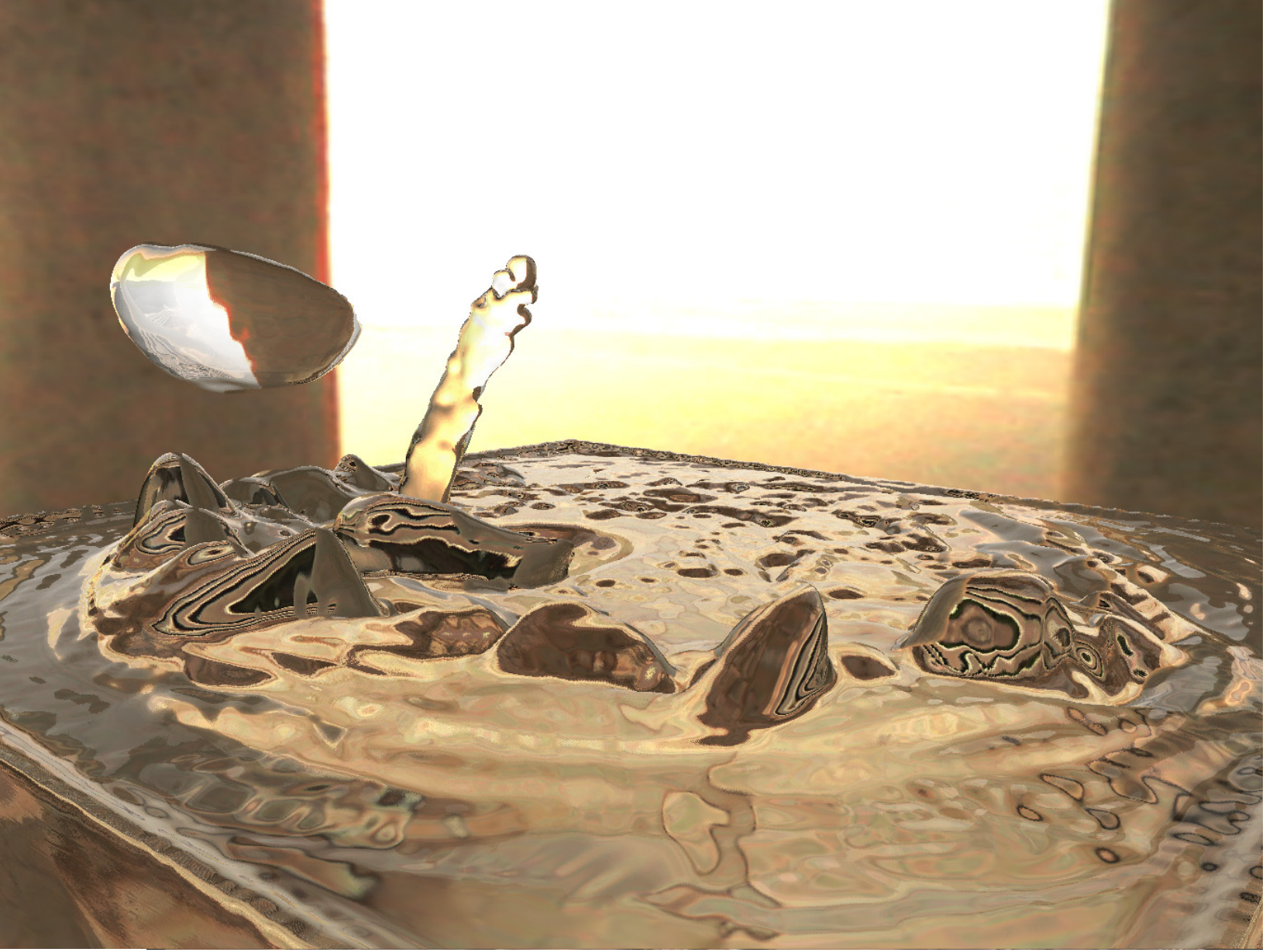
流体力学による煙、炎、水

Chris Kim デベロッパー テクノロジ エンジニア

2007年9月

NVIDIA SDK 10のサンプルにおける煙

NVIDIA SDK 10のサンプルにおける炎



NVIDIA SDK 10のサンプルにおける水

概要：

- 3D流体シミュレーションが重要である理由
- 流体シミュレーションの基礎
- 任意の境界を持った動的障害物
- 煙と炎のレンダリング
- 水のレンダリング

重要な理由

- リアルタイムな流体への現在のアプローチ
  - パーティクル システム
  - ビデオ テクスチャ
  - 手順: Perlinノイズ、炎、水面モデル
  - 事前演算/モデルの縮小
- 制限されたインタラクションの可能性の提供
  - 余り現実的ではない
  - 事前に演算されている
  - まったくインタラクションがない

- 流体シミュレーションが提供するもの
  - 煙、水、および炎の統合されたフレームワーク
  - ダイナミック オブジェクトとの現実的なインタラクション
  - リアルタイム（一部トレードオフあり）

なぜ今なのか？

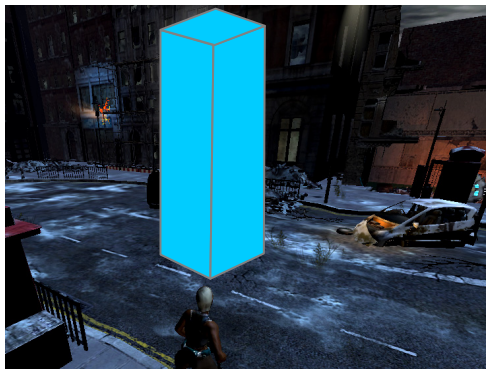
- 過去の作業が道を開く
  - Jos Stam 2003: ゲームのためのリアルタイムな流体
  - Harris 2003、Sander 2004、GPUにおける流体シミュレーション
- GPU演算能力の向上：
  - ハイエンドGPUの実用性のみを必要とする多くの操作（2006年以降）
- GPUメモリの増大
- Direct3D 10の新機能
  - 3Dテクスチャへのレンダリング、ジオメトリ シェーダ、ストリーム アウト

「**Hellgate: London**」ゲームの煙

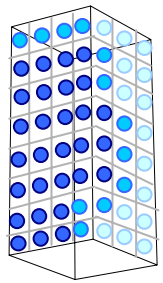
概要

シーン

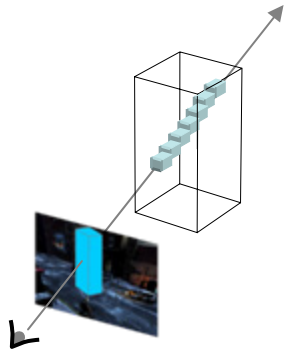
シーンのトップに合成



流体を配置する場所を
決定



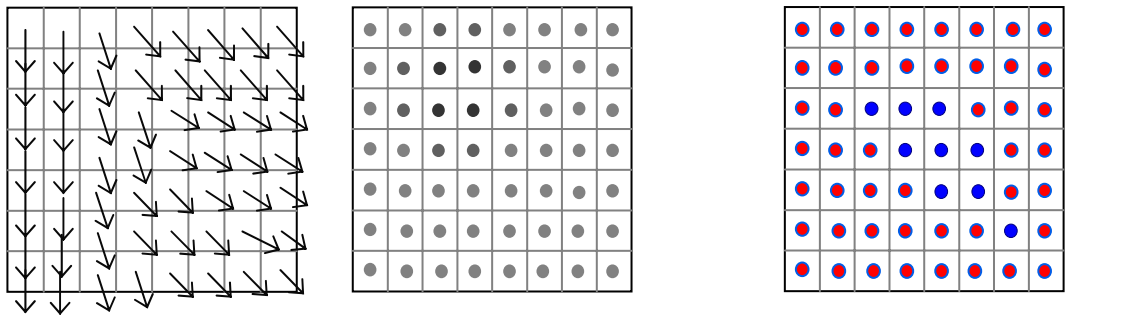
スペースを
離散化し
シミュレート



レンダリング

流体シミュレーションの基礎 - 1/3

- 流体の動きをシミュレート
  1. シミュレーション ドメインを決定する(スペース領域)。→グリッドに離散化する。
  2. 各ポイントで流体状態変数を決定する 。各グリッド セルでは以下をストアする。
     - 速度(ベクトル)と圧力(スカラー)
     - 煙または炎の密度(スカラー)
     - 水と液体のLevel Set面(スカラー)
  3. 時間をかけてこれらの *フィールド*を展開する



流体シミュレーションの基礎 - 2/3

- ***Navier-Stokes方程式***を使用して、速度フィールドの展開を計算する。この構成は以下のとおりである。

運動量

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{f}$$

質量
（非圧縮）

$$\nabla \cdot \mathbf{u} = 0$$

流体シミュレーションの基礎 - 3/3

Navier-Stokes方程式

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{f}$$

$$\nabla\cdot\mathbf{u} = 0$$

Helmholtz-Hodge 直交分解定理
[『Stable Fluids, Jos Stam, 1999』を参照]

$$\frac{\partial \mathbf{u}}{\partial t} = \text{Project}\left(\mathbf{f} - (\mathbf{u}\cdot\nabla)\mathbf{u}\right)$$

外力の付加　移流

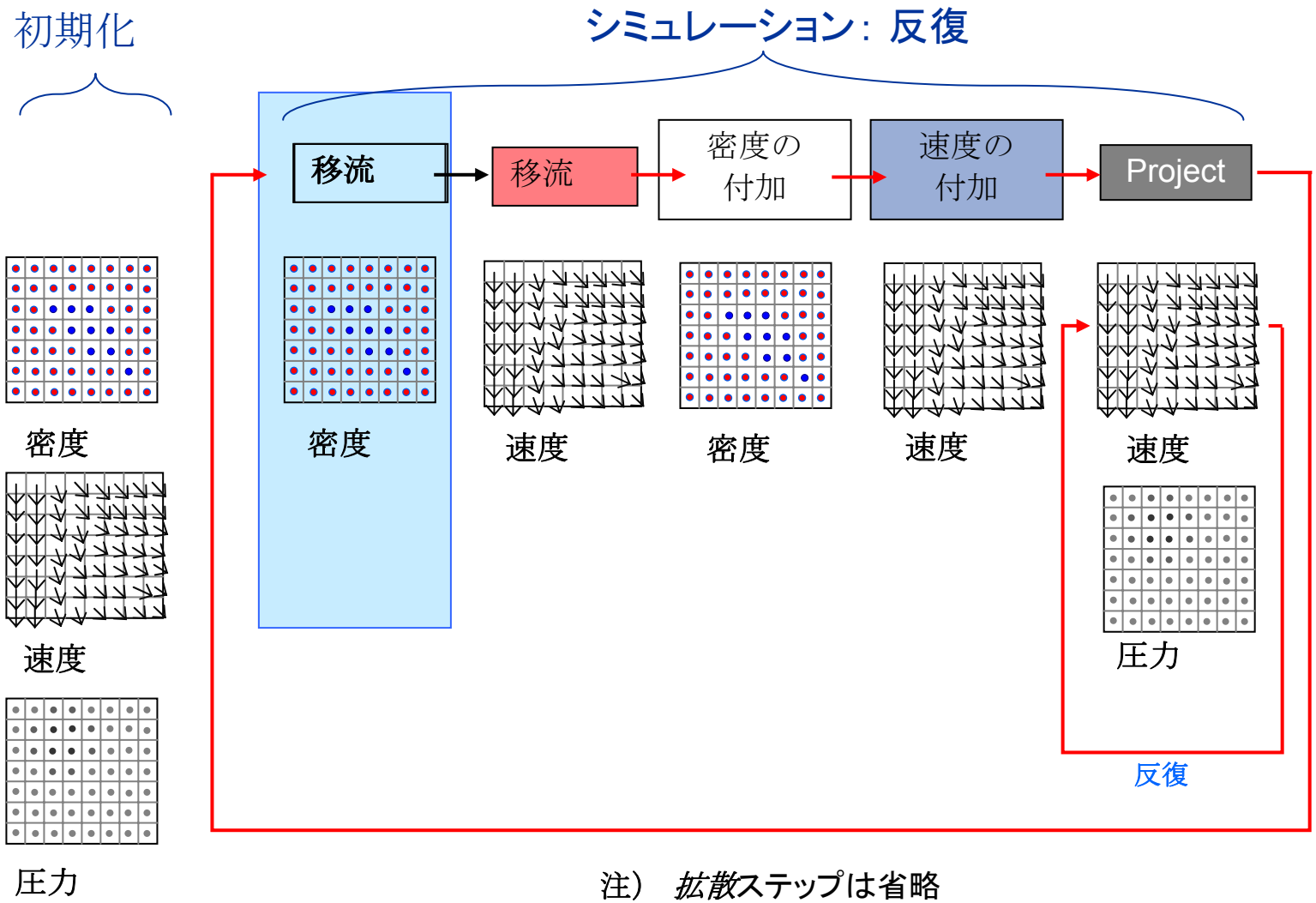
$$\nabla^2 p = \nabla\cdot\mathbf{u}$$

$$\mathbf{u}' = \mathbf{u} - \frac{\Delta t}{\rho}\nabla p$$

- 圧力の射出：
  - 方程式：“*「圧力のラプラシアン = 速度のダイバージェンス」*
  - *「速度のダイバージェンス」*がわかれば、圧力が求められる
  - *ポアソンのような方程式*→ 反復 ソルバ（Gauss-Seidel法、Jacobi法、共役勾配法）を使用して求める
  - 求めた後、速度から*「圧力のダイバージェンス」*を引く
  - 結果：非圧縮性を考慮し、境界条件を満たす

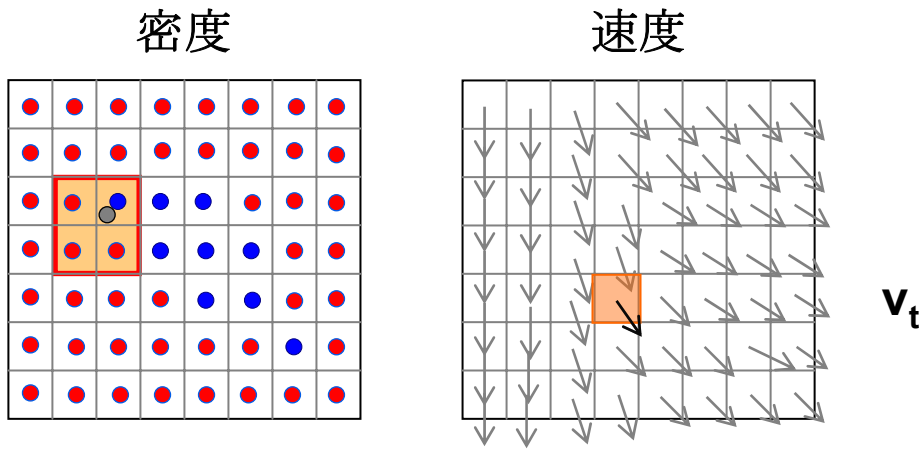
•圧力はゼロに初期化される（最初のタイムステップで仮定する）
•次のタイムステップは、前のタイムステップのソリューションから開始する

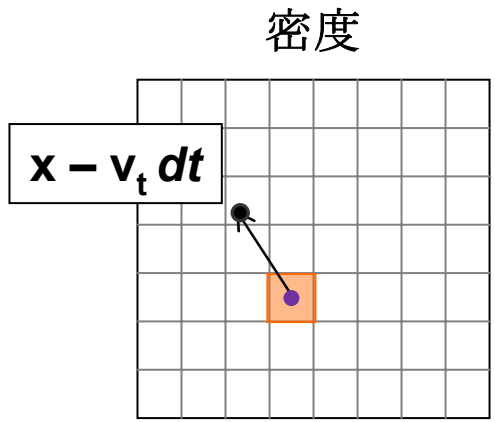
流体シミュレーションのステップ



注） *拡散*ステップは省略

密度の移流





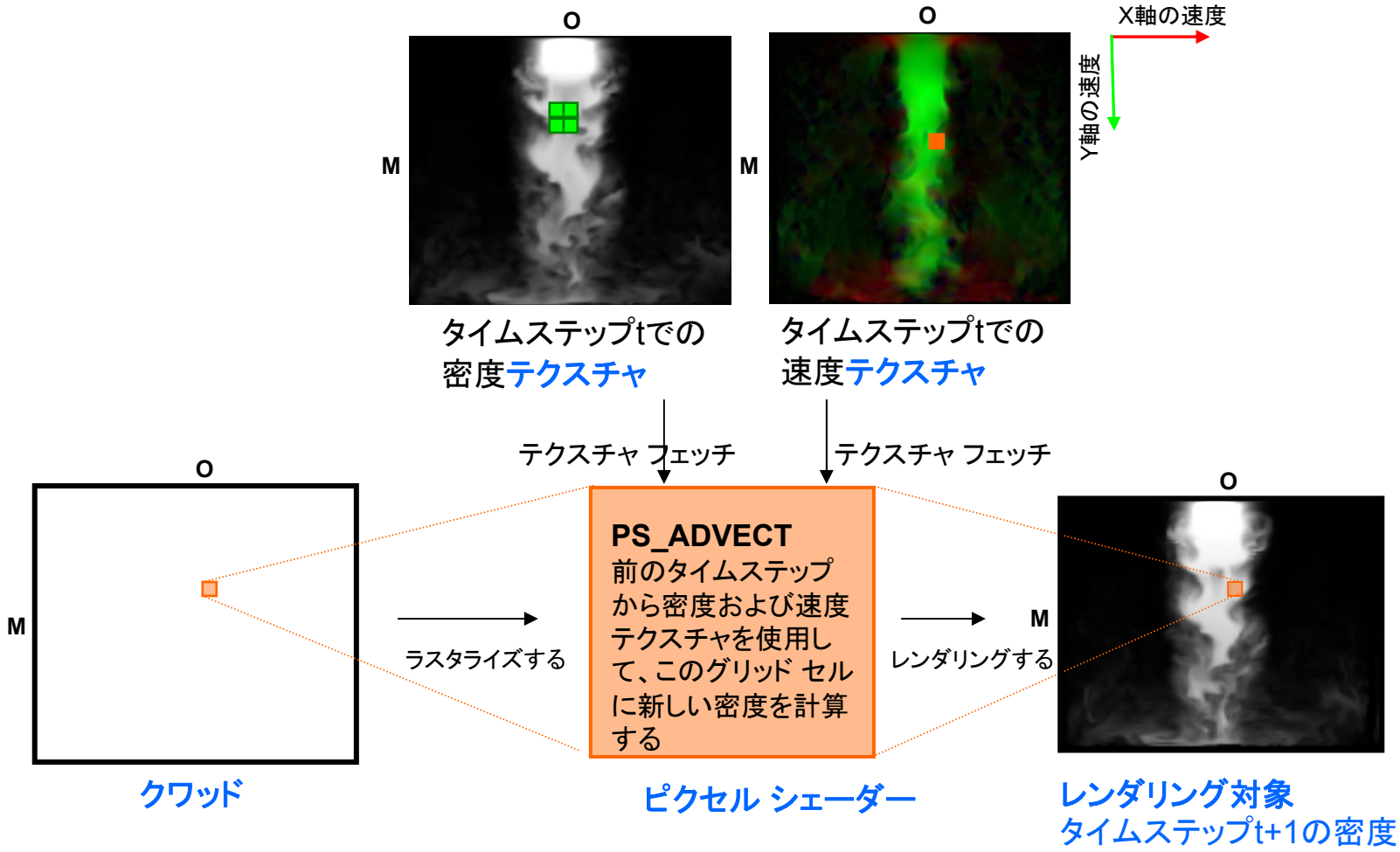
GPUでの流体シミュレーション

従来のGPGPUアプローチ:

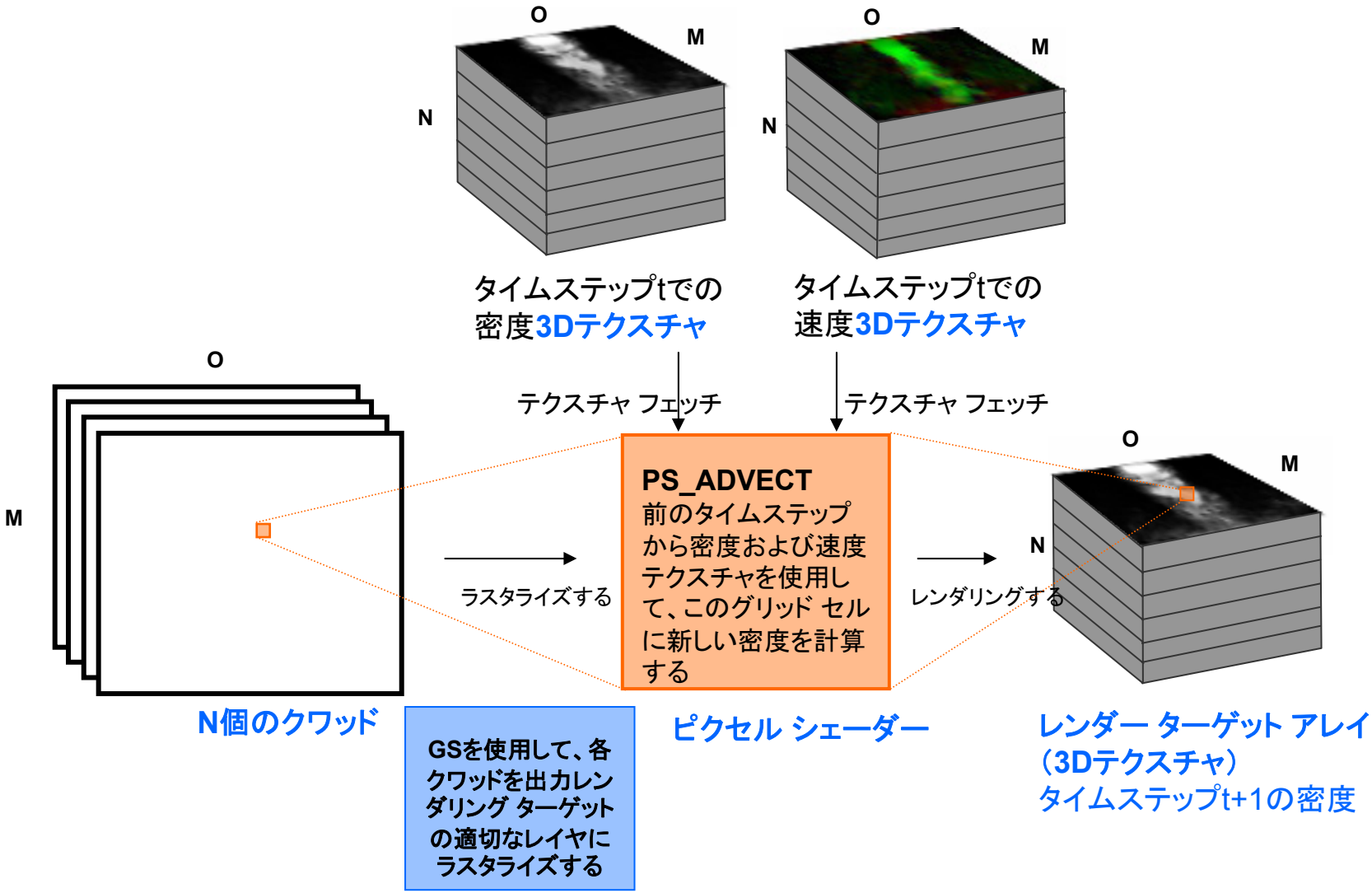
- 速度、密度、圧力→ テクスチャ
- グリッド全体のサブステップをシミュレートする
  → グリッド サイズで、画面調整したクワッドをレンダリングする
- グリッド セルの計算→ ピクセル シェーダー

出力値→ テクスチャのレンダリングを使用

GPUでの移流


O
M
O
M
X軸の速度
Y軸の速度
タイムステップtでの
密度テクスチャ
タイムステップtでの
速度テクスチャ
テクスチャ フェッチ
テクスチャ フェッチ
O
M
ラスタライズする
PS_ADVECT
前のタイムステップから密度および速度テクスチャを使用して、このグリッド セルに新しい密度を計算する
レンダリングする
O
M
クワッド
ピクセル シェーダー
レンダリング対象
タイムステップt+1の密度

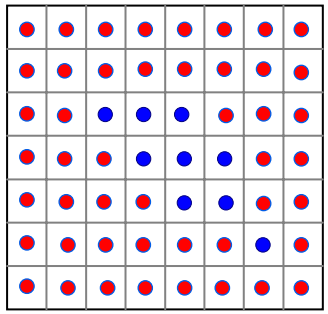
GPUを3Dで移流


O
M
N
タイムステップtでの
密度3Dテクスチャ
O
M
N
タイムステップtでの
速度3Dテクスチャ
テクスチャ フェッチ
テクスチャ フェッチ
O
M
N
N個のクワッド
ラスタライズする
PS_ADVECT
前のタイムステップ
から密度および速度
テクスチャを使用し
て、このグリッド セル
に新しい密度を計算
する
レンダリングする
O
M
N
GSを使用して、各
クワッドを出力レン
ダリング ターゲット
の適切なレイヤに
ラスタライズする
ピクセル シェーダー
レンダー ターゲット アレイ
（3Dテクスチャ）
タイムステップt+1の密度

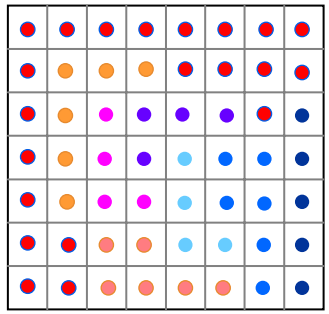
炎のシミュレーション

- 煙のシミュレーションと類似
- 以下の温度テクスチャの移流は除く
  - タイムステップごとに固定された量ずつ減らす
  - 目的のレンダリングに使用
  - 速度にも影響を与える可能性あり（浮力）

密度



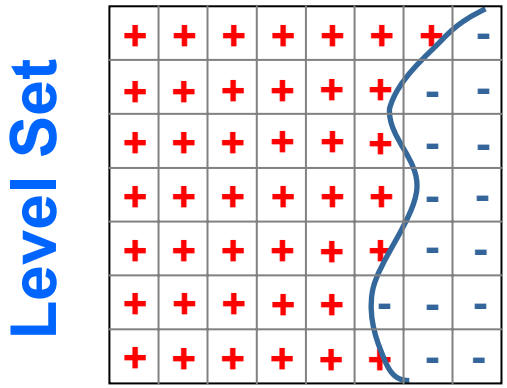
温度



- 追加する調節
  - 速度および密度の付加量
  - 付加する頻度

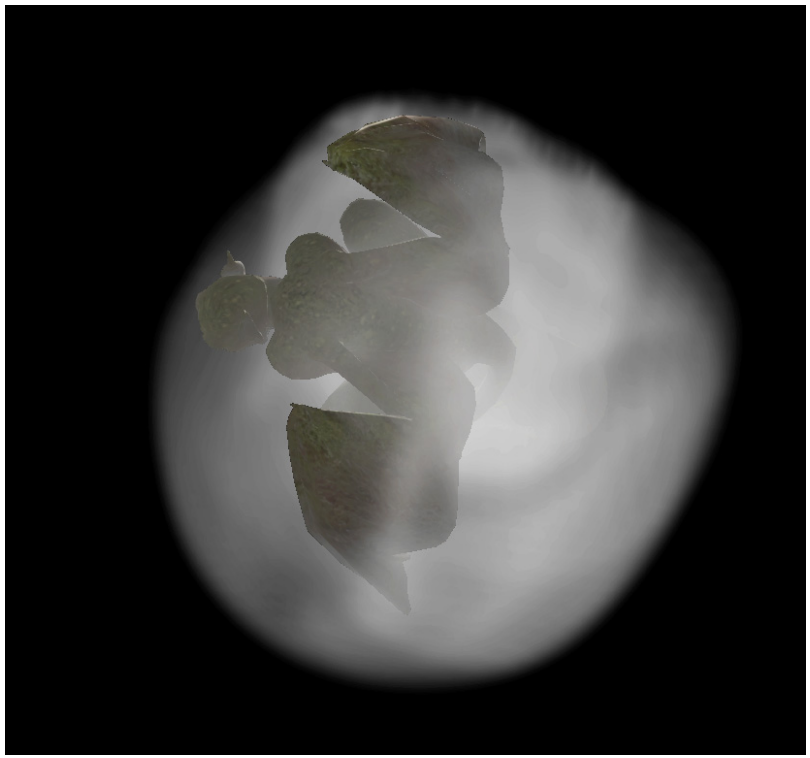
水のシミュレーション

- Level Setを使用して表示された水面
  - ボクセルごとに、符号付きの水面までの最短距離を保存する
    （外側ではプラス/内側ではマイナス）
- Level Setをスカラー フィールドとして移流する

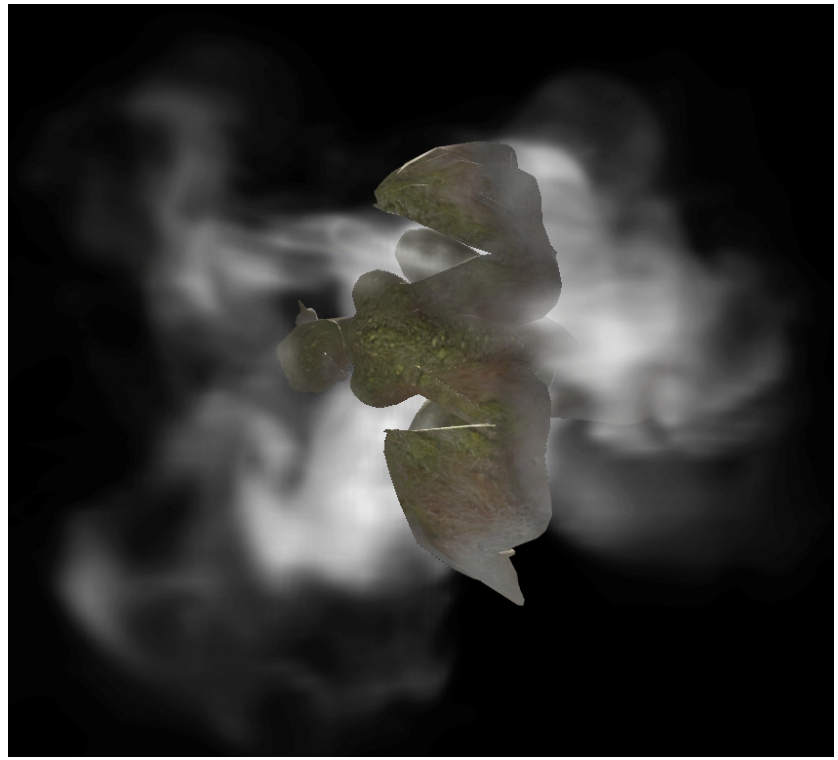


- Level Setは流体のダイナミクスに影響を与える
  - 液体の外側のセルの圧力をゼロに保つ
  - 重力などの外力を液体の外側にかけない
  - 液体の外側ではシミュレーションを省略する

動的障害物

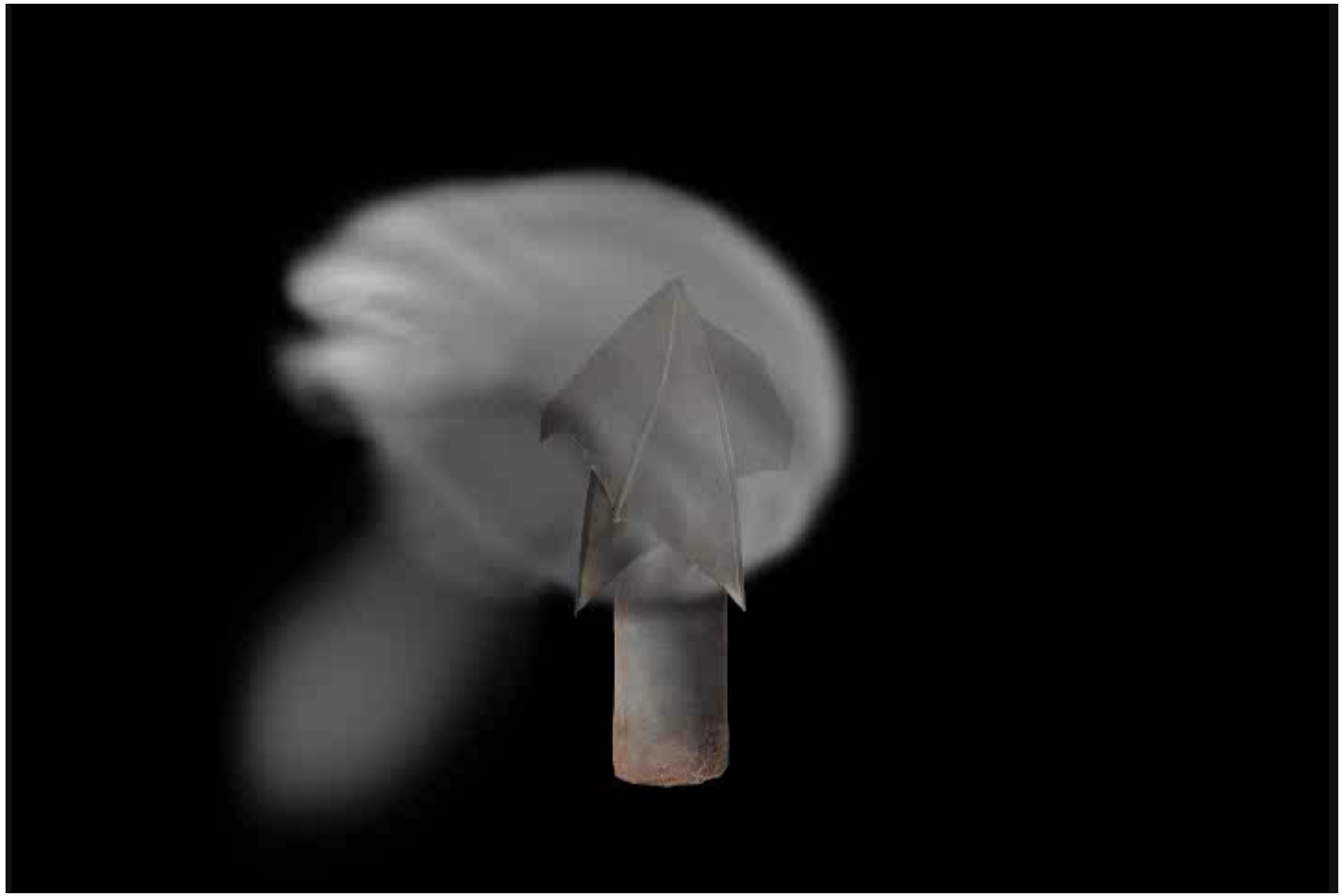


煙とシーンの合成のみを行う

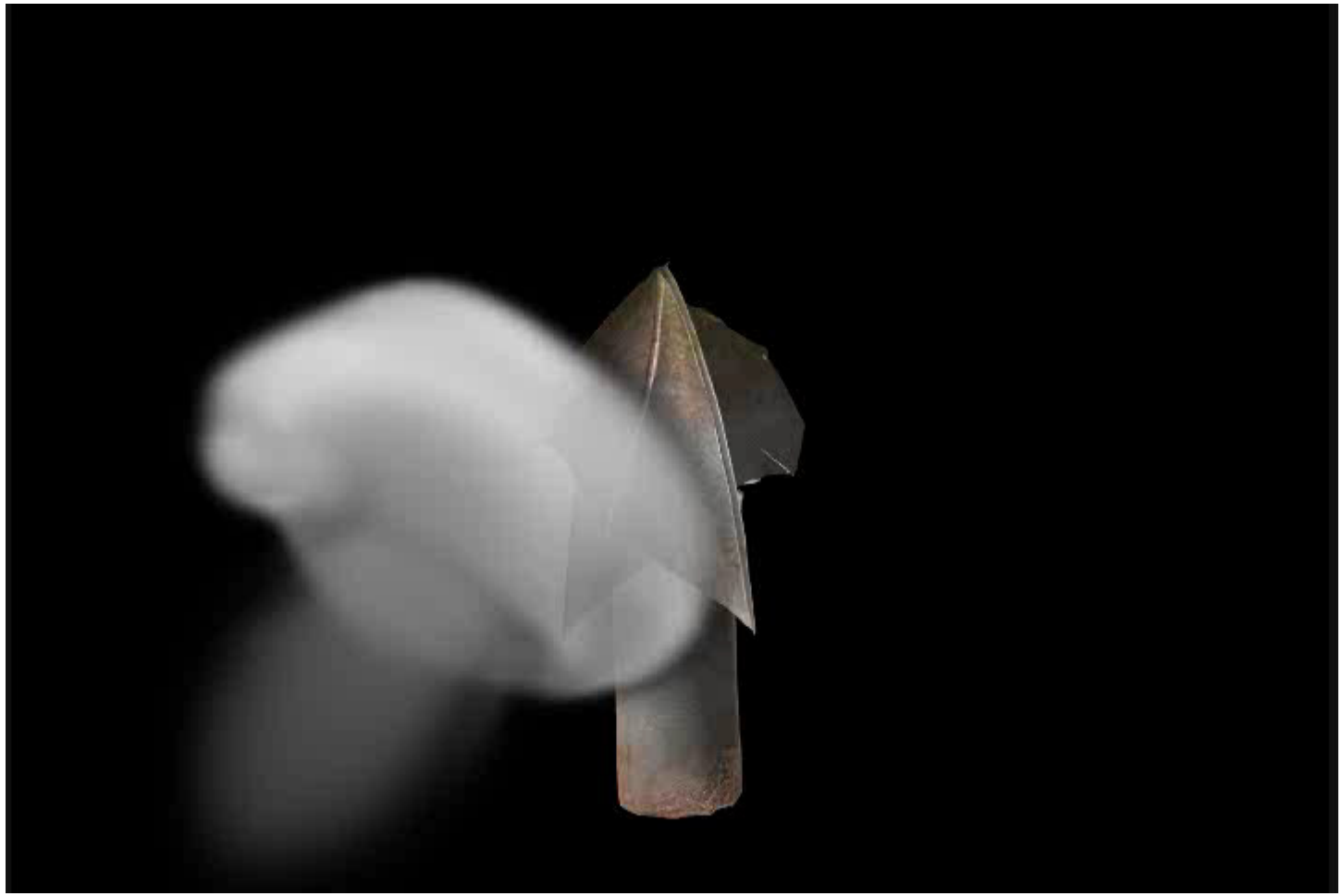


煙を障害物とインタラクトし、
シーンに合成する

障害物なし



障害物あり



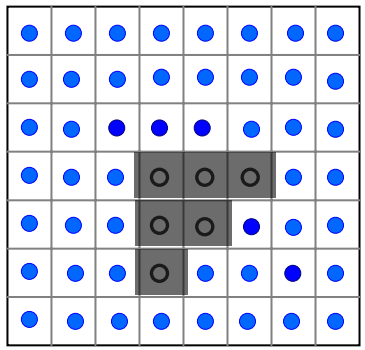
動的障害物の処理

- 流体が障害物に反応する方法を指定する
  境界条件

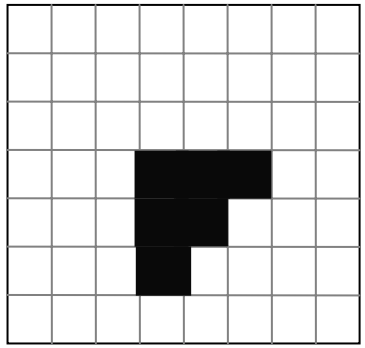
- 以下を満たすこと
  - 流体は障害物に囲まれたボリュームに入ってはならない
  - 移動する障害物は、流体に運動量を伝える

密度の境界条件

- 密度は、障害物の内部に付加または移流してはならない



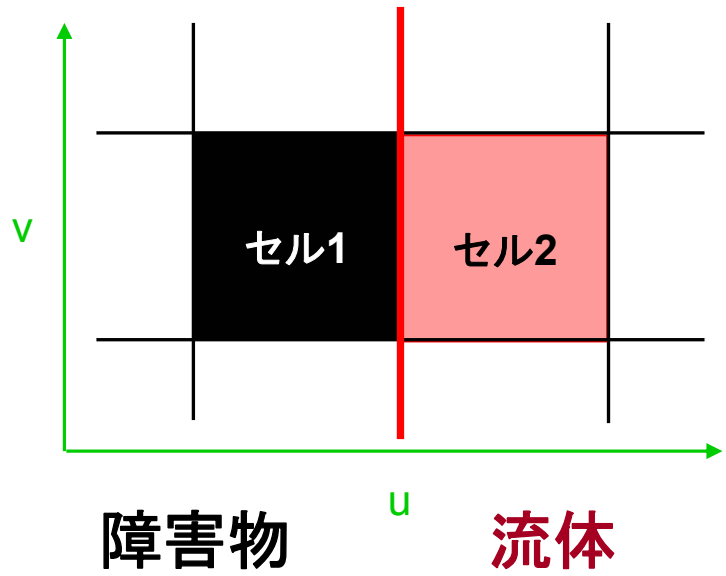
密度



障害物

圧力の境界条件

- 境界にまたがる圧力変化はゼロであること

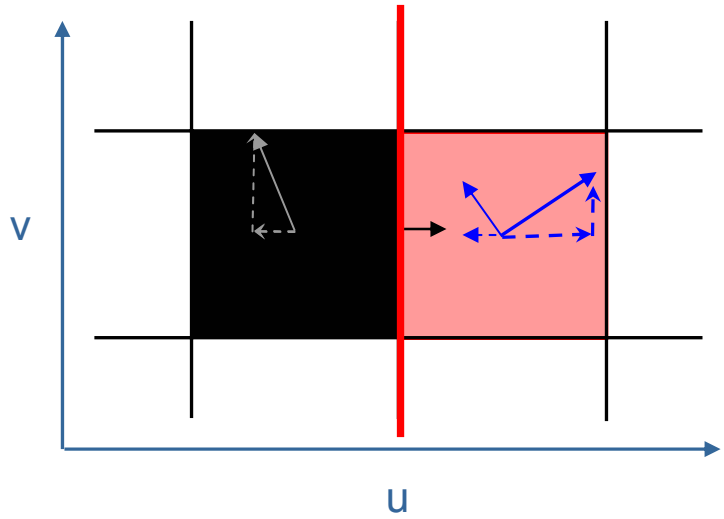


圧力セル1 – 圧力セル2 = 0

圧力セル1 =圧力I2

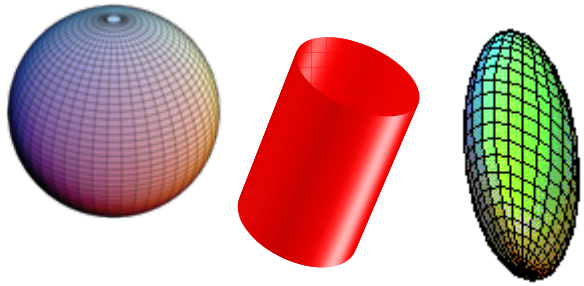
速度の境界条件

- 障害物の境界の法線の速度は、流体と障害物で同一である

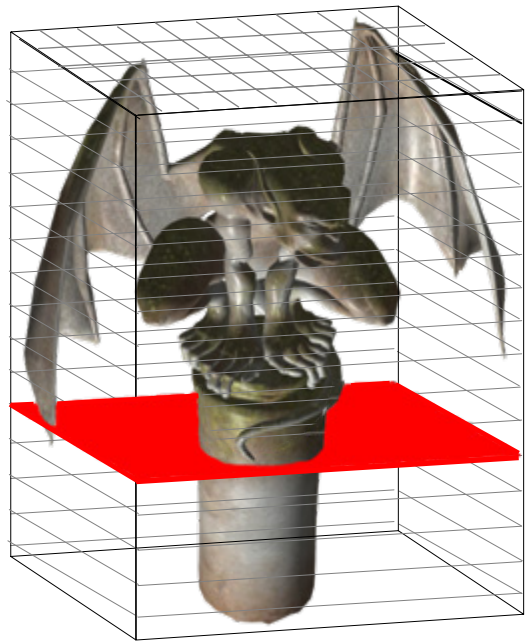


障害物の描写

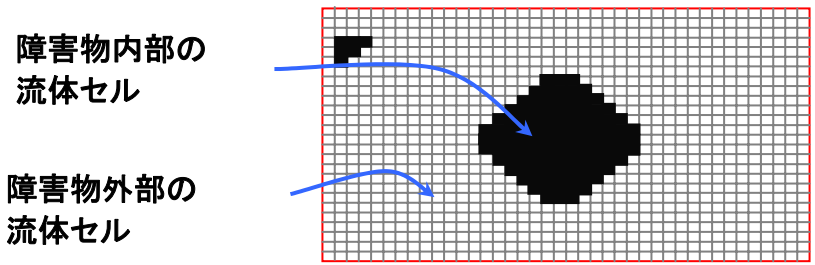
- 暗黙の形状
  - 球体、円柱、楕円形



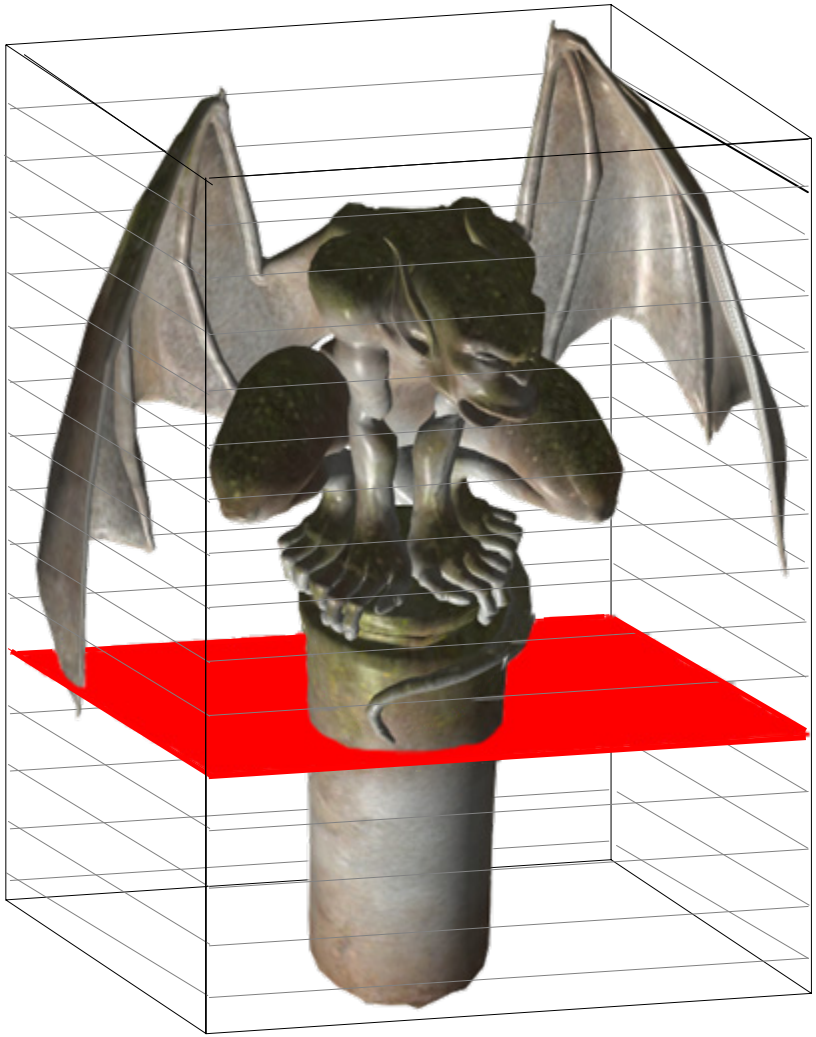
- ポリゴン モデル（三角形メッシュ）
  - 3Dテクスチャへボクセル化する
  - 静的:　　一度ボクセル化する
  - 動的:　　各フレームでボクセル化する



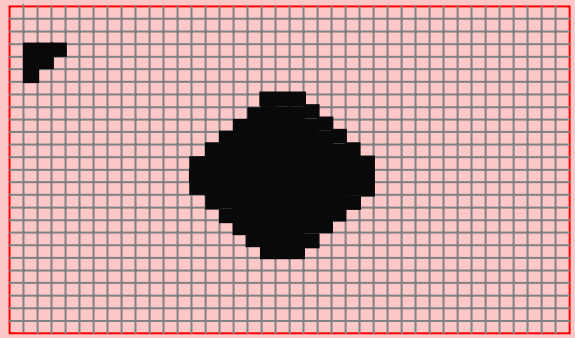
ボクセル化された障害物の3Dテクスチャ



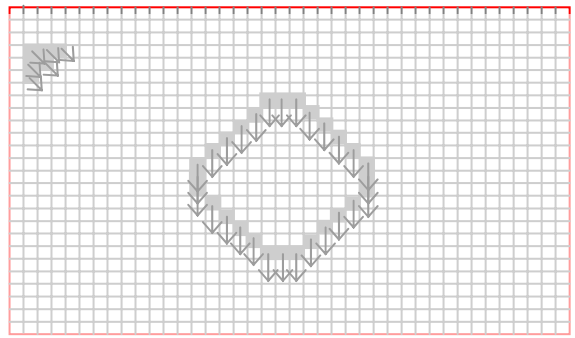
三角形メッシュのボクセル化



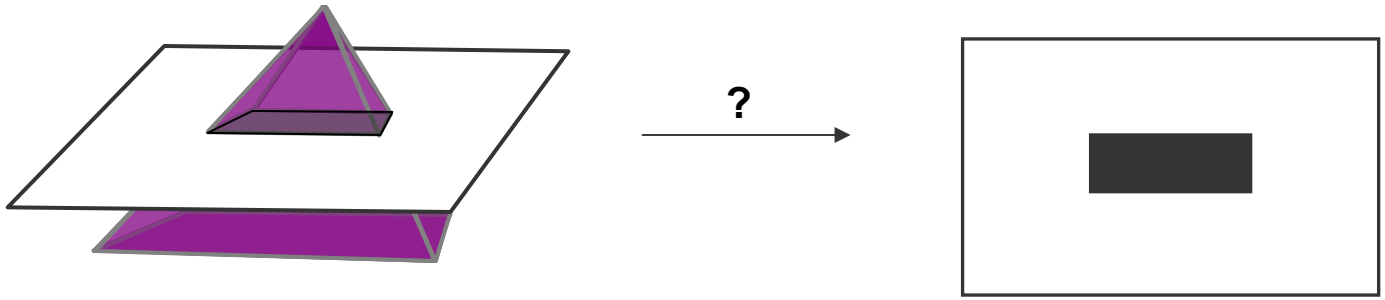
障害物のテクスチャ



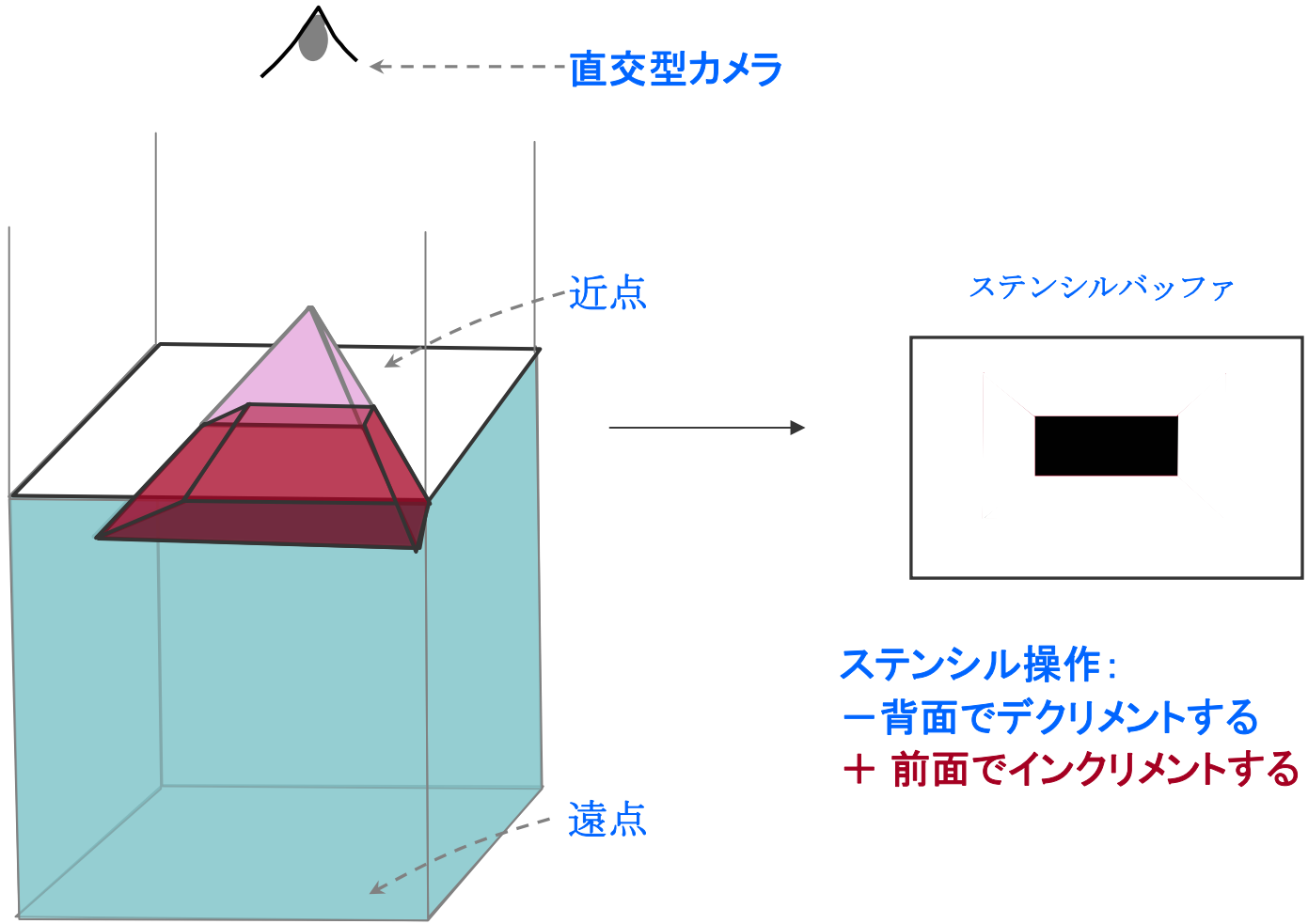
障害物の速度テクスチャ



ボクセル化の内部/外部－1/3

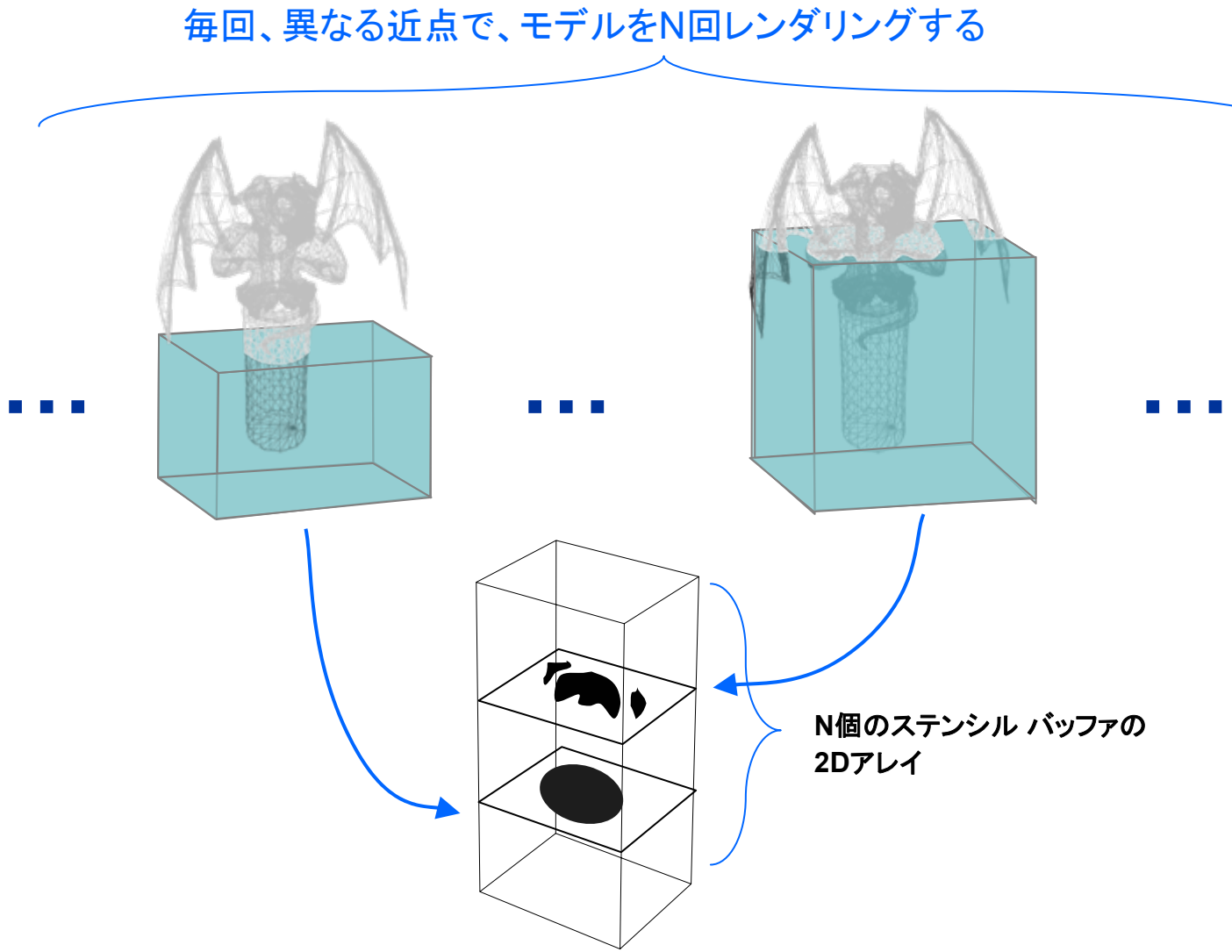


ボクセル化の内部/外部－2/3

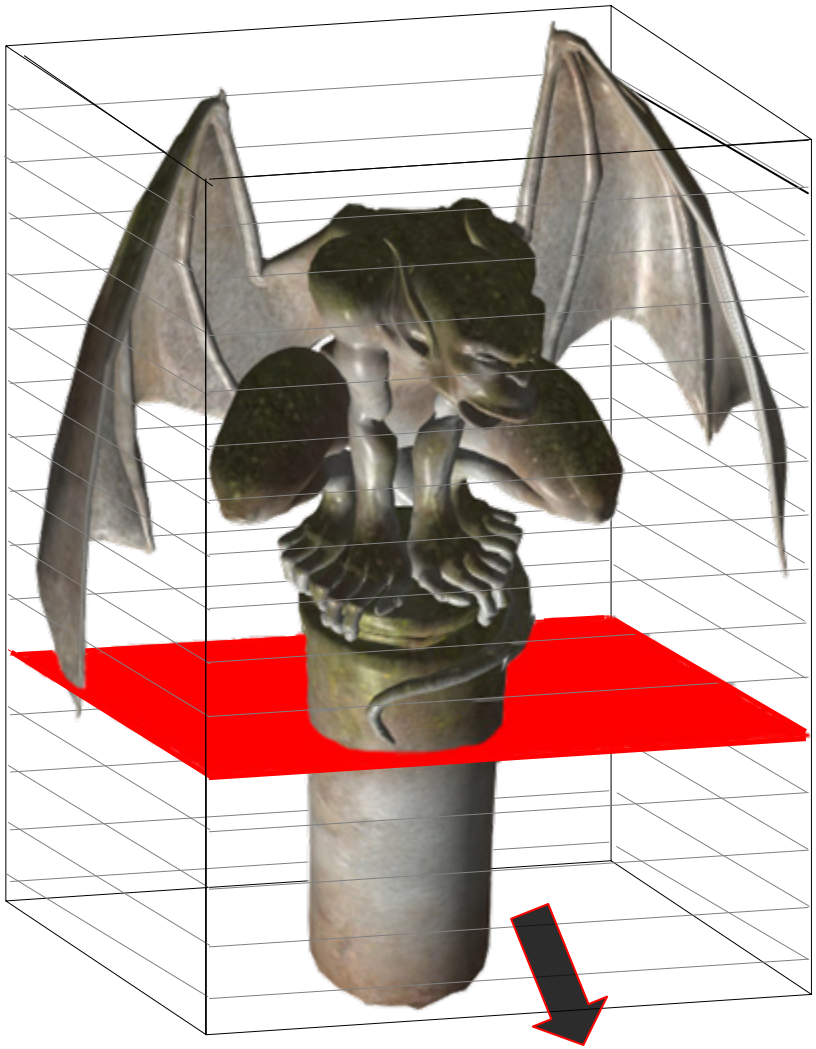


ステンシル操作:
－背面でデクリメントする
＋ 前面でインクリメントする

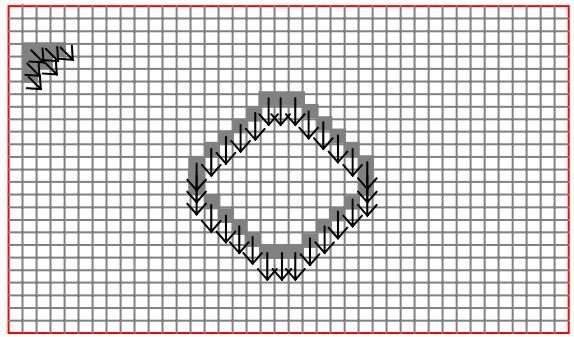
ボクセル化の内部/外部－3/3



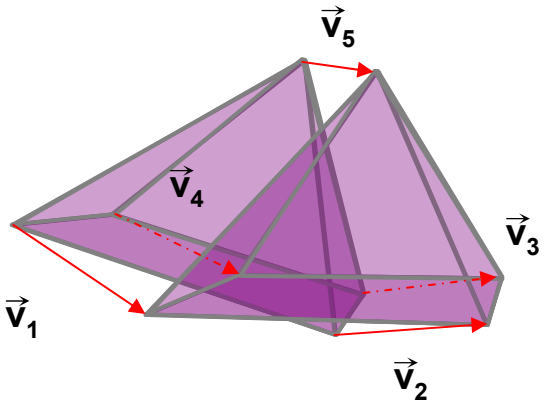
ボクセル化された速度



障害物の速度テクスチャ



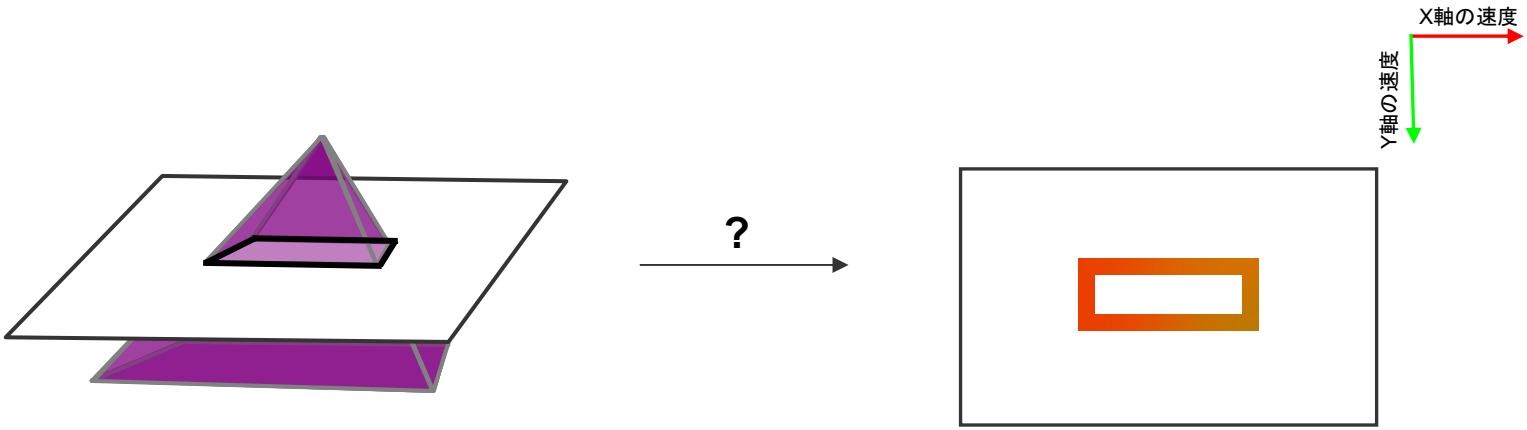
バーテックス単位の速度



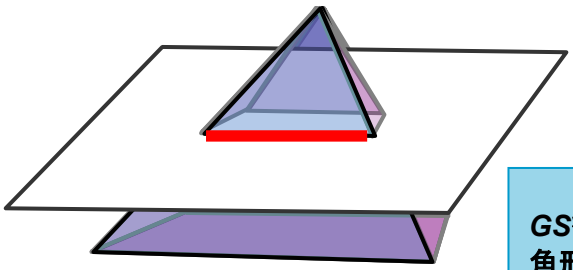
t^{n+1}

$$v = \frac{dx}{dt}$$

速度のボクセル化－1/3

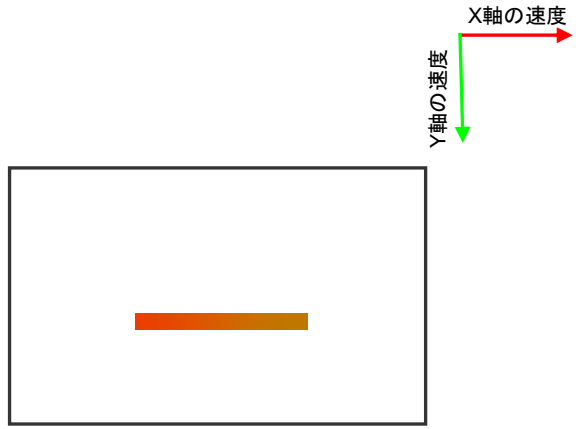


速度のボクセル化－2/3

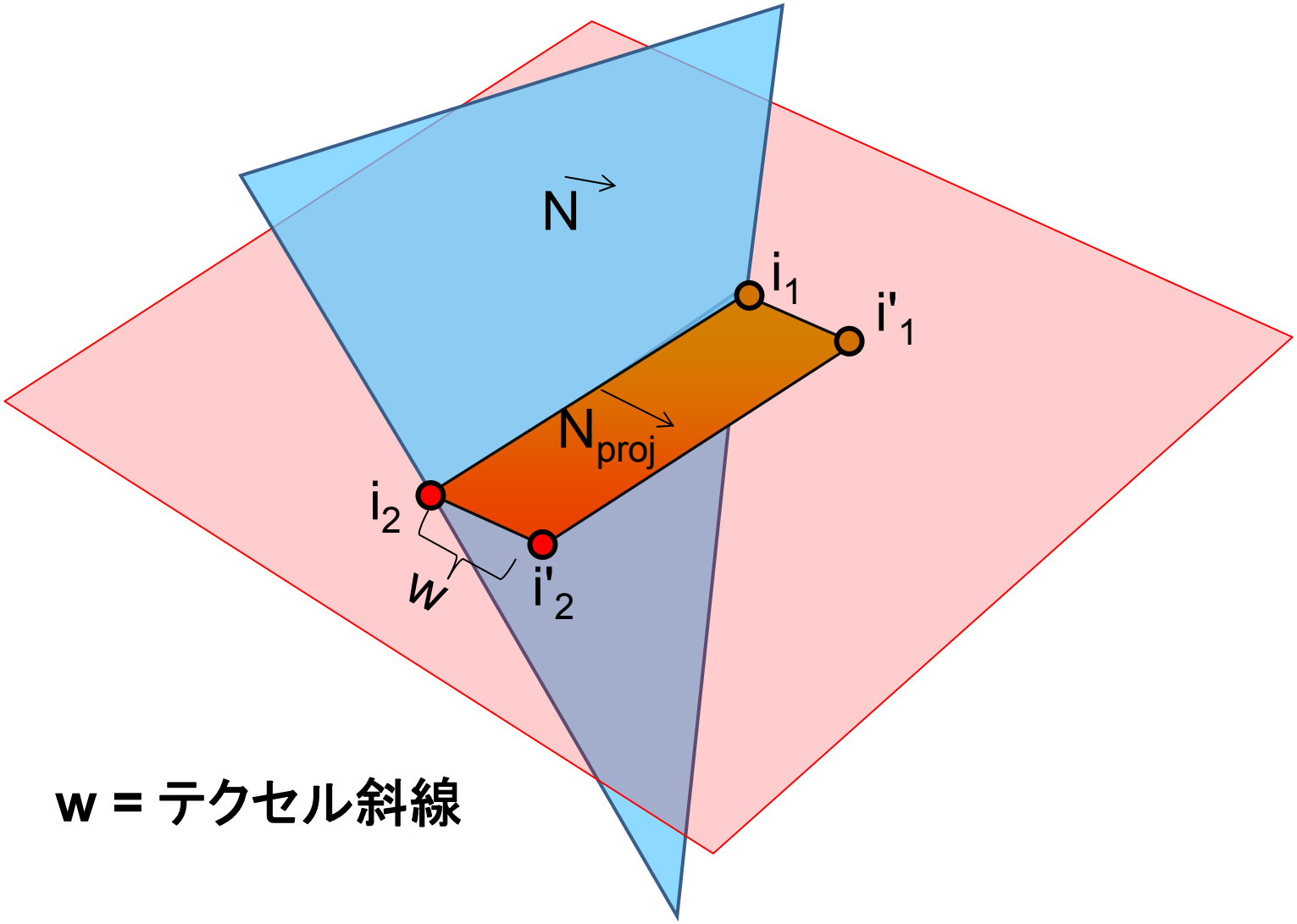


GSを使用して、スライスした三角形の断面を計算し、その部分に沿ったクワッドを出力する

クワッドのバーテックスで出力された「カラー」は、補間した速度値を示す



速度のボクセル化－3/3



w = テクセル斜線

最適化1: ストリーム出力

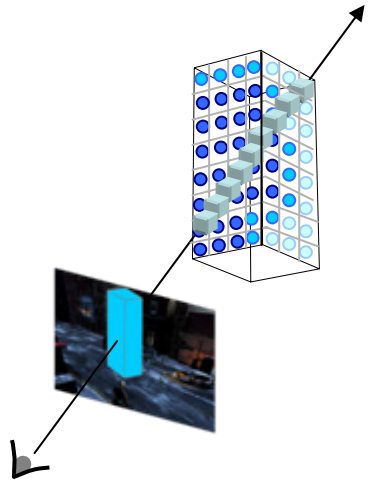
- フレームごとに1回、メッシュをはいでいく: 中間バーテックス バッファに*出力する*
- *n*回レンダリングする(おそらくインスタンシング使用)
- インスタンスごとに異なるスライス インスタンス:
  - GSを使用して、SV_RenderTargetArrayIndexを選択する
  - 異なるクリッピング面
  - ボクセル化の内外: 異なる直交射影行列
  - 速度のボクセル化: クリップした異なる面

最適化2: low. res.モデル

- 少ないジオメトリを高速でボクセル化する
- 内外のボクセル化アルゴリズムでは、水密性メッシュを必要とする
- ボクセル化グリッドの解像度は低いので(~100^3)、詳細な部分が失われる

レンダリング

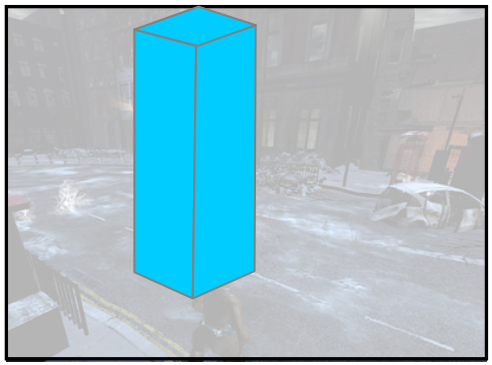
シーン



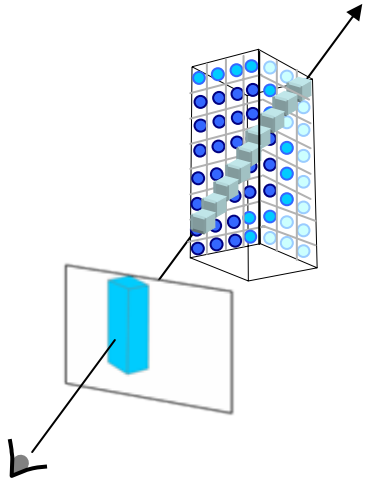
流体をレンダリングする

シーンに合成

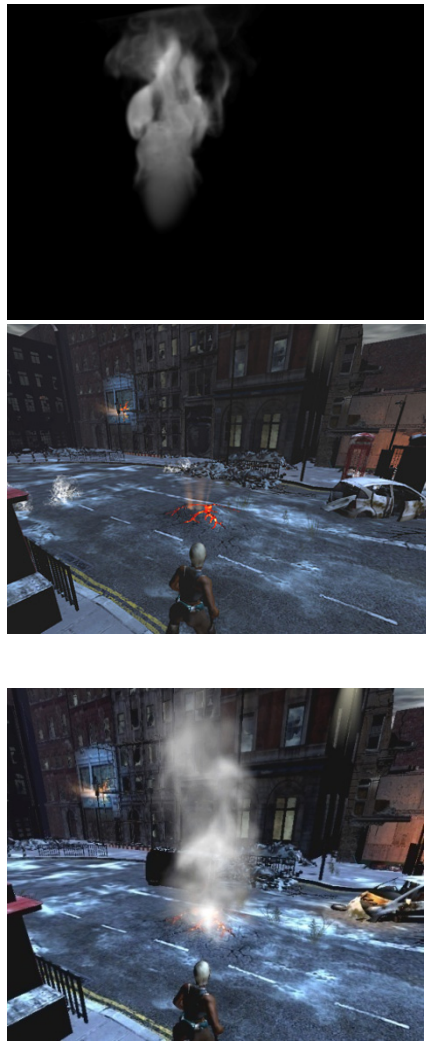
レンダリング



シミュレーション面を
レンダリングする



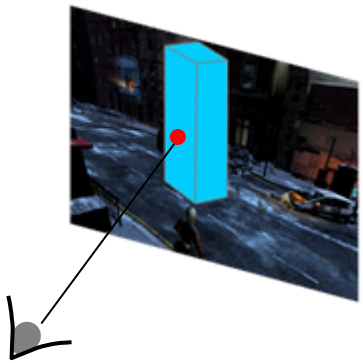
3Dテクスチャにレイキャストし、
値を組み合わせる



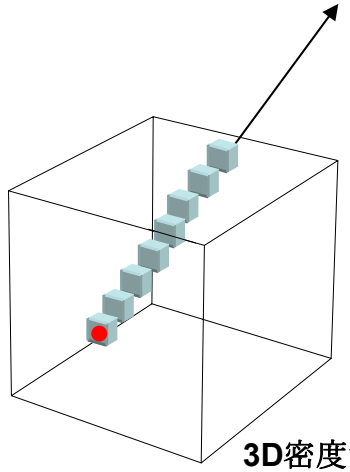
シーンのトップのアルファ
ブレンド

3Dテクスチャへのレイキャスティング

可視部分



不可視部分



3D密度テクスチャ

ー視点からボックスへのレイ
ーレイとボックスの交差部分

ー視点のレイがボックスを
通過する距離

実空間からテクスチャ スペースへの
変換

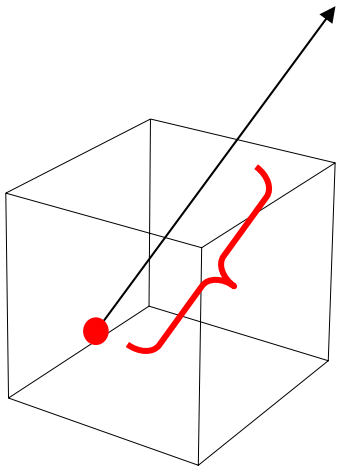
ーテクスチャ スペースのレイ
ーテクスチャのレイ エントリ ポイント

実空間からグリッド スペースへの
変換

ーレイが通過するボクセル数
= 取得するサンプル数

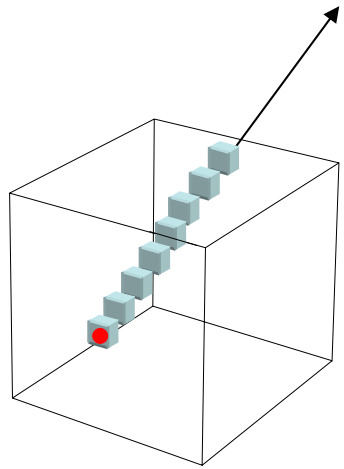
レイキャスティング : 2つのステップ プロセス

ステップ1



レイのエントリー ポイントと
レイがテクスチャを通過し、
補助テクスチャに到達する
距離を書き出す

ステップ2

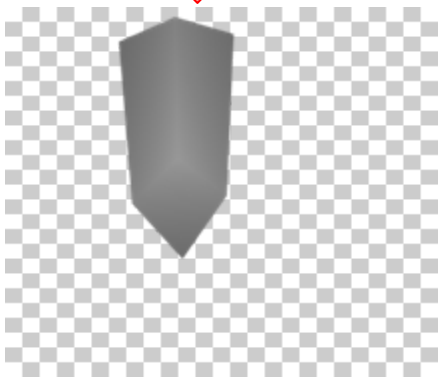


この情報を使用してシミュ
レーション テクスチャを展開
し、
出力テクスチャを計算する

レイキャスティングの設定

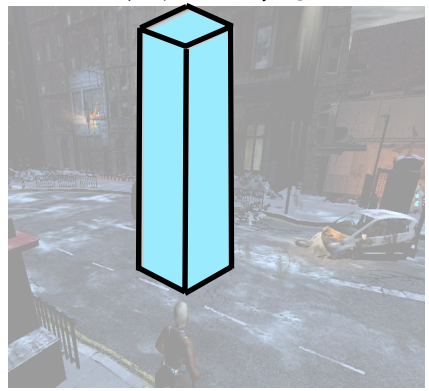
ボックスの背面を
RayDataTextureに
レンダリングする

float4(0,0,0, distanceToEye)

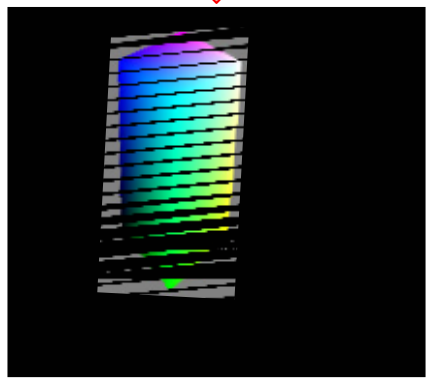


RayDataTexture.a

ボックスの前面を
RayDataTextureに
レンダリングする



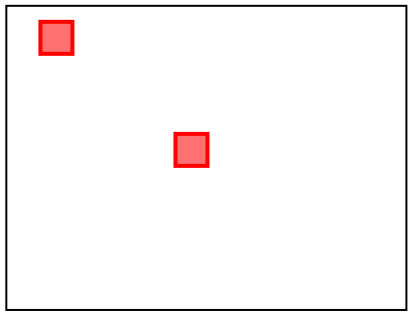
float4(-TextureSpacePos,
distanceToEye)
subtractive blending (DST -SRC)あり

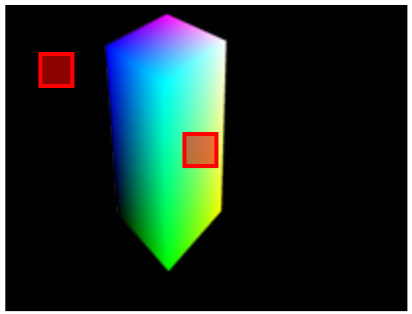


RayDataTexture.rgb

レイキャスティング（煙）

フルスクリーン クワッドを
フレーム バッファに
レンダリングする





RayDataTexture

NumberOfSamples =
TransformToGridSpace
(RayDataTexture.a)

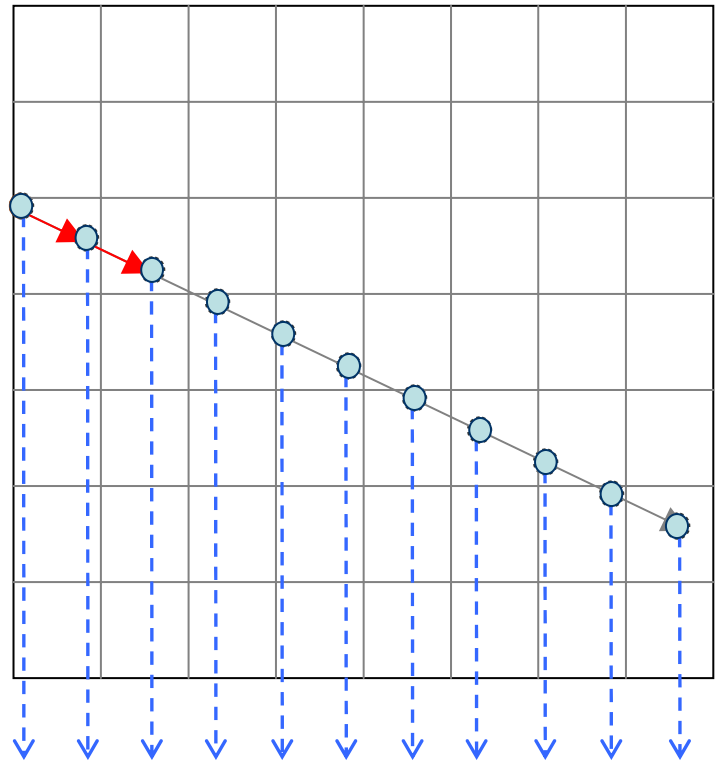
PositionInTexture =
TransformToTexSpace
(RayDataTexture.rgb)

MarchingVector =
TransformToTexSpace
(eye - RayDataTexture.rgb)

FinalColor.rgba = 0

○ ＝3Dテクスチャからの
トリリニア サンプル

密度テクスチャ



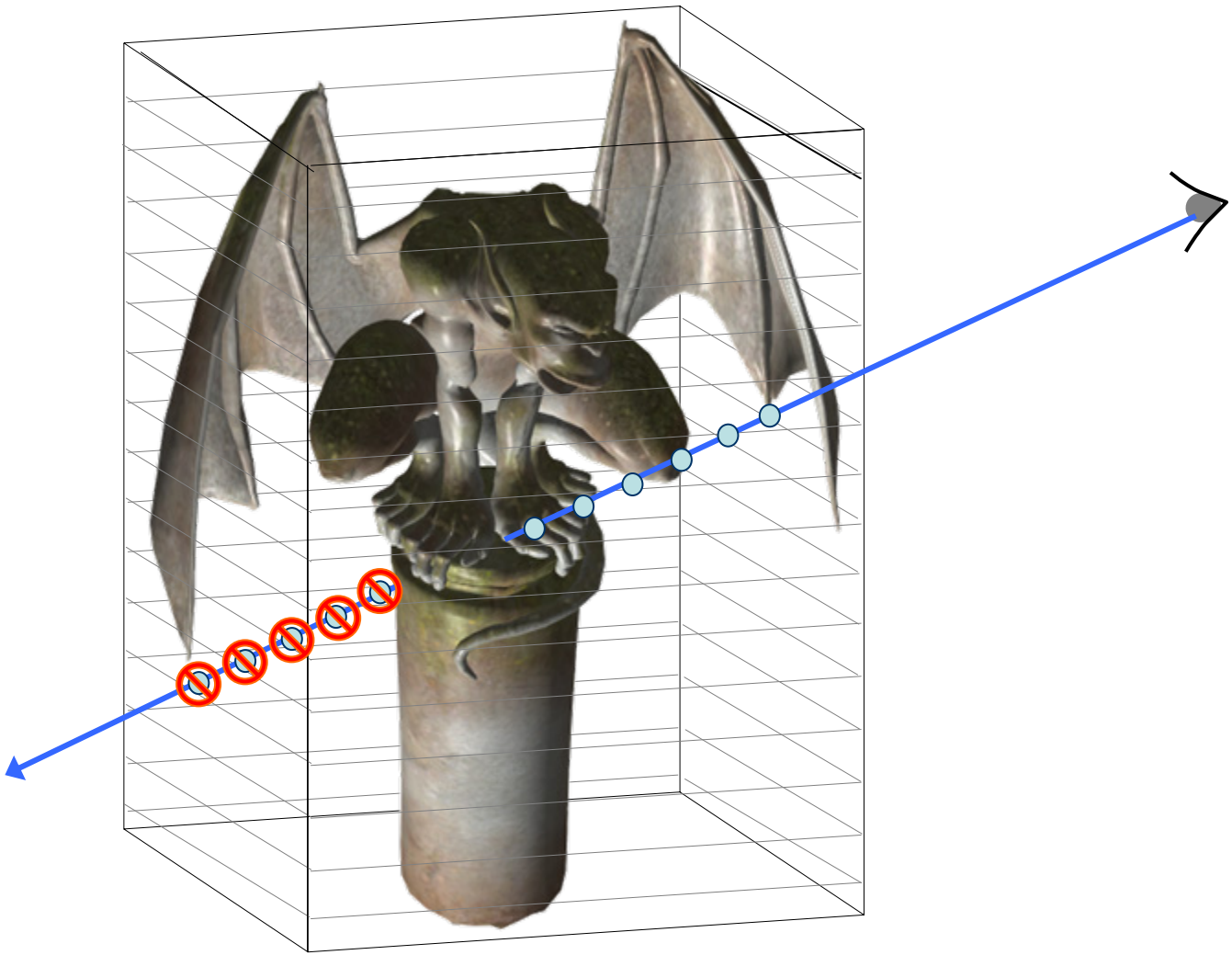
サンプル＝密度の一部の機能
FinalColor.rgb += sampleColor.rgb * SampleColor.a *
(1.0 – FinalColor.a)
FinalColor.a += SampleColor.a * (1.0 – FinalColor.a)

シーンの遮へい

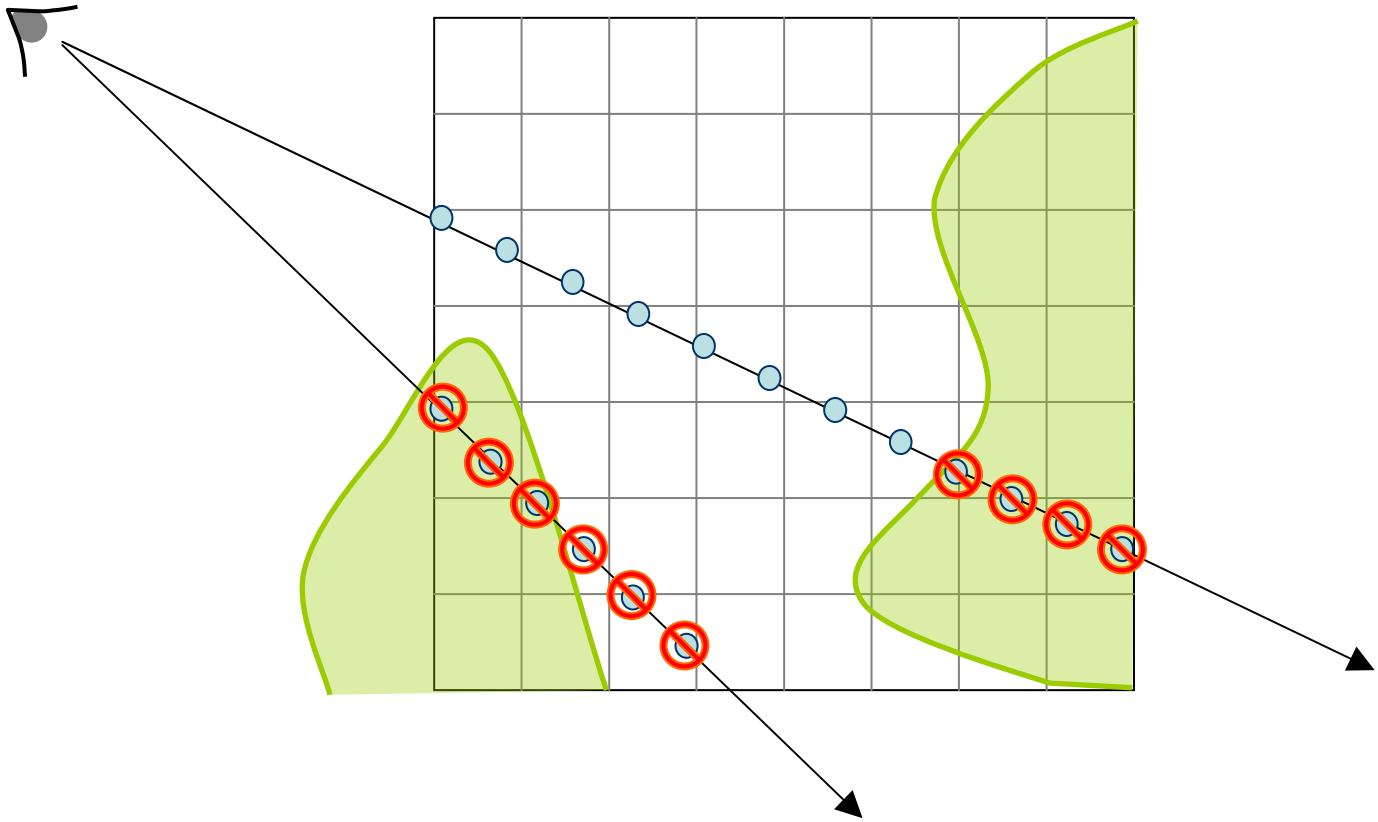
シーンのトップで
直接ブレンディングされた煙

シーンと正しく合成された煙

統合シーン デプス



統合シーン デプス

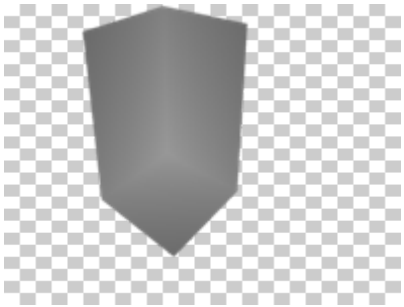


統合シーン デプス

使用前

シーン デプスの使用後

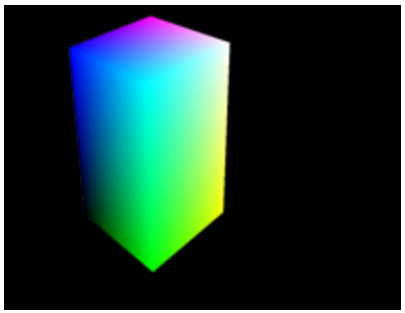
背面

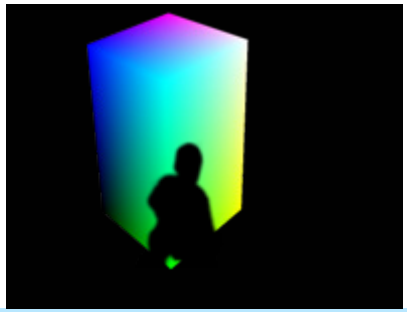


float4(0,0,0,distanceToEye)

float4(0,0,0,**min(sceneDistanceToEye, fragmentDistanceToEye)**)

前面





float4(-TextureSpacePos,distanceToEye)

(sceneDistanceToEye < fragmentDistanceToEye)
float4(0,0,0,0)の場合

その他の場合

float4(-TextureSpacePos,fragmentDistanceToEye)

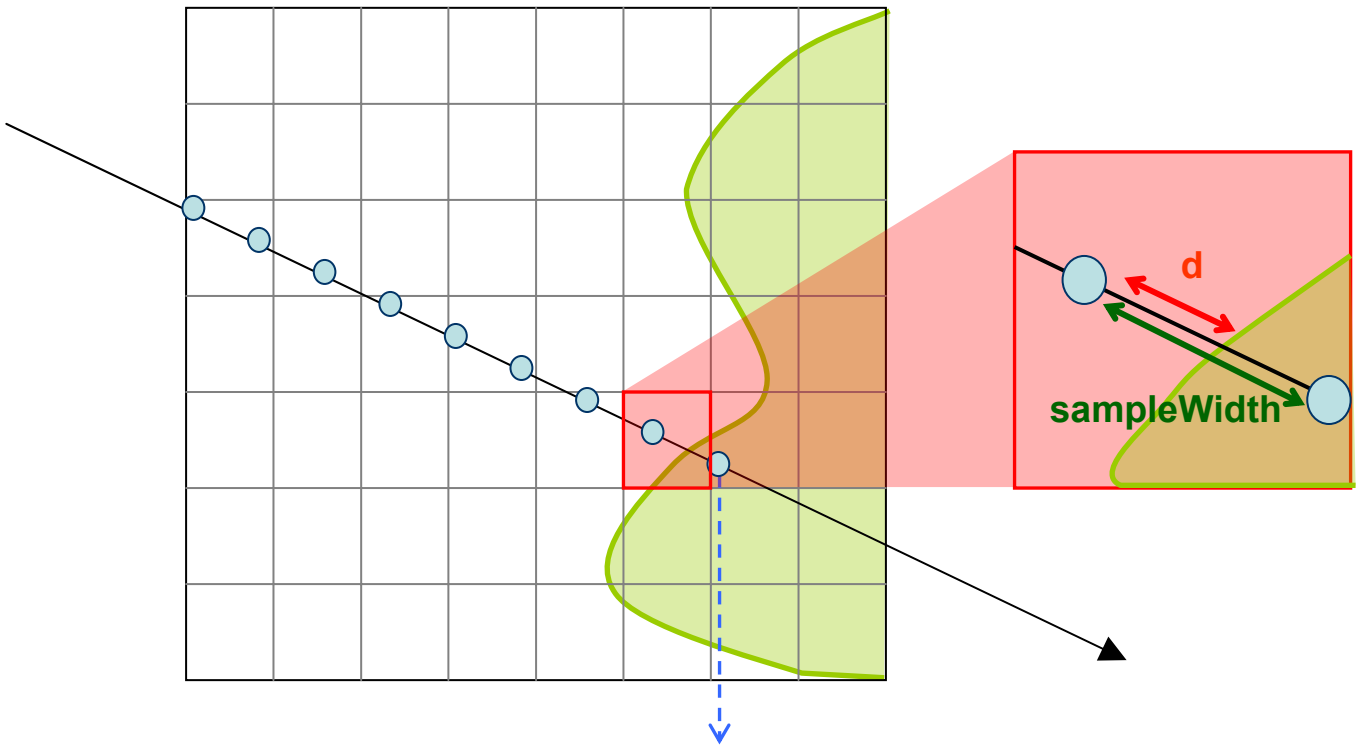
アーチファクト

整数化したサンプルによる
アーチファクト

重み付けしたサンプリングに
より、デプスを正しく使用

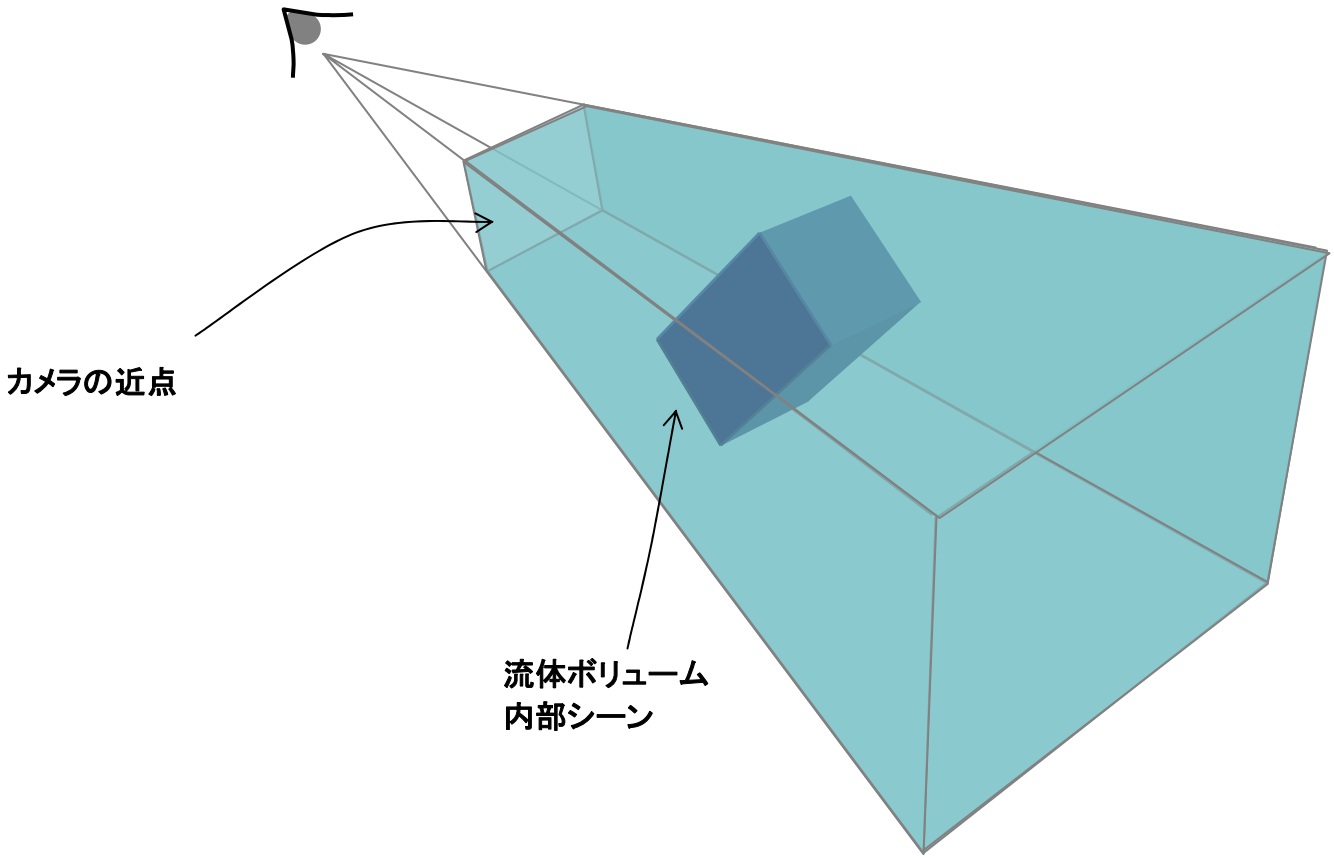
アーチファクト

正しい統合シーン デプス

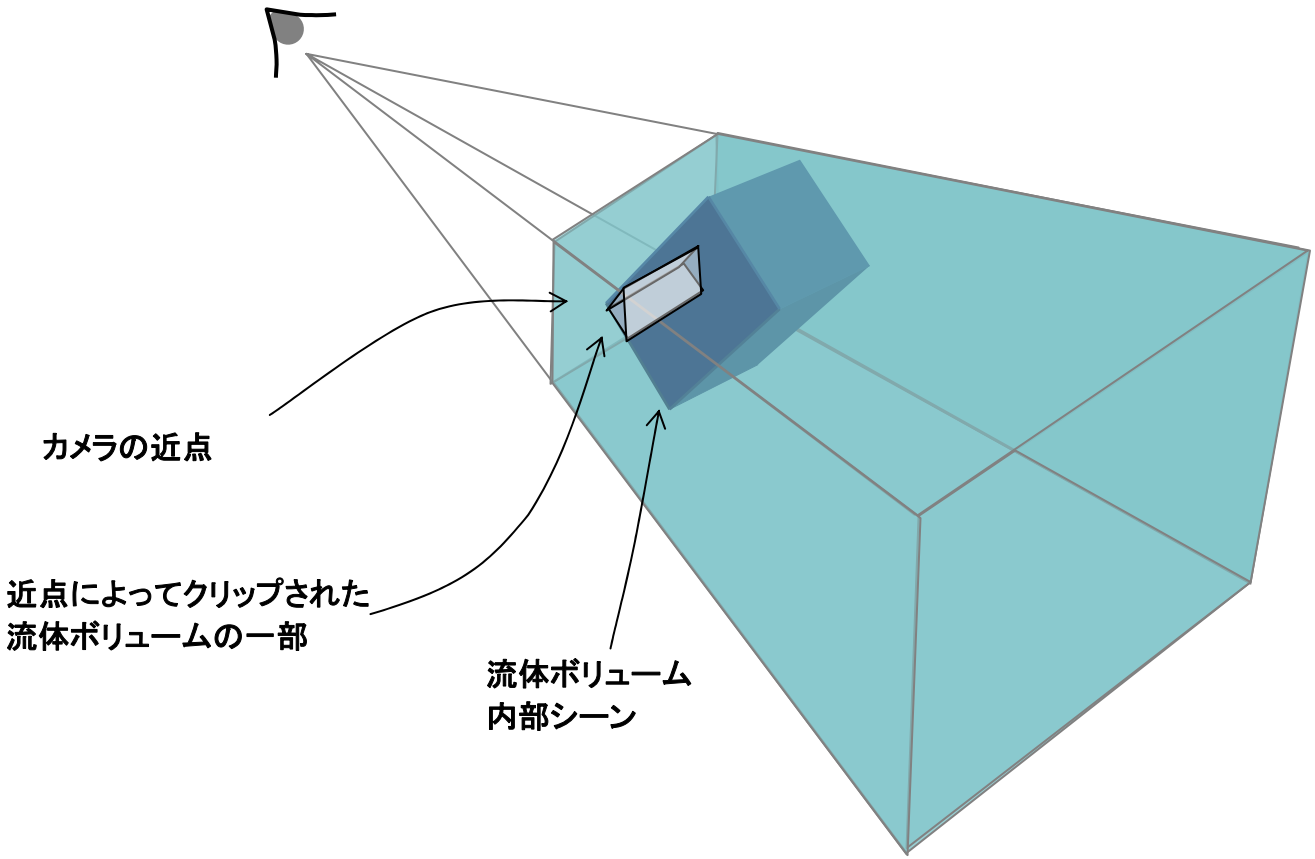


FinalColor.rgb += d/sampleWidth * SampleColor.rgb * SampleColor.a * (1.0 – FinalColor.a)
FinalColor.a += d/sampleWidth * SampleColor.a * (1.0 – FinalColor.a)

流体ボリューム内のカメラ

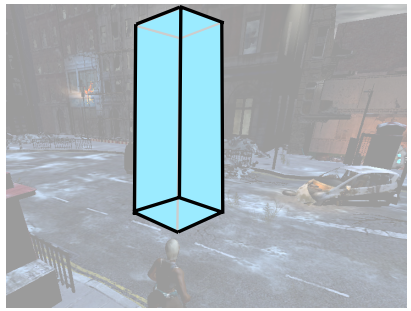


流体ボリューム内のカメラ

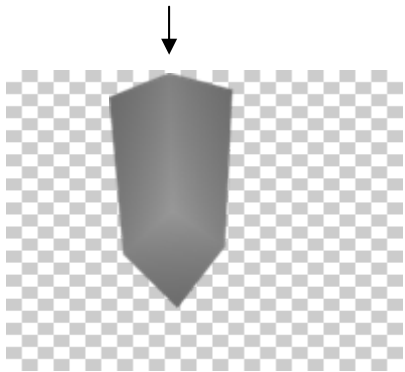


煙ボリューム内のカメラ

ボックスの背面を
RayDataTextureに
レンダリングする

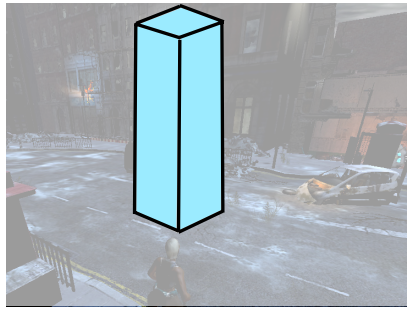


Float4(0,0,0,distanceToEye)

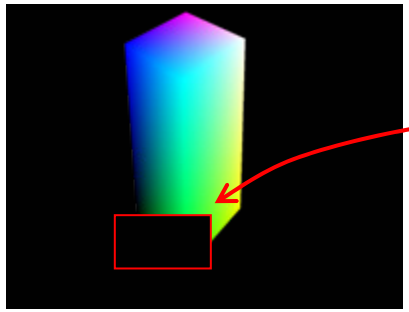


RayDataTexture.a

ボックスの前面を
RayDataTextureに
レンダリングする



Float4(-TextureSpacePos, distanceToEye)
subtractive blending (DST -SRC)あり



RayDataTexture.rgb

近点によってクリップされた前面：
ピクセルでのデプスは正しくない
テクスチャのピクセル位置に関する
情報はなし

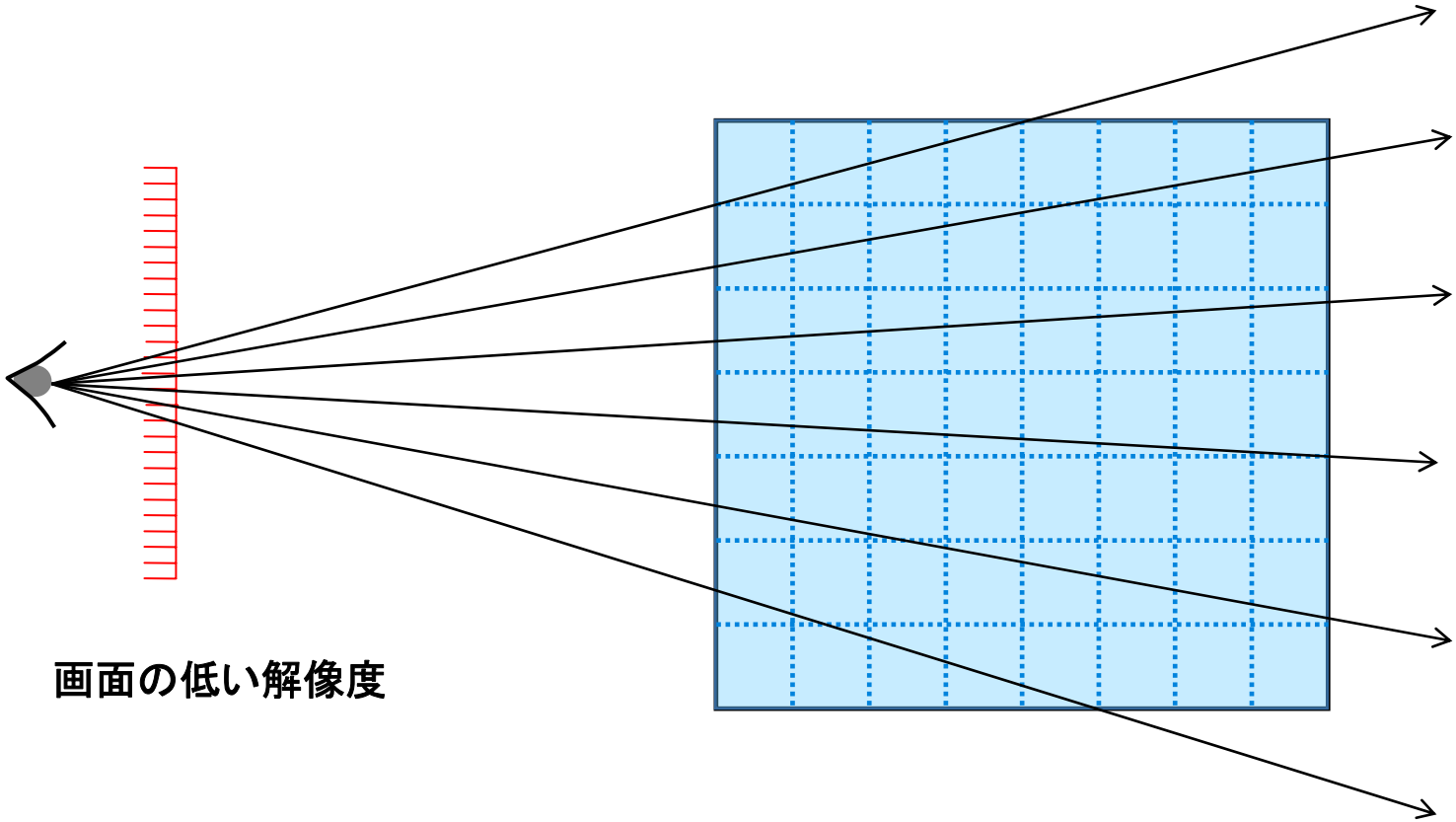
煙ボリューム内のカメラ

- 背面がレンダリングされ、前面がレンダリングされていないピクセルをマークする
- レイキャスティング ステップでは、上記のマークされたピクセルの場合
  - 位置をテクスチャに明示的に設定する
  - 符号化されたレイ デプスから、視点から近点の適切なポイントまでの距離を引く

精度問題の回避

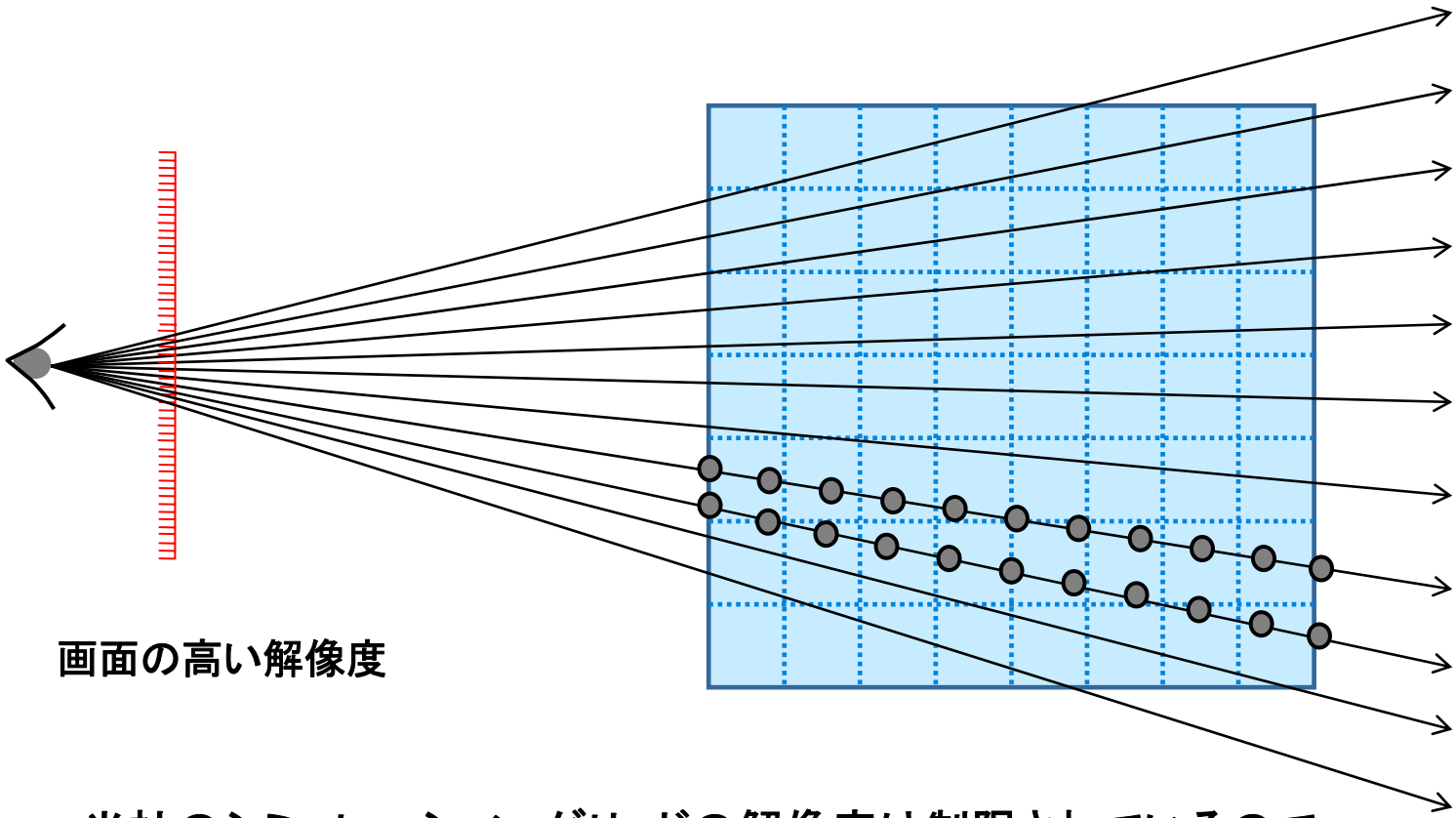
- 32ビット以上の精度浮動小数点を使用して、RayData テクスチャを保存する
- 移動時のギクシャクした動きを回避するのに十分遠く、シーンの大半の省略を回避するのに十分に近い近点の距離を設定する

独立したレイキャスティングの画面サイズ



画面の低い解像度

独立したレイキャスティングの画面サイズ



画面の高い解像度

当社のシミュレーション グリッドの解像度は制限されているので、
高解像度でレイキャスティングする意味はない

独立したレイキャスティングの画面サイズ

- 当社はレイキャスティングの解像度を制限しているので、投影されたボクセルのサイズはピクセルのサイズとほぼ同じである
- これより高い画面の解像度は、多くの近接するピクセルの同じボクセルによってレイキャスティングすることを意味する

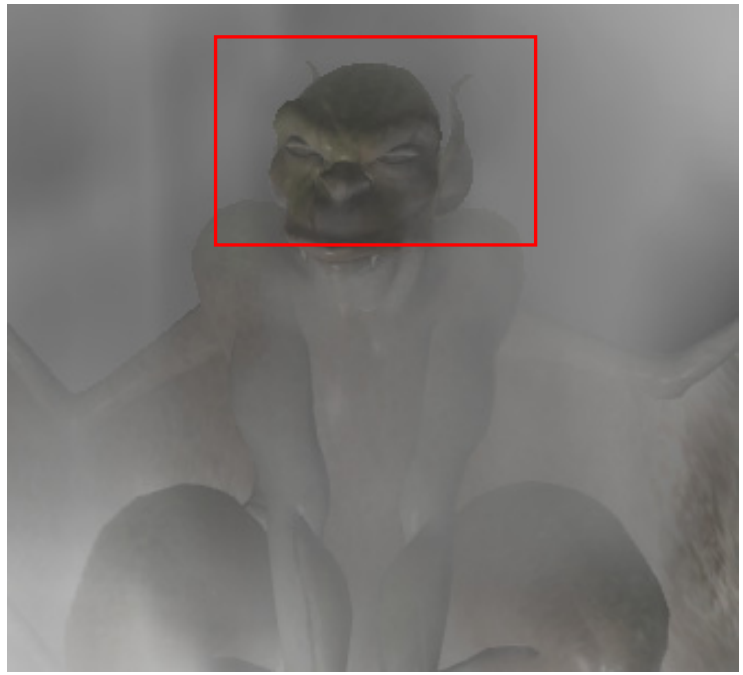
異なる解像度でのレイキャスティングの比較

高解像度レイキャスティングの解像度:
1264×958
フレーム レート: 31 fps

低解像度レイキャスティングの解像度:
519×393
フレーム レート: 90 fps

低解像度レイキャスティングの問題

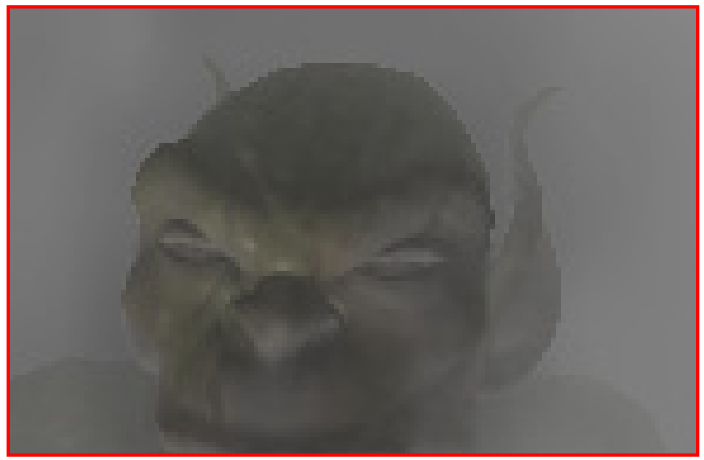
低解像度レイキャスティング



高解像度レイキャスティング

低解像度レイキャスティングの問題

低解像度レイキャスティング



高解像度レイキャスティング

問題

- 低解像度で煙をレイキャスティングすることは良いことである。理由として以下が挙げられる。
  - 煙に関する高解像度の情報がなかった
  - 煙自体が曖昧である
- シーン デプスには高解像度情報があるが、エッジは含まれない

本当のシーン デプス

レイキャスティングが使用する効果的なシーン デプス

様々な解像度を用いたレイキャスティング

低解像度で
レイキャスティングする

RayDataTextureで
エッジを検出する

高解像度でエッジを
レイキャスティングする

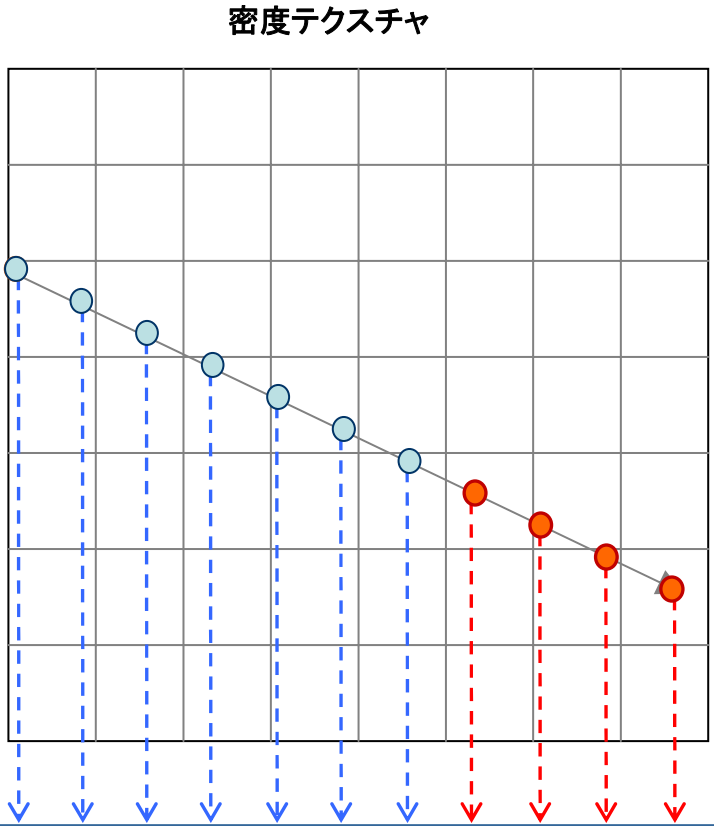
炎のレンダリング

- 炎はブレンディングを付加してレンダリングする
- 温度値を使用して、1D伝達関数を調べる

高温　　低温

- 炎の温度が特定のスレッシュホールドを下回った場合、煙としてレンダリングする

炎と煙を一緒にレンダリング

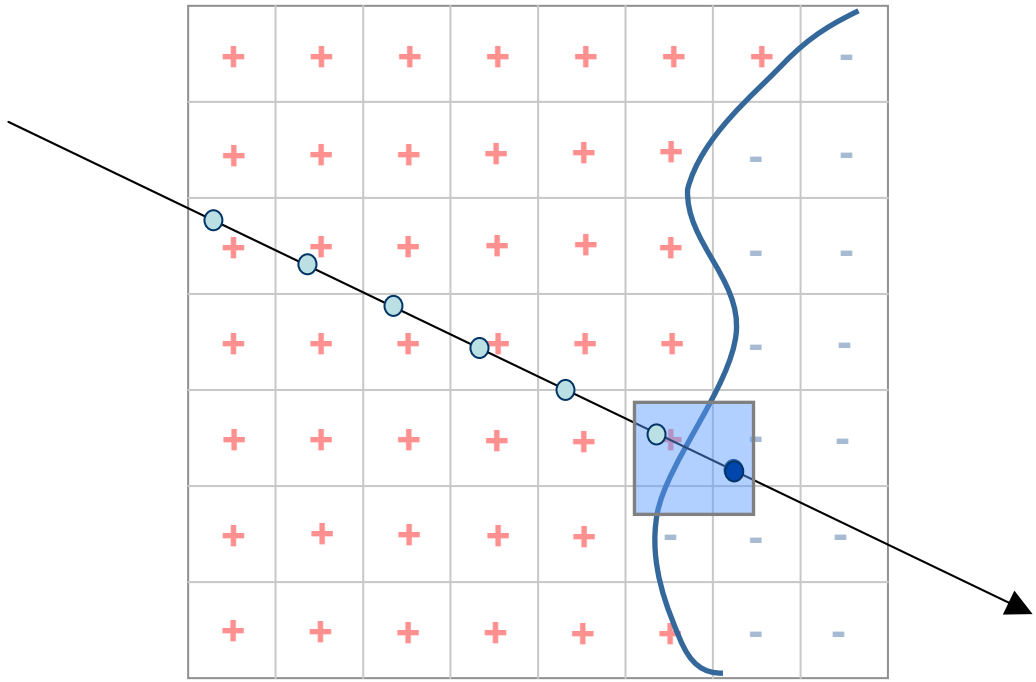


温度≤スレッシュホールドの場合
サンプル＝密度の一部の機能
FinalColor.rgb = (1 - sample.a) * FinalColor.rgb + sample.a * sample.rgb;
FinalColor.a = (1 - sample.a) * FinalColor.a + sample.a;

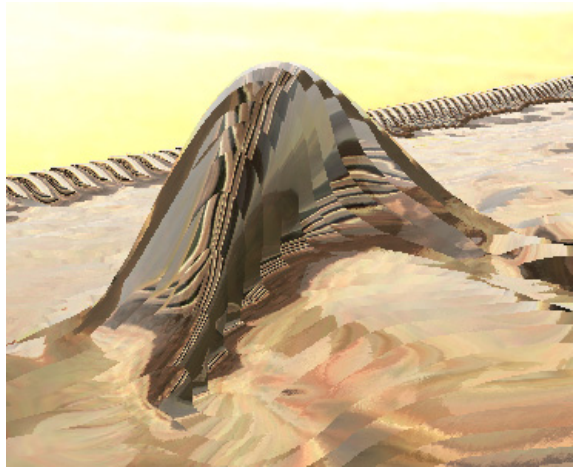
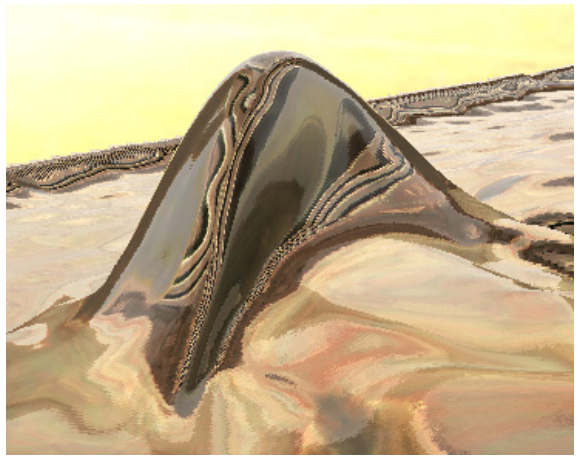
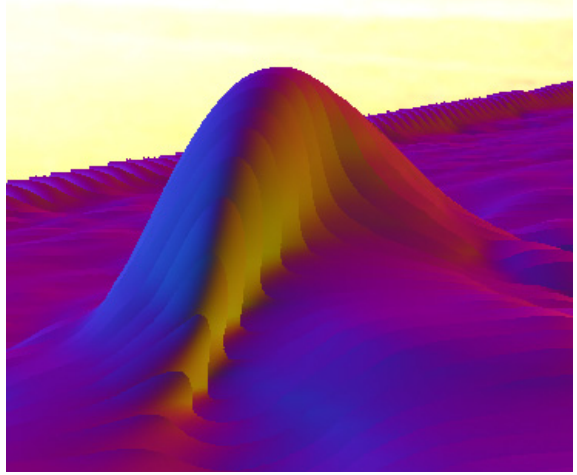
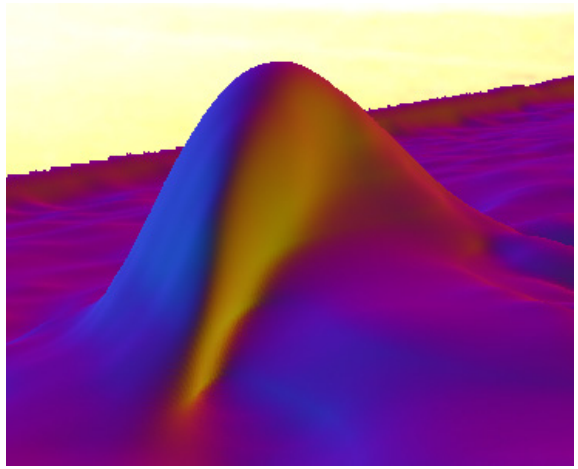
水のレンダリング

- 以前と同様にレイキャスティングする
- 値を累積するのではなく、Level Setの変化が現れた最初の場所を探す
- Level Setのグラジエントを取得することで、この点での法線を推定する
- 屈折と反射でフラグメントをレンダリングする

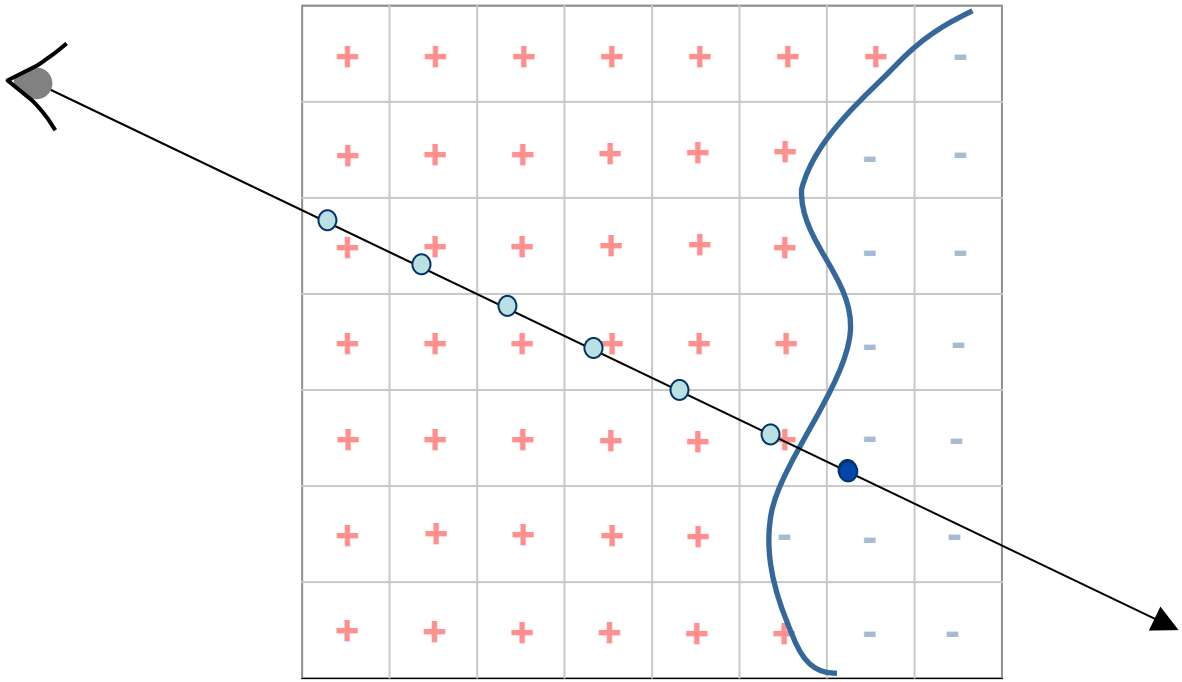
水面の求め方



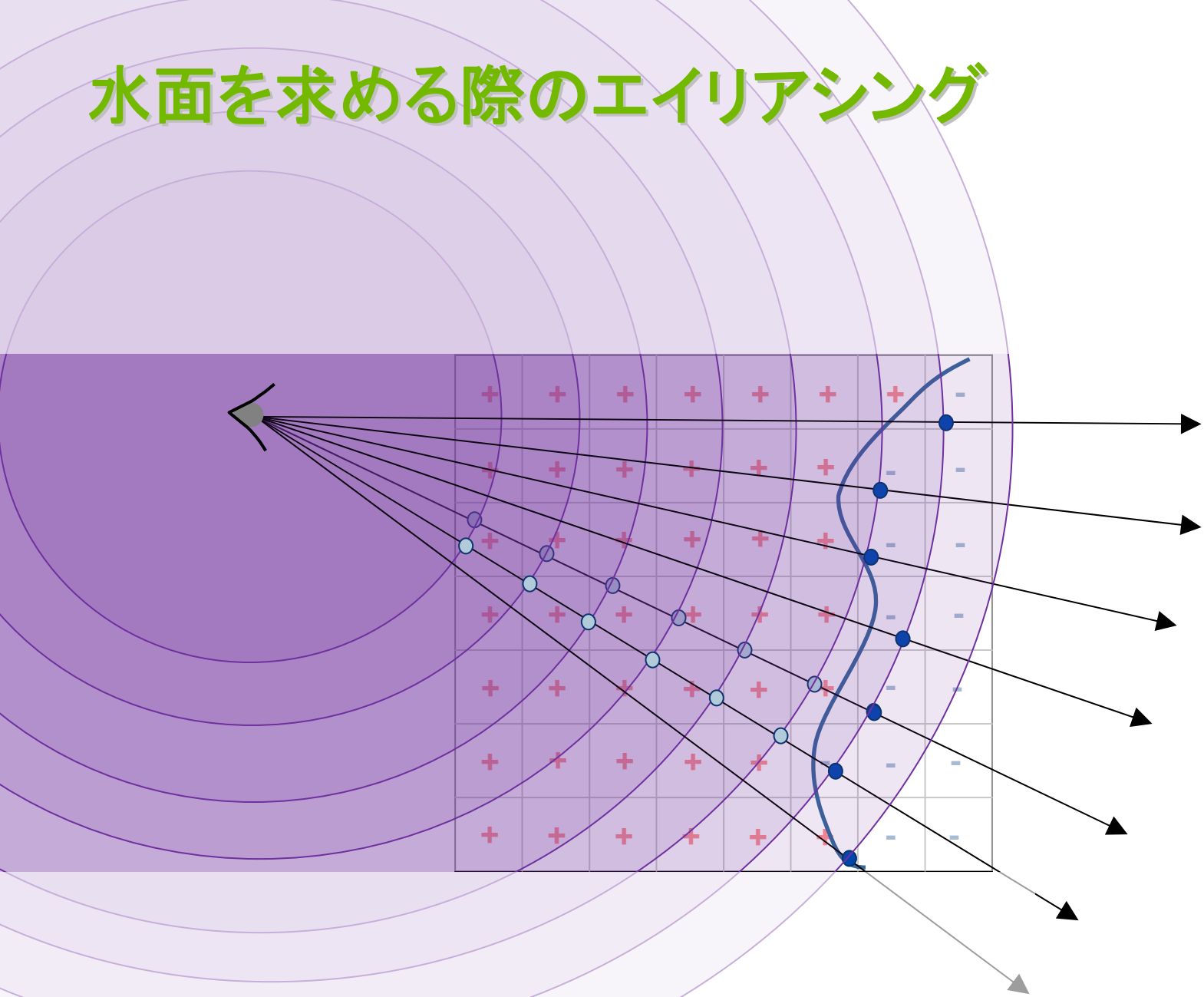
サンプリングによるエイリアシング

	現状	目標
最終 レンダリング		
グラジエント または法線		

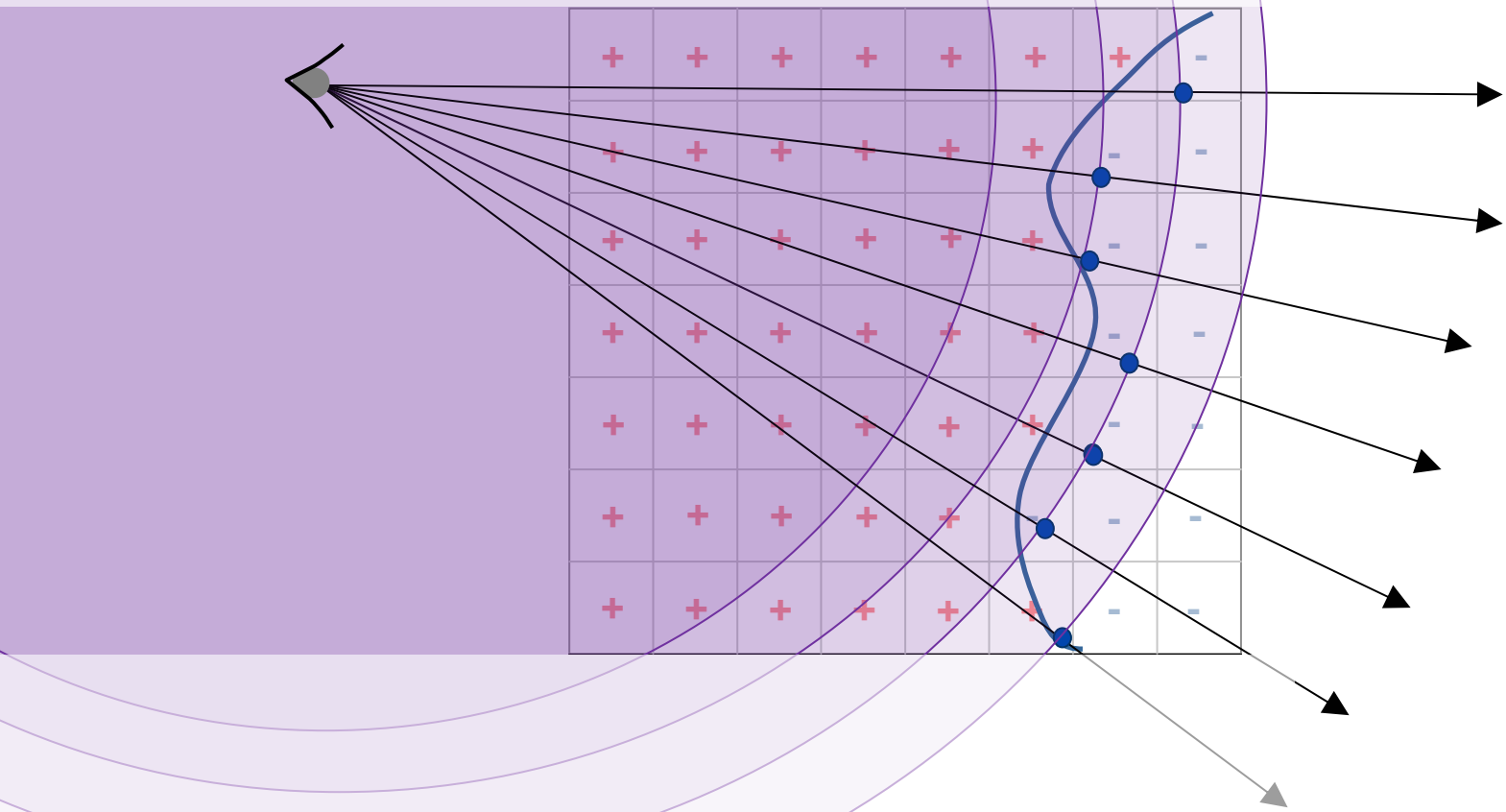
水面を求める際のエイリアシング



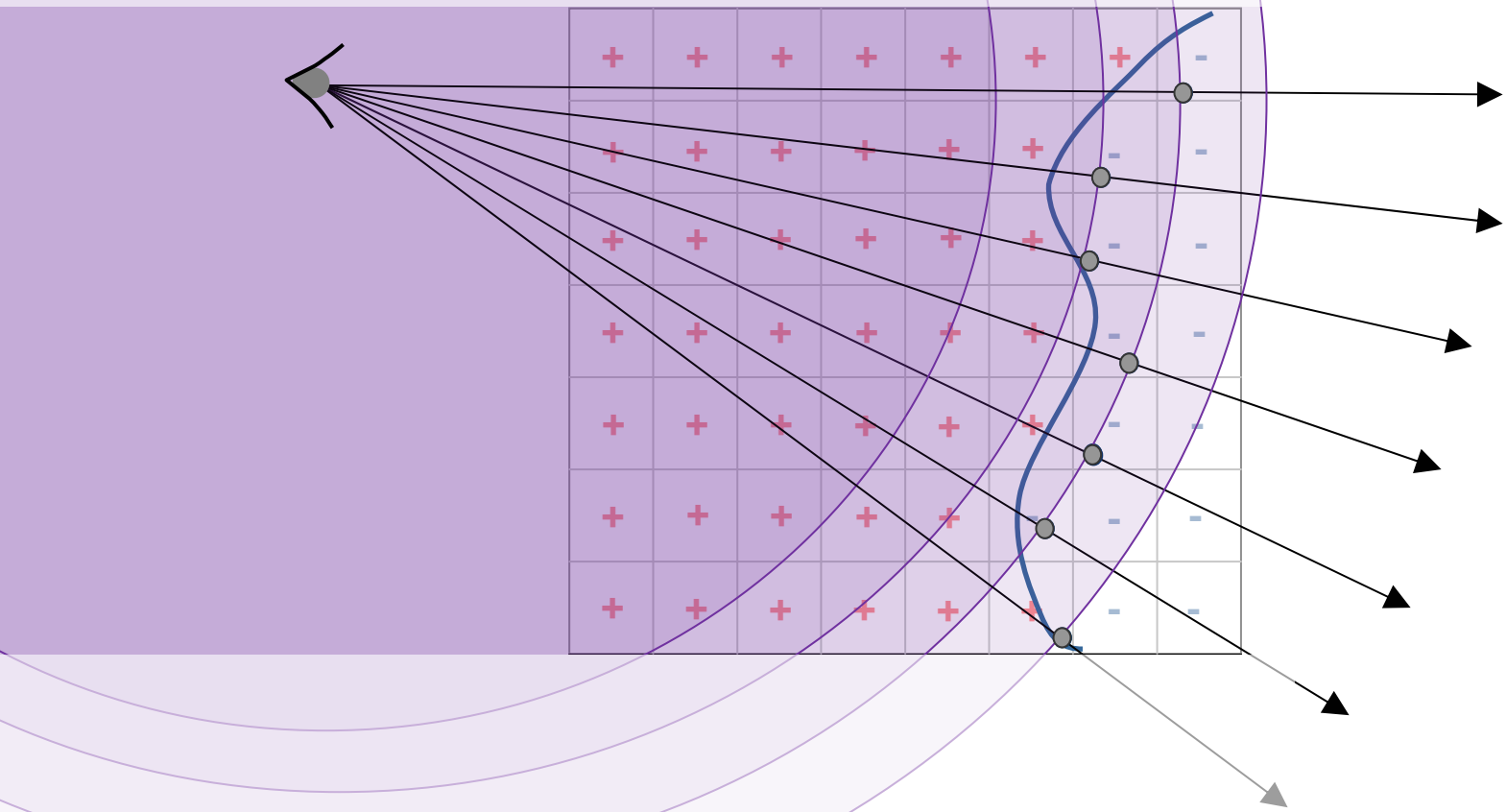
水面を求める際のエイリアシング



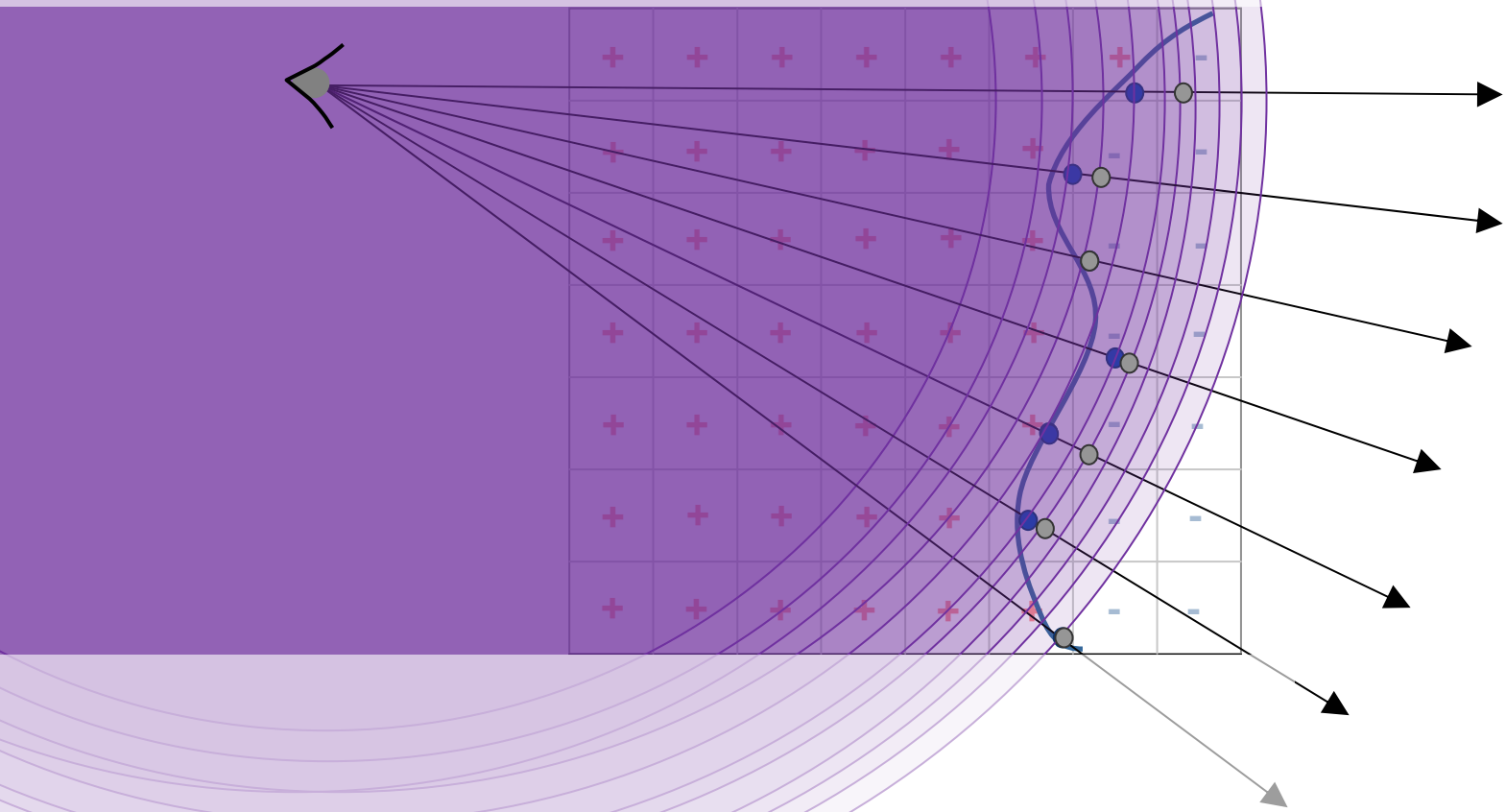
サンプル数の増加



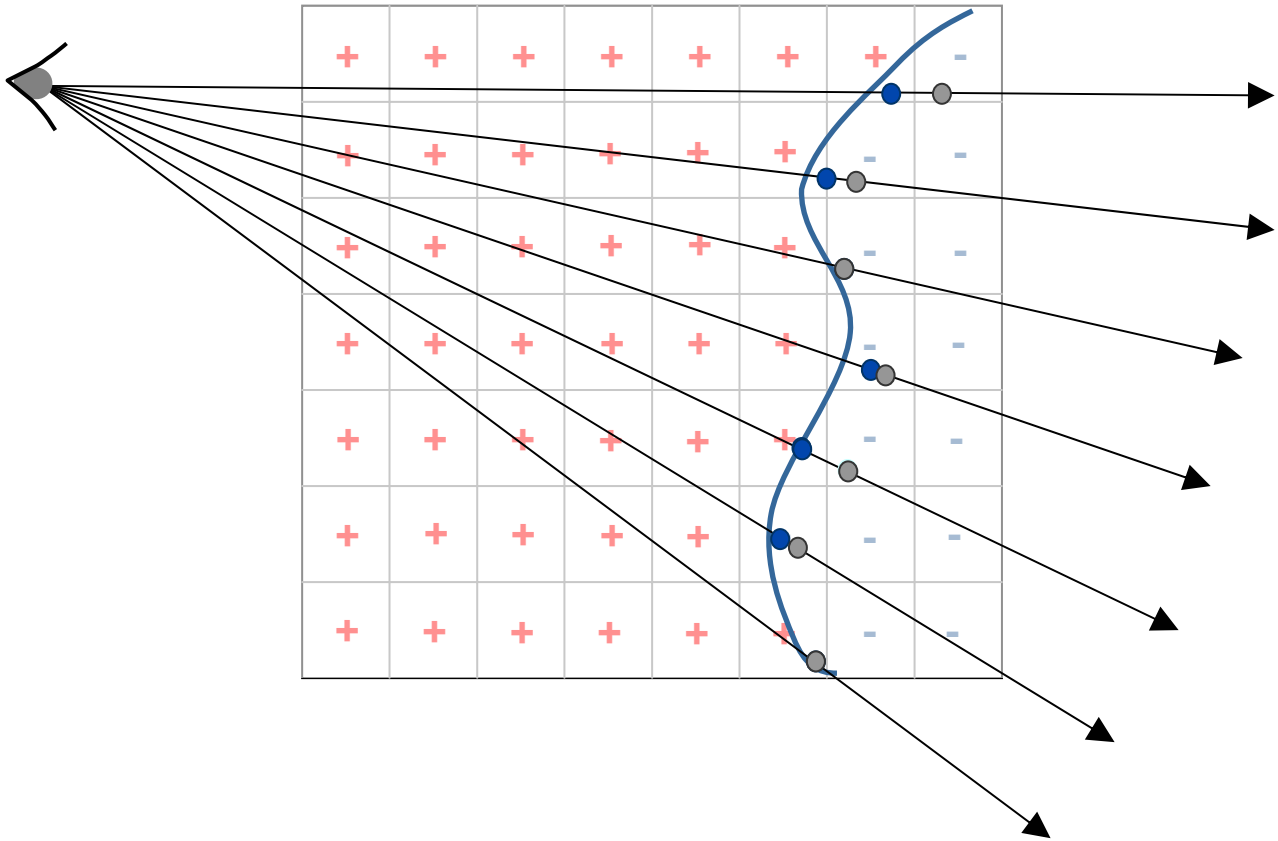
サンプル数の増加



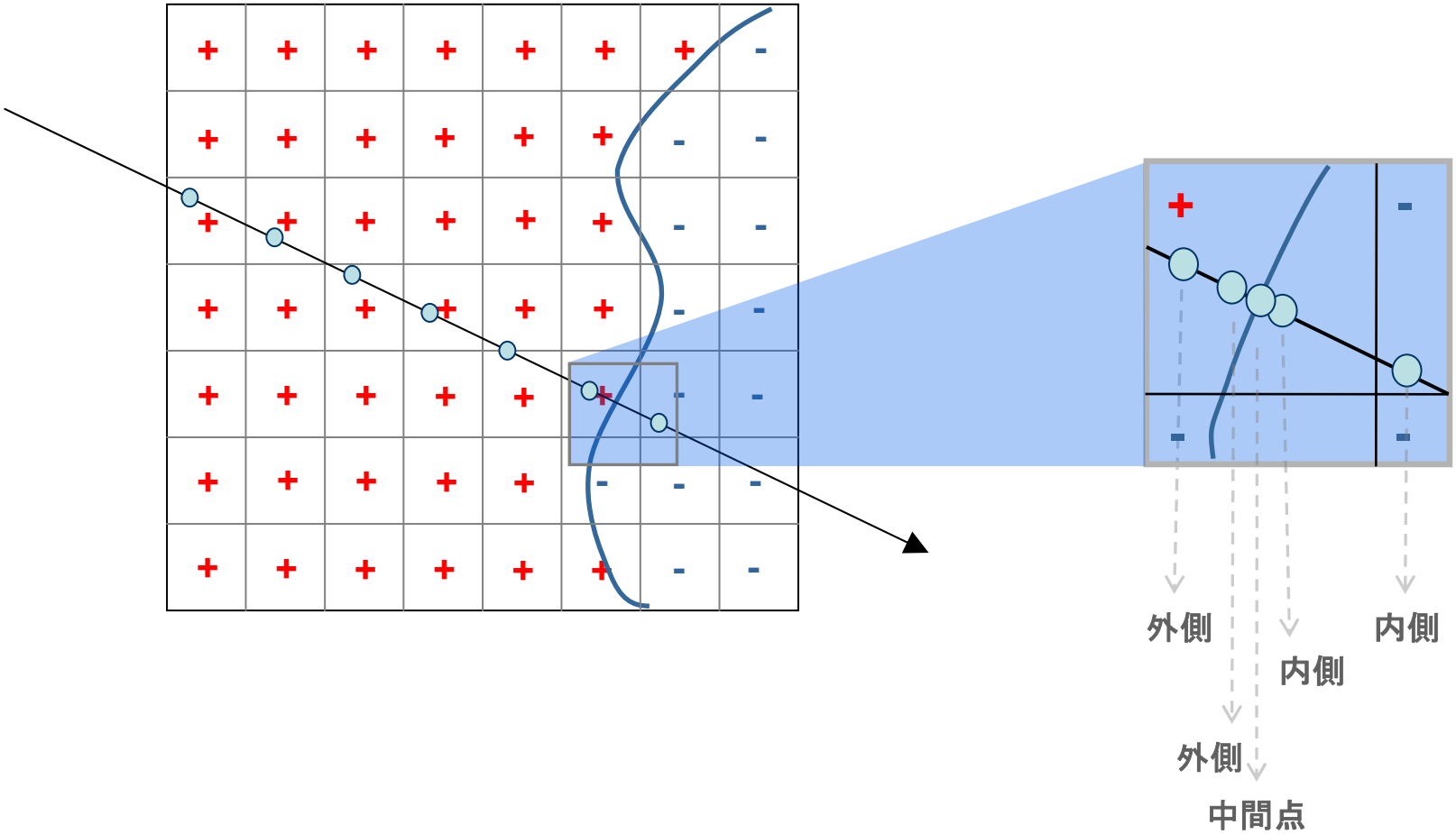
サンプル数の増加



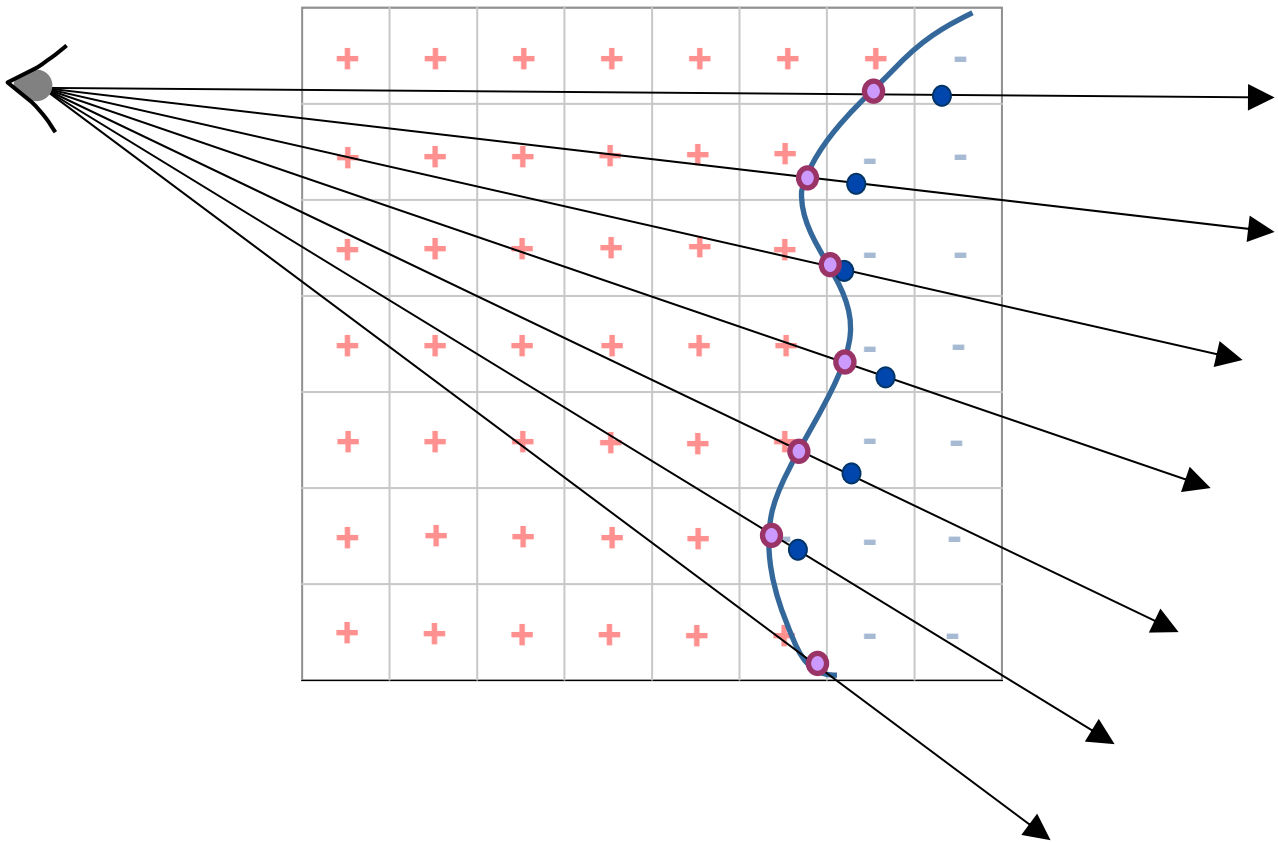
サンプル数の増加



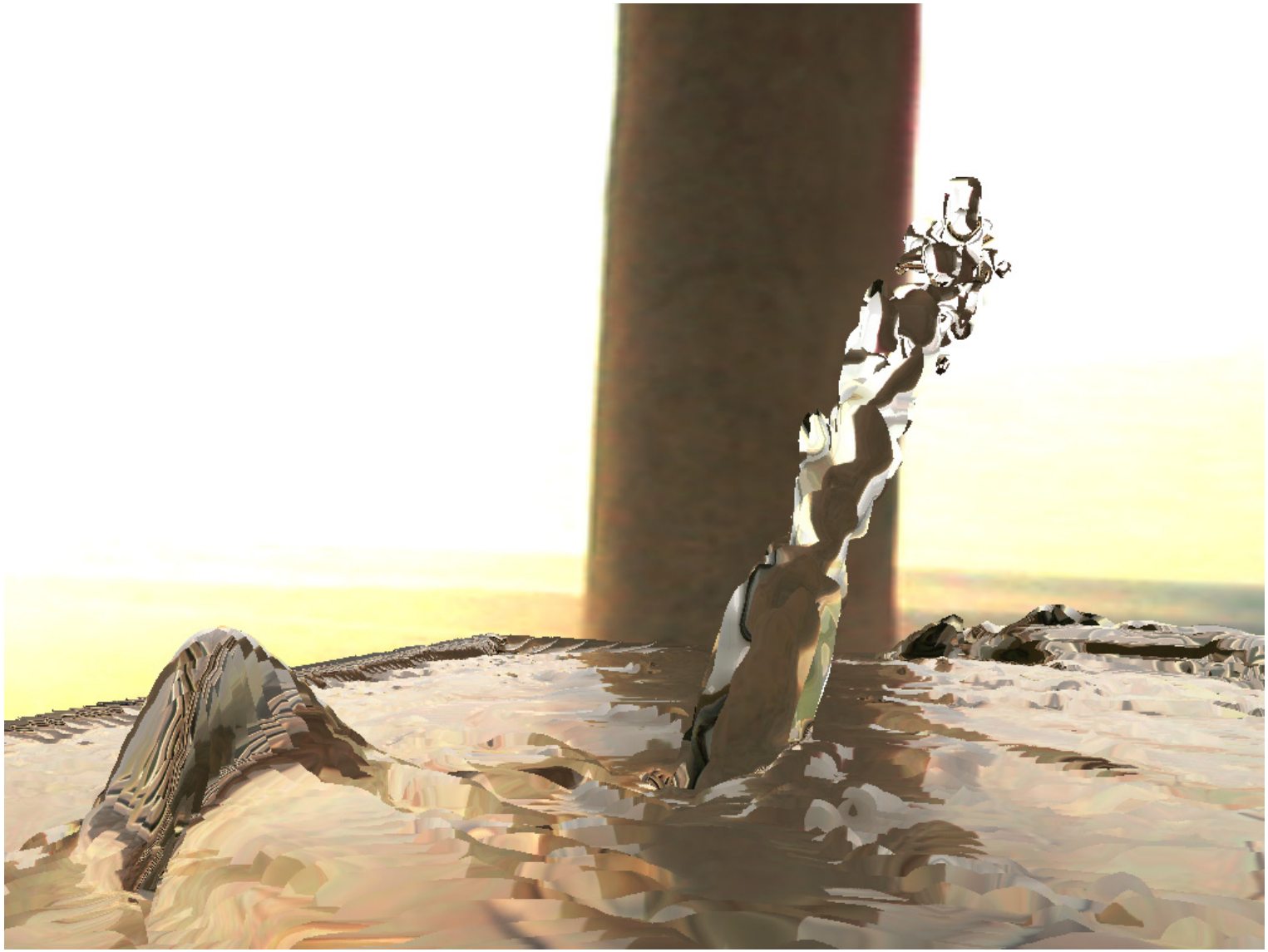
水面を求めるための二分探索



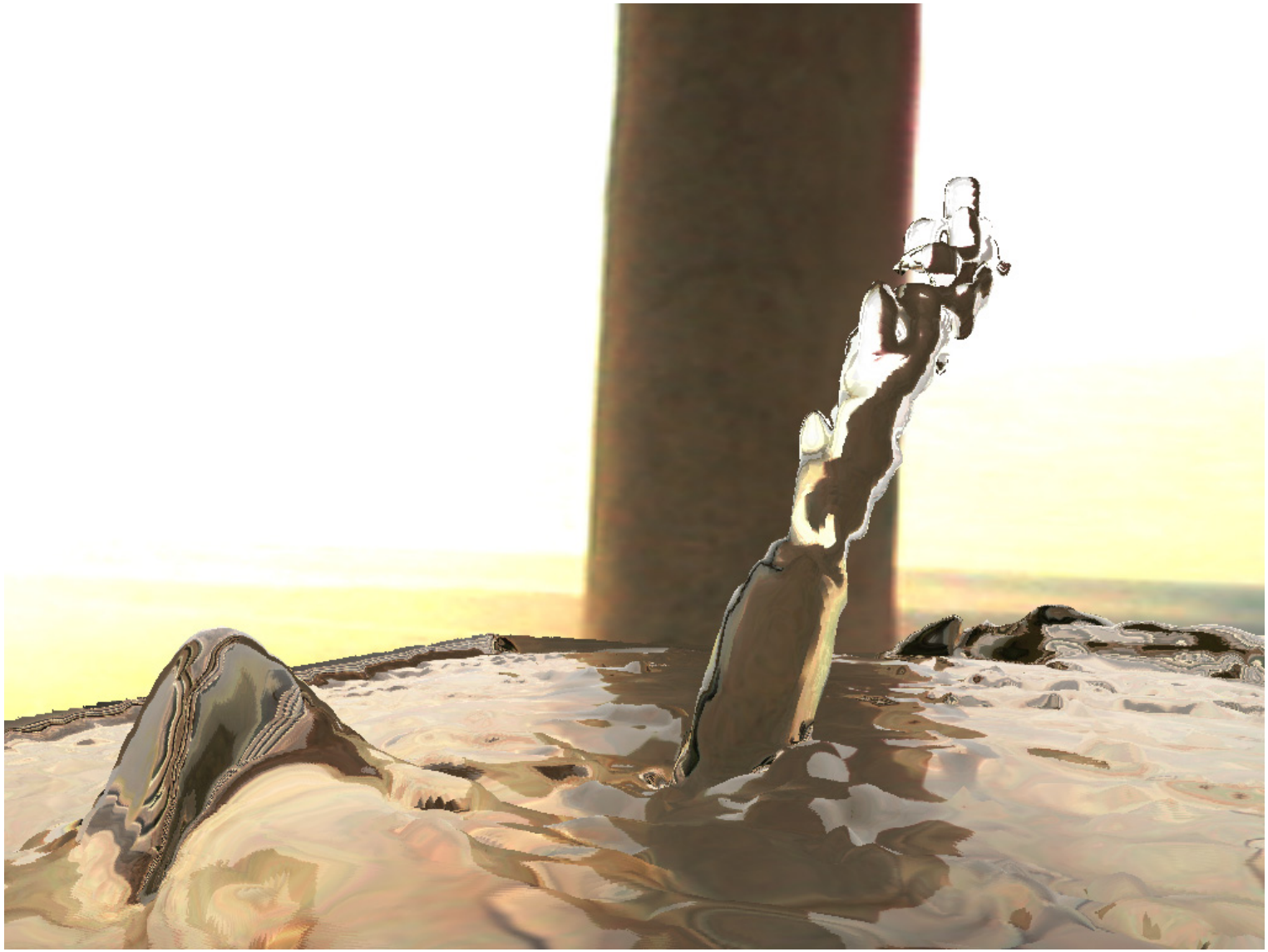
比較



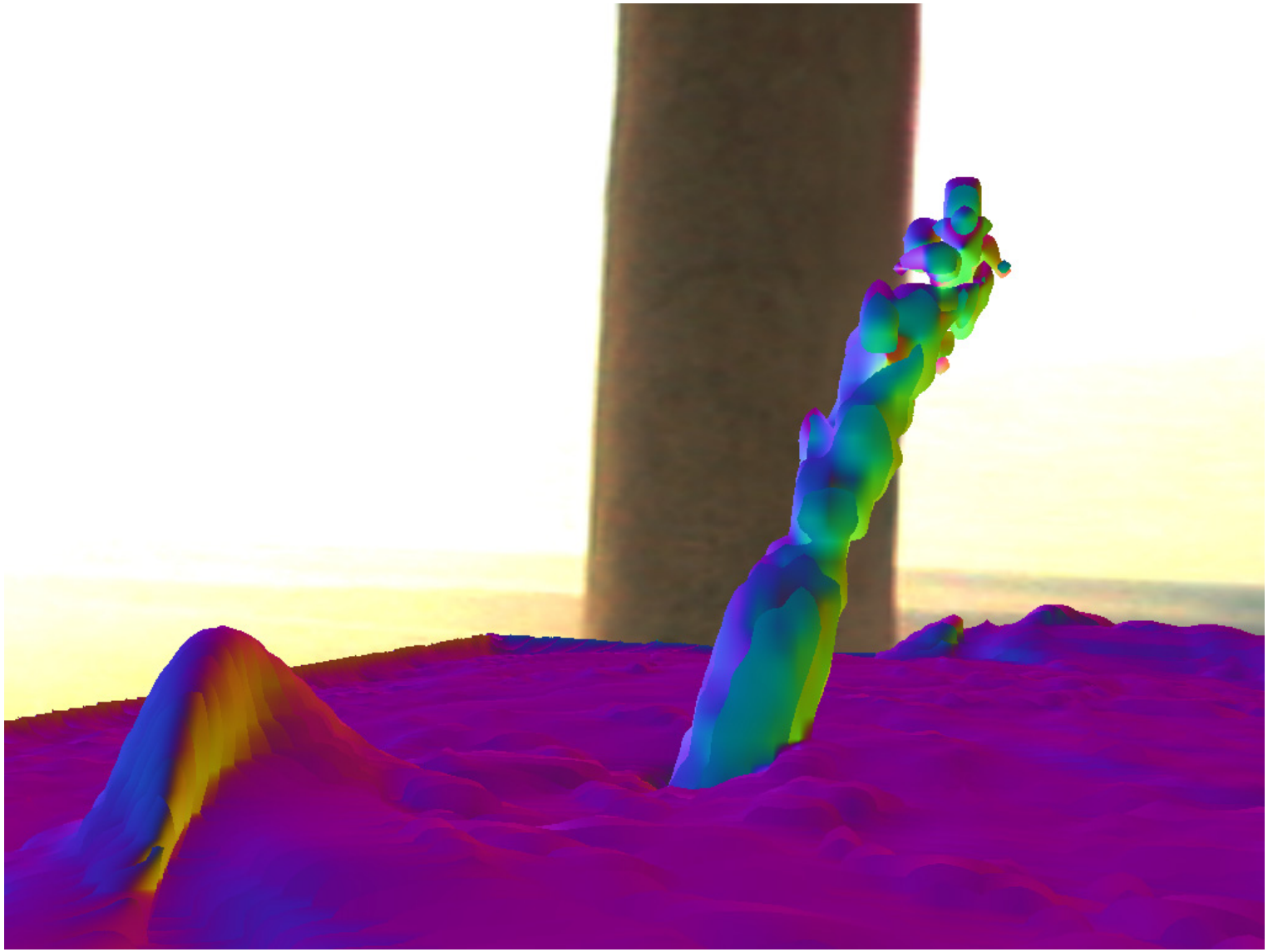
トリリニア フィルタリング－二分探索なし



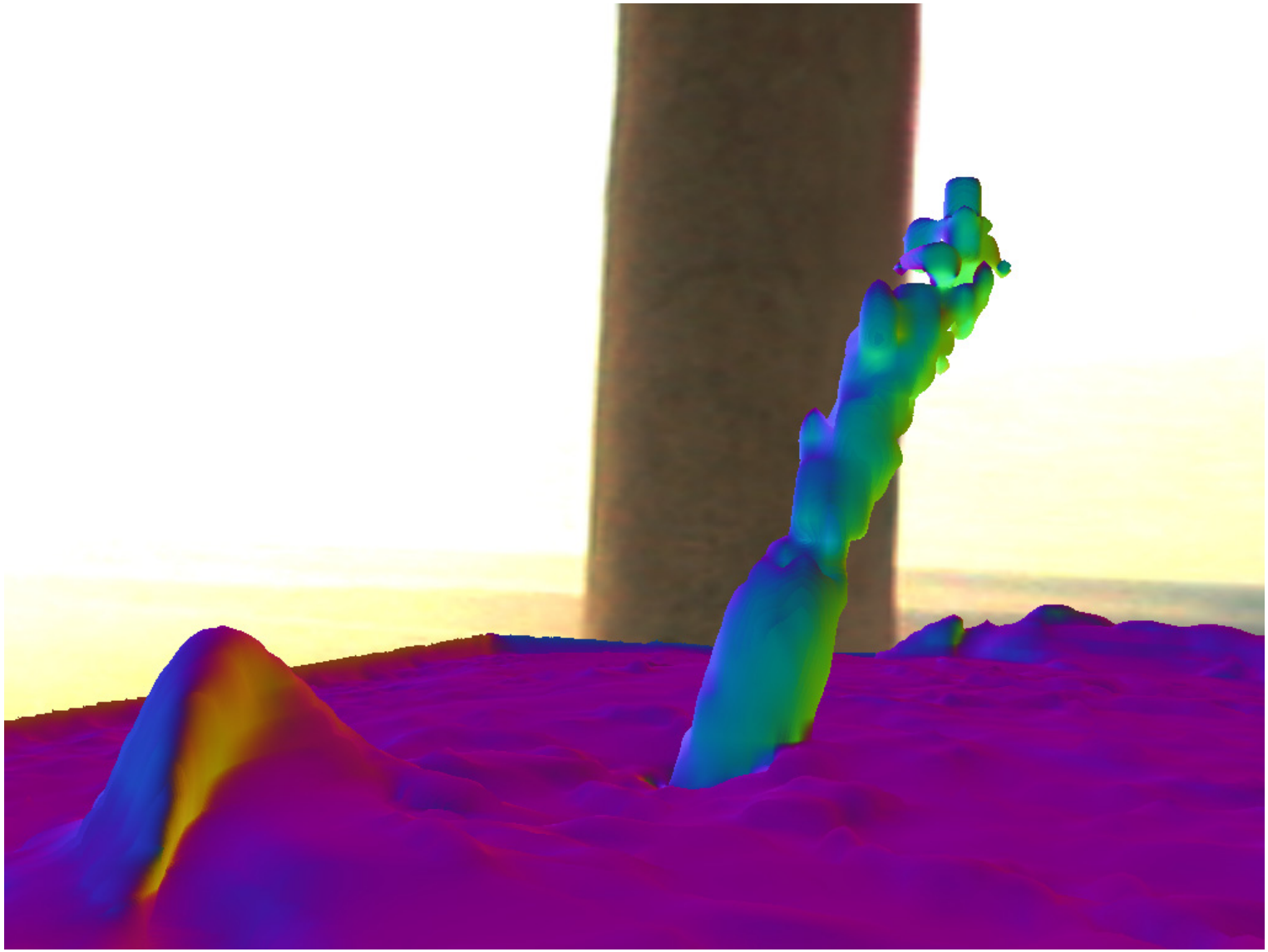
トリリニア フィルタリング ー二分探索あり



トリリニア フィルタリング ー 二分探索なし



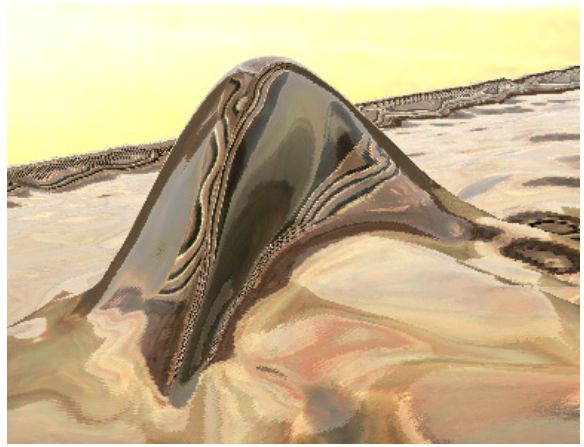
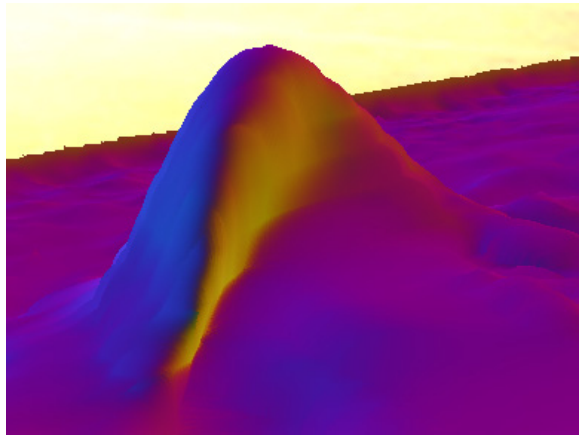
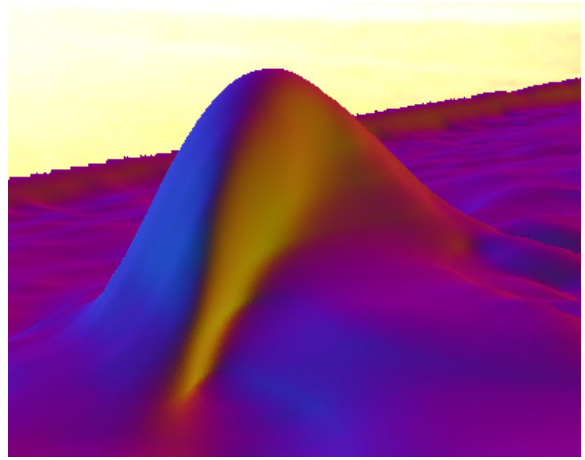
トリリニア フィルタリング ー 二分探索あり



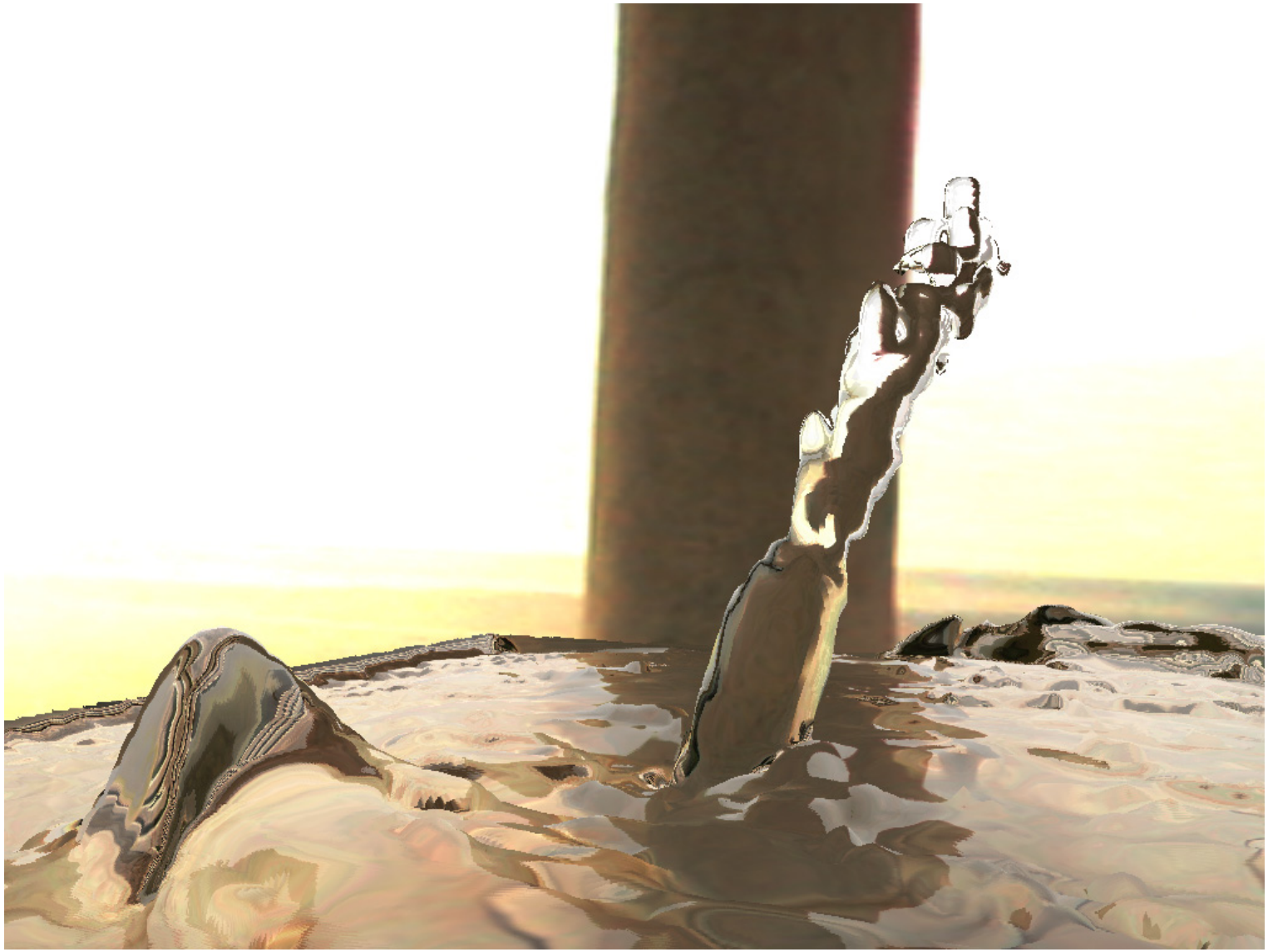
Tricubicフィルタリングの追加

- トリリニア フィルタリングの代わりにTricubicフィルタリングを使用すると、より高画質で処理が早くなる
- トリリニア フィルタリングは、ハードウェアによる単一テクスチャ検索で提供される
- Tricubic－8つのトリリニア ルックアップ＋いくつかの計算
- 位置検索とグラジエント計算の両方にTricubicを使用する

Tricubic補間

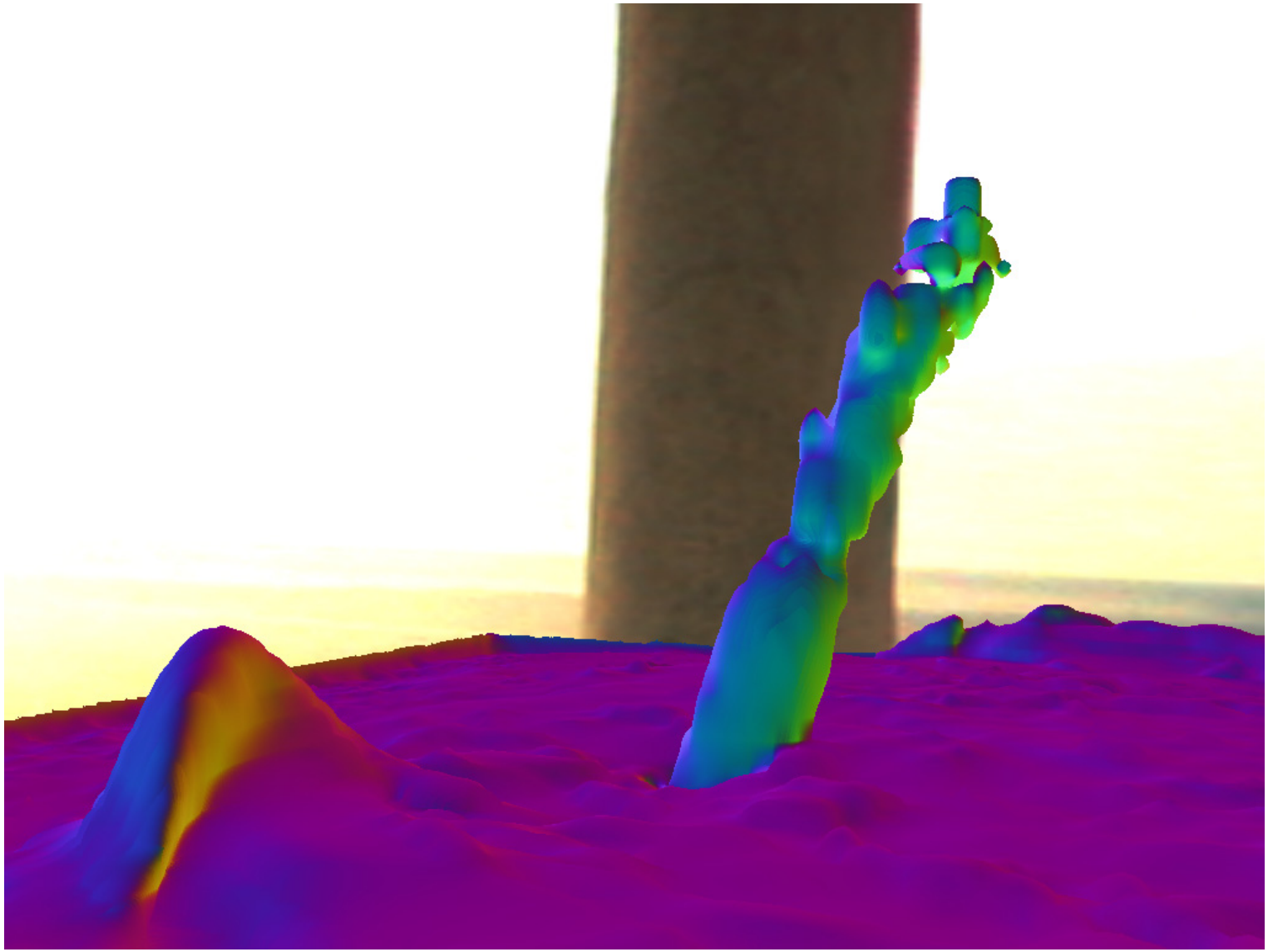
	トリリニア	Tricubic
最終 レンダリング		
グラジエント		

トリリニア フィルタリング ー二分探索あり

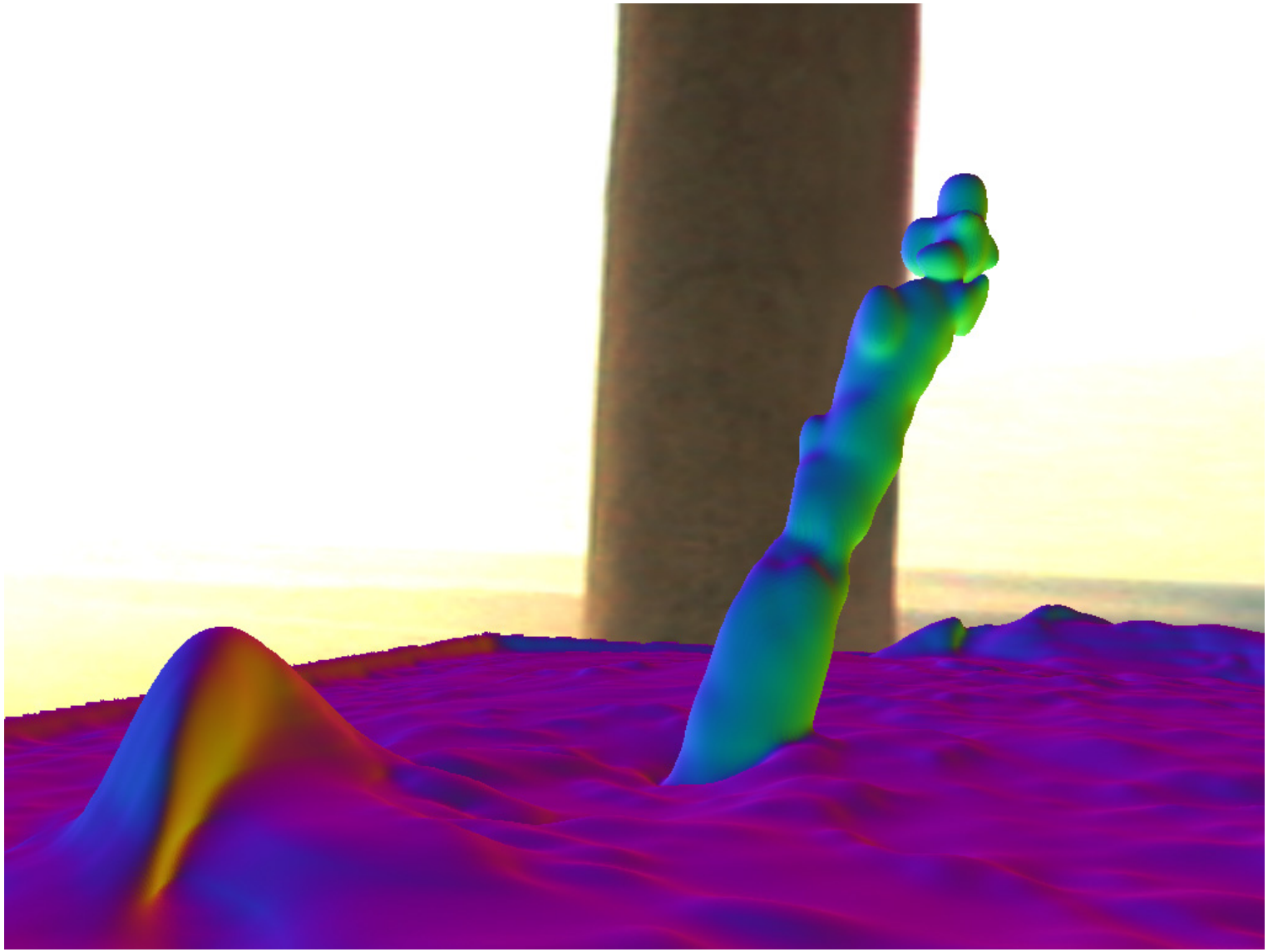


Tricubicフィルタリング－二分探索あり

トリリニア フィルタリング ー 二分探索あり

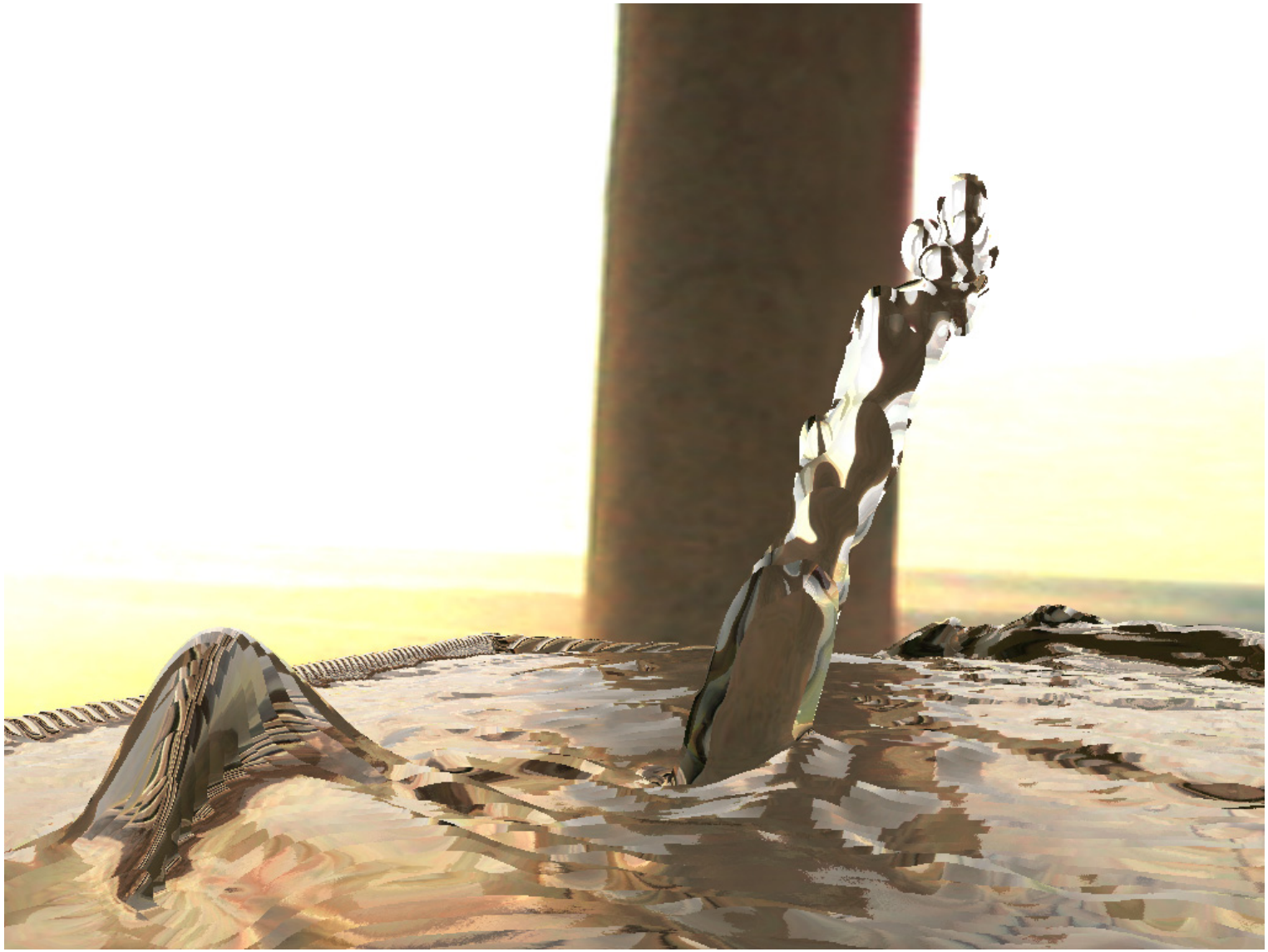


Tricubicフィルタリング－二分探索あり



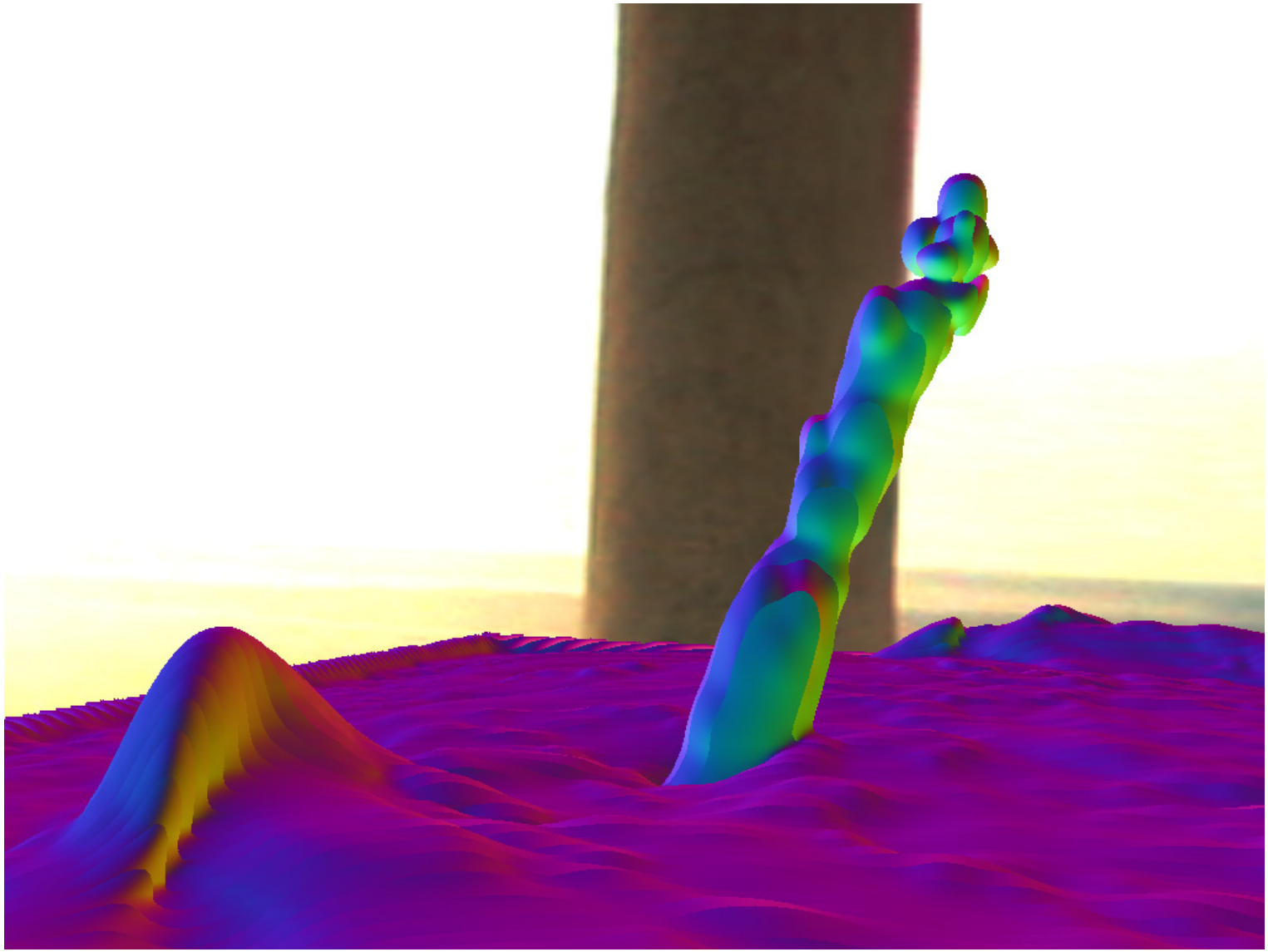
Tricubic自体は十分でない

Tricubicフィルタリング－二分探索なし



Tricubic自体は十分でない

Tricubicフィルタリング－二分探索なし



スペース要件

	総スペース	独占	共有
シミュレーション	セルごとに32バイト	セルごとに12バイト 1 x RGBA16 2 x R16	セルごとに20バイト 2 x RGBA16 2 x R16
ボクセル化	セルごとに9バイト	-	セルごとに9バイト 1 x RGBA16 1 x R8
レンダリング	ピクセルごとに20バイト 画面の解像度の場合	-	ピクセルごとに20バイト 1 x RGBA32 1 x R32
	ピクセルごとに36バイト 小さなレンダリング対象の解像度の場合	-	ピクセルごとに36バイト 2 x RGBA32 1 x R32

デモンストレーション用スペース要件

グリッドサイズ： 70×70×100
画面の解像度: 1280 × 1024
小さなレンダリング対象の解像度: 519 × 393

	総スペース	独占	共有
シミュレーション	14.95 MB	5.6 MB	9.3 MB
ボクセル化	4.2 MB	-	4.2 MB
レンダリング	26 MB	-	26 MB
	7 MB	-	7 MB

結論

- 合理的なグリッド解像度でのインタラクティブ3D流体シミュレーションは、ゲームに適している
- ここでは、プロセス全体の概要を短く説明する
- 詳細については以下を参照
  - NVIDIA DirectX10 SDKコード サンプル
    - http://developer.download.nvidia.com/SDK/10/direct3d/samples.html
  - GPU Gems 3アーティクル
    - 『Real-Time Simulation and Rendering of 3D Fluids』 *Keenan Crane, Ignacio Llamas and Sarah Tariq. Chapter 30*
    - 『High Speed, Off-Screen Particles』 *Iain Cantlay, Chapter 23*

参考文献と謝辞

NVIDIAデベロッパ テクノロジー チームのKeenan Crane、Chris Kim、
およびHellgate:Londonのデベロッパの協力に感謝します

貴重な先行研究:

- 『Hardware Accelerated Voxelization』 *S. Fang and H. Cheng. Computers and Graphics, 2000*
- 『Real-Time Fluid Dynamics for Games』 *Jos Stam, GDC 2003*
- 『Simulation of Cloud Dynamics on Graphics Hardware』 *Mark Harris, W. Baxter, T. Scheurmann, A. Lastra. Eurographics Workshop on Graphics Hardware. 2003*
- 『Fast Fluid Dynamics Simulation on the GPU』 *Mark Harris, NVIDIA. GPU Gems 2004*
- 『Explicit Early-Z Culling for Efficient Fluid Flow Simulation and Rendering』*Pedro V. Sander, Natalya Tatarchuk, Jason L. Mitchell, ATI Research Technical Report, August 2004*
- 『Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces』 *Markus Hadwiger, Christian Sigg, Henning Scharsach, Katja Bühler, Markus Gross, 2005*

その他

ご質問

GPU Gems 3が登場！

- SIGGRAPH Bookstore
  - 主な書籍販売店

- 以下の章を含む
  - Adobe Systems
  - Apple
  - Crytek
  - Cornell University
  - Electronic Arts
  - Havok
  - Juniper Networks
  - Microsoft
  - SEGA
  - その他