を調整する – 単純な構成
- 他の高度なトーン・マップの方法* では輝度の$\log_2$ の総和を求め、特別な方法で調整:
  - AvgLumLog = Σ $\log_2$ luminance
  - AvgLum = $2^{\text{AvgLumLog}}$
  - 最終パス: color' = grey * color / (1+color)
    - ここでgreyは "平均的な灰色" (約0.5).
- [Demo: 't'を押してトーン・マップモードに入る]

** from **Photographic Tone Reproduction for Digital Images** - Erik Reinhard, Mike Stark, Peter Shirley and Jim Ferwerda.*

log総和とトーン・マップ:

(+) 輝度の配分が良い
い

(-) コントラストが弱い
い

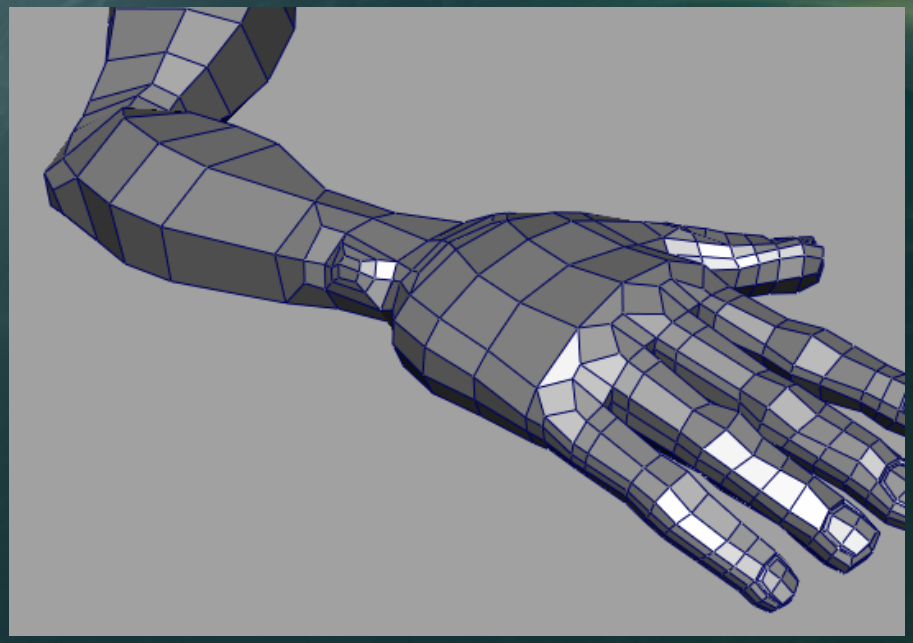
線形総和と
単純なスケール:

(-) 輝度の配分が良くな

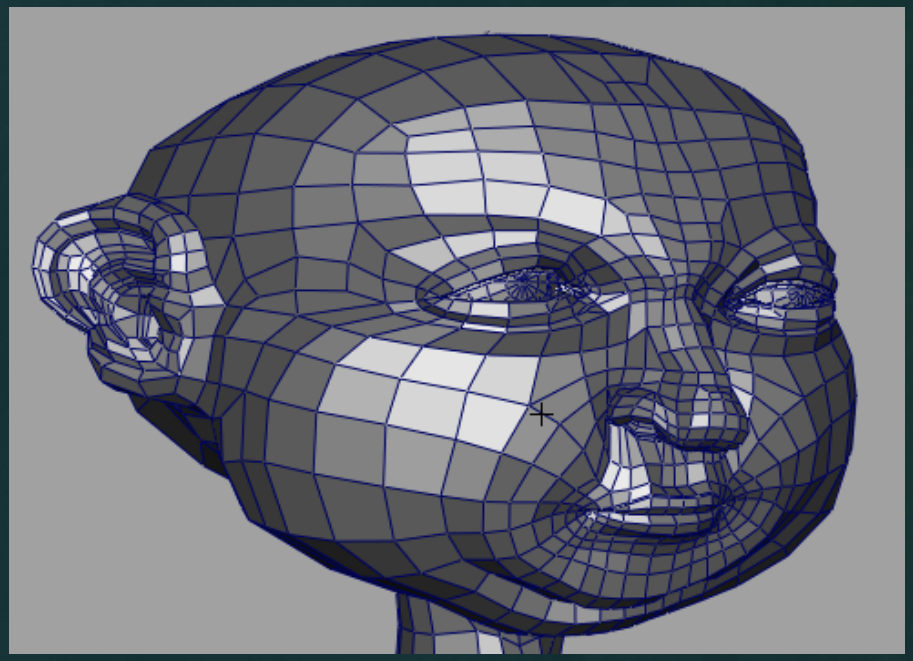
(+) コントラストが強

適応型のサブディビジョン・サーフェイス

- NVIDIAのMichael Bunnellによる
- 目標: 必要に応じてポリゴン分割を行い、メッシュを見栄え良く、効率よく描画
  - カメラがズームしたり、ひじが曲がったりするとポリゴンを足す
- もとのメッシュ: 四角形メッシュのほうが良いが、三角形メッシュでも可能
  - 解像度は低くなければならないが、モデルの特徴を全て表現していなければならない
- 最初にメッシュを全てCatmull-Clarkパッチに変換(四角形)
- 目標: フレームごとに、必要に応じてポリゴンを分割し、画面での誤差が1.0以下になるように調整



もとになる制御メッシュの腕と手



もとになる制御メッシュの頭部

適応型のサブディビジョン・サーフェイス (2)

- “誤差”はエッジが理想的な位置からどれくらい離れているか
  - 誤差はエッジの中心から理想的な位置にある湾曲したエッジの中心までのモデル空間での距離
  - この距離を画面空間に投影する – ピクセルでの誤差
- もし四角形の反対に位置するエッジの誤差が大きければ、この線に沿って分割 – もう一組のエッジも同じように処理
  - 腕の円柱などの場合、分割が必要な方向に多く分割が行われる

リンク

http://www.nzone.com/object/nzone_timbury_home.html

http://developer.nvidia.com/

http://www.openexr.org

未収録ショット

未収録ショット

“Clear Sailing”の制作

Joe Demers

NVIDIA

海洋シミュレーションと描画

- 現実感のある海と荒い波、その上を航海する船を描画することが目的
- このためには:
  - あらい海の波をシミュレート
  - 海面の分割
  - 海の水(と泡)を描画
  - 船の波による影響を物理シミュレートする
  - 船に波が当たるところでしぶきを上げる

海のシミュレーションと描画

海のシミュレーション

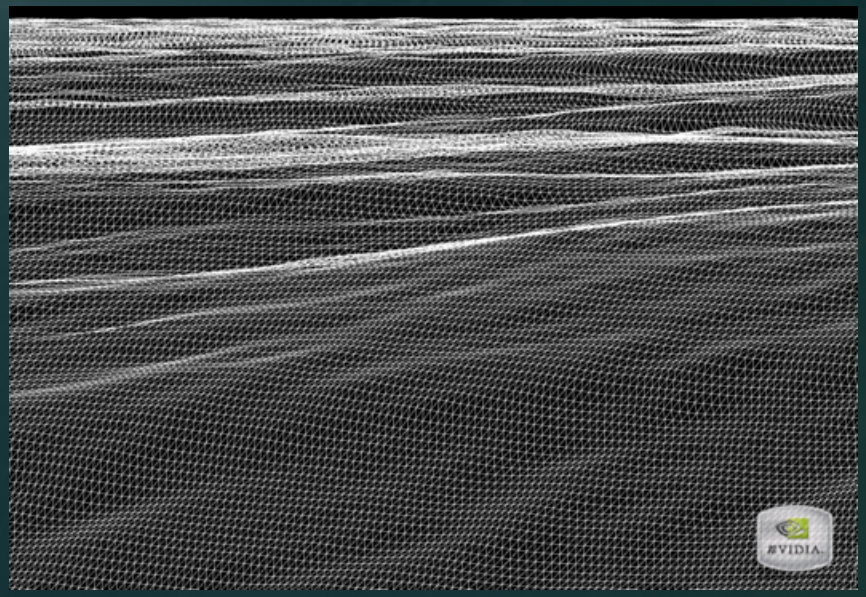
- ハイエンドのふたつの主流な方法はGerstner波モデルとFFTベースの統計モデル
- その簡潔さと繰り返し性の無さからGerstner波モデルを選択した
- Gerstner波モデルは水面上の点を波の移動する方向に対して平行な円で移動し、波が高いところに角が発生するなどの特徴がある
- Clear Sailingデモでは、150x150グリッド上で45のGerstner波を使えば画質とパフォーマンスの折り合いがちょうど良くなるとわかった

海面の分割

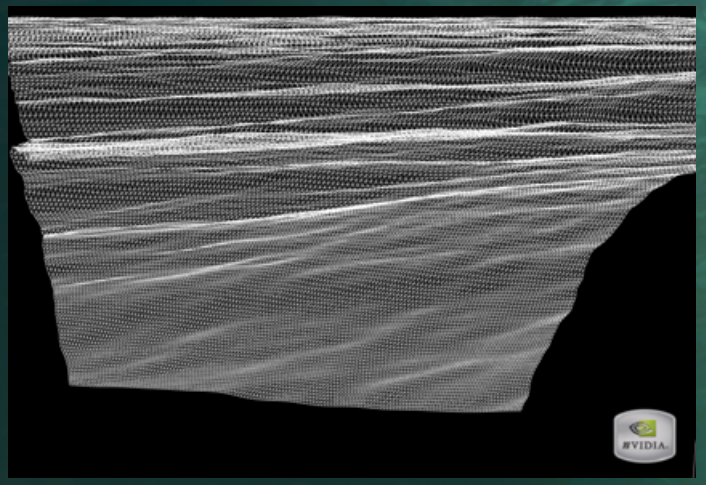
- 以前の方法ではほとんどの場合、ワールド空間に対して等間隔で頂点を並べてグリッドをタイルするか遠くを濃いフォグでぼかした
- このデモではカメラのビューポートと海面の交差面をグリッドで、カメラ座標で等間隔に分割した
- これで実際に視野に入っているジオメトリのみについてシミュレートと描画を行うことができ、遠くより近くをより細かく分割できる

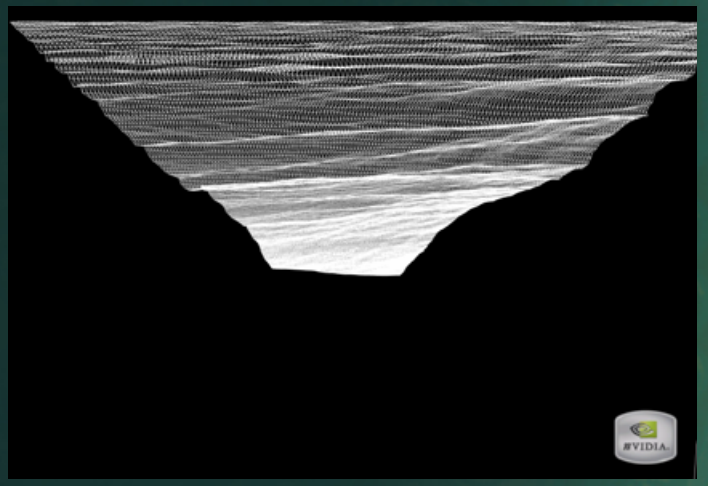
海面の分割

画面いっぱいの分割



カメラを引いて分割を確認





ジオメトリを固定してカメラを引くと実際描画されているジオメトリが良くわかる

海洋の描画

- 深い水を描画するにはいくつもの光源が必要
  - 太陽光(方向光源)は水面で反射
  - 空の光(キューブマップ・テクスチャの光源)は水面で反射
  - 拡散光(定数項)は水面の下から
- 簡単なフレネル関数を用いて、反射光と拡散光を合成することができる
- しかし、水面をちょうど良く見せるフレネル指数では、水を見下ろしたときに塗りつぶされてしまうので、全てのカメラアングルを考慮すると小さいフレネル指数を使う必要がある

船の写りこみ

- 船はふたつの反射光を遮蔽する
  - Zバッファのシャドウにより、太陽光を遮蔽する
  - 空の光は2Dのジオメトリ(船)に対してレイキャストを行い遮蔽する
- 反射に少しピクセル毎のノイズを入れて、波が更に高周波数であるように見せる

船の写りこみ

泡の描画

- 泡は実際は水面の上に描画される半透明の層
- 波に角が立つときと船の航跡で、泡が生成される(泡の層が不透明になる)
- テクスチャへの描画を加算ブレンドとともに用い、泡が時間とともに減衰するように調整した

泡の描画

船の物理シミュレーション

- 船は波に揺られるが、船から波への影響は無い
- 船は上下し傾くが、波の上を移動したり垂直を軸とした回転をしたりはしない
- 水上では、船は引力と風の影響を受ける
- 水面下では、船は摩擦と船の水面下の体積に比例する浮力の影響を受ける
- 物理シミュレーションを見栄え良くするには多くの時間を調整に費やさなければならないが、それはいろいろな遊び方があるということでもある
  - 波の速度を上げて物理シミュレーションのスケールを下げれば(物理シミュレーションの世界を大きくする)、船はおもちゃびように動く

ライティング

- 船は4つの光源から光を受ける
  - Zバッファでシャドウを受ける太陽の直接光
  - ディフューズとスペキュラのキューブマップによる空の光
  - 空からの広域光によるアンビエント光
  - 下からの青い方向光源による反射光
- 船の色にはディフューズ・マップ
- バンプ・マップとスペキュラ・マップで船により緻密な表現を与え、実際のジオメトリより複雑に見せる

ロープと帆

- 帆は両面で、太陽が後ろにあるときやわらかく光る
- 太陽に向かうときのシェーダーと、太陽と反対を向くときのシェーダー同じ値にならなければ、法線が太陽のほうを向いたときにつなぎ目が見える
- ロープは線で描画し、カメラからの距離で太さを変えている
- 線の太さが1ピクセル以下になったときには、透明度を加えて、アンチエイリアス効果を出すことができる
- アルファで収束の方法を用いているので、オブジェクトの並べ替えは必要なし

ロープと帆

飛沫と水煙

- 飛沫
  - 大砲が水面に当たったときにパーティクルを放出する
  - たくさんの水滴を描いたテクスチャの、大き目のパーティクルを描画
- 水煙
  - 船の竜骨や梁が水面を突き抜けたとき、その梁の間にパーティクルを放出
  - 速い動作ではより遠くまで(高く)パーティクルを飛ばす

飛沫と水煙

煙と破片

- 煙
  - 船の大砲が射出されるとき、射出の方向にテクスチャ・アニメーション付の大きめなパーティクルを放出(3Dテクスチャ使用)
  - 煙のパーティクルはすばやく動き始めるが、すぐに動きにダンプがかかり、ゆっくりと拡大して消える
  - 煙のパーティクルは下半分を暗くすることにより“明るくなる“ – 安っぽい方法だが見栄えは良い
- 破片
  - 砲弾が船に当たったとき、たくさんの小さな三角形を作る
  - 簡単で高速なVerlet動力学で動かす

煙と破片

後処理

- HDRっぽい輝き
  - 単純に8ビットの単チャンネルでHDR効果を表現
  - 明るすぎる成分をアルファに書き込む
  - アルファをブラーしてシーンに合成
- 多少のフォグがあったほうが、水と空と船をまとめるのに良い

後処理

質問?

リファレンス

- Jerry Tessendorf. “Simulating Ocean Water”. SIGGRAPH 2001 Course notes
  - course notes no longer online, here are the slides:
  - http://probe.ocn.fit.edu/slides2001.pdf
- Hinsinger, D., Neyret, F., and Cani, M.P. “Interactive Animation of Ocean Waves”, Symposium on Computer Animation, 2002
  - http://www-imagis.imag.fr/Publications/2002/HNC02/index.gb.html