ムケテ、未来。

CEDEC 2008

CESA DEVELOPERS CONFERENCE 2008

FOR NEXT 10 YEARS

プロシージャル グラフィックス

理論と実践

株式会社バンダイナムコゲームス

コンテンツ制作本部　制作統括ディビジョン

技術部　プロジェクトサポート課

今給黎 隆

本日の内容

- プロシージャルとは？
- プロシージャルなシェーディング
- 移流テクスチャ
- 樹木生成

ムケテ、未来。
CEDEC 2008
CESA DEVELOPERS CONFERENCE 2008
FOR NEXT 10 YEARS

プロシージャルとは？

プロシージャルとは？

- 教科書的な定義
  - 簡潔なアルゴリズムの組み合わせで複雑な処理を実現
- 今回のCEDECの別なお話
  - 全体的な概要:「IMAGIRE DAY(3) ゲーム開発のためのプロシージャル技術の応用」三宅　陽一郎
  - 学術的、歴史的なお話:「プロシージャル技術の動向」宮田　一乗 9月11日(木) 14:50～16:10

他のジャンルとの関わり

- さまざまな分野を横断
  - CG, AI, 物理, 音楽, ゲームデザイン

ゲームデザイン

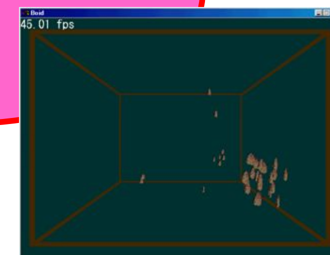
物理シミュレーション

音楽

プロシージャル

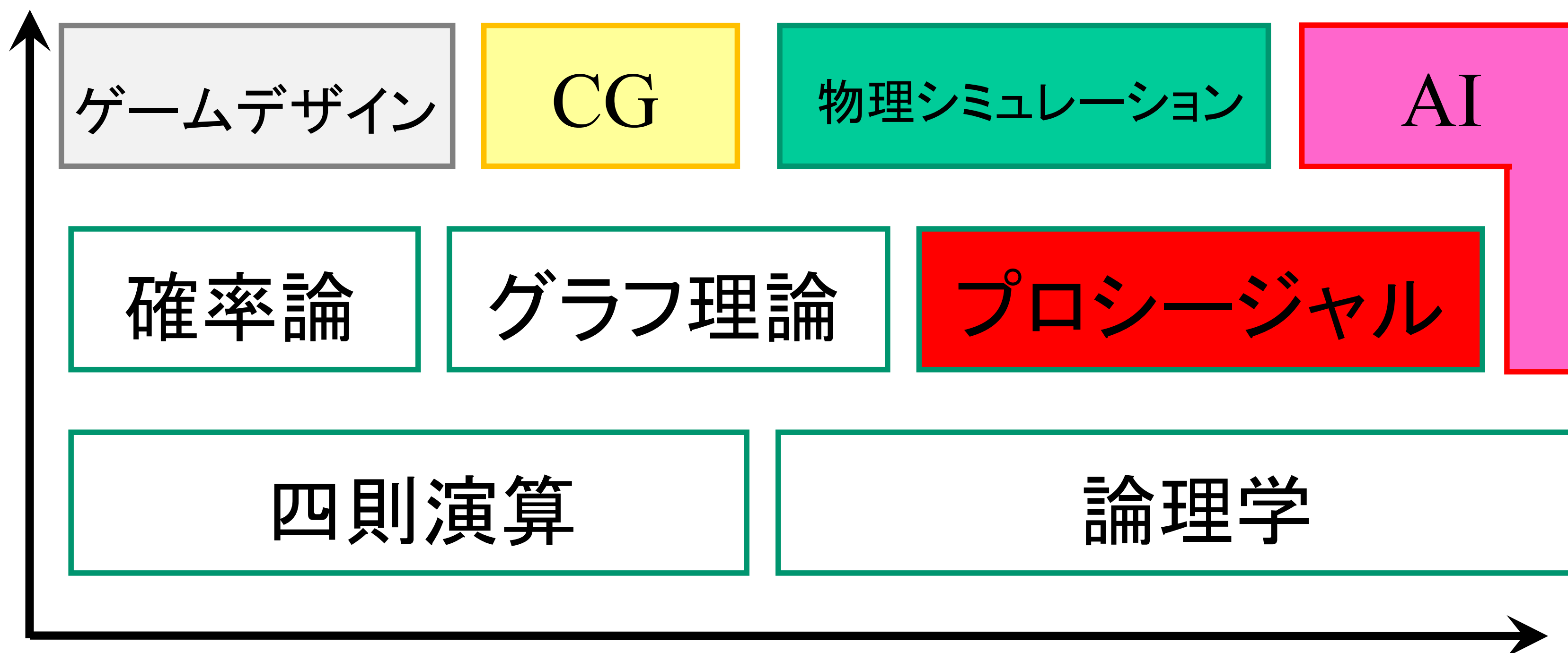
CG

AI



プロシージャルは分野じゃない

- 各分野を下支えする技術



注目されている理由

- すごい物量への対応
  - SPORE
  - Far Cry 2 (50km 四方の世界)
- 動的な環境へ対応
  - Far Cry 2 (壊した植物が復活、天候の変化)
  - SPORE (多彩な生物)
  - 負荷コントロール
    - 見えにくいものは、途中で
      アルゴリズムを止めたりすれば
      良いんじゃない？

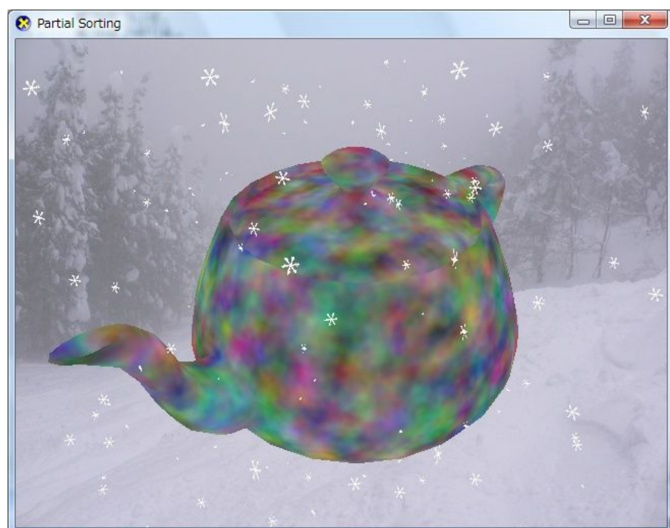
グラフィックスに関係したものは？

- 地形
  - Terragen (http://www.planetside.co.uk/terragen/)
- 植物
  - Speed Tree
- 空・雲
- 街
  - CityEngine (http://www.procedural.com/)
- テクスチャ
- 群衆（NPCモデリングの自動化）

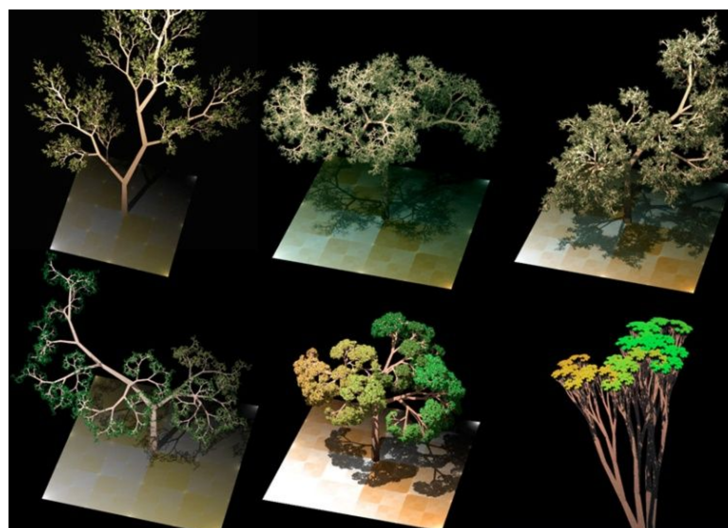
木目

例えばどんな技法

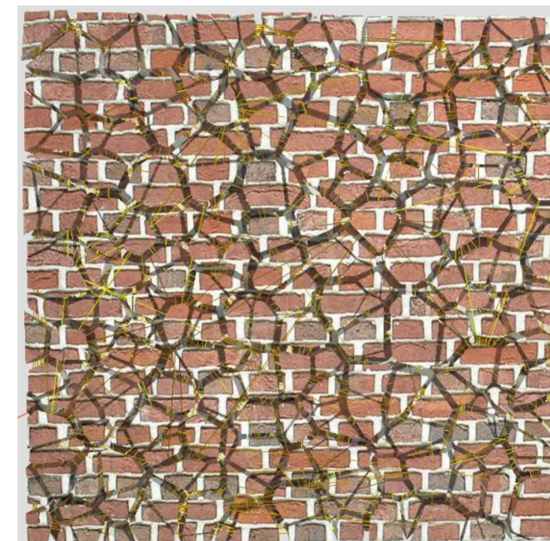
- ノイズ関数
- L-System
- ドロネー図、ボロノイ図



３次元ノイズ



L-system による木



ボロノイ図で亀裂作成

フラクタルノイズ

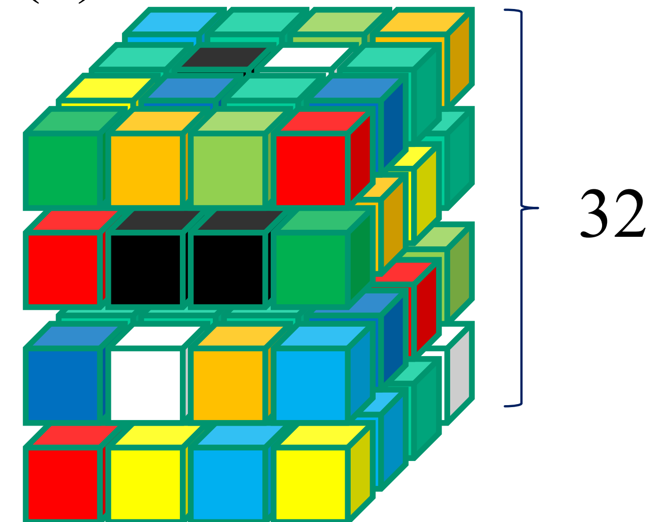
- 乱雑な連続関数f(x)を用意し、周波数と振幅を変えながら足し合わせ
  - “Perlin Noise” は、サンプリング方法にスムージングを掛けていたりするので、ここでは一般のフラクタルノイズとしました

3次元のフラクタルノイズ

Partial Sorting

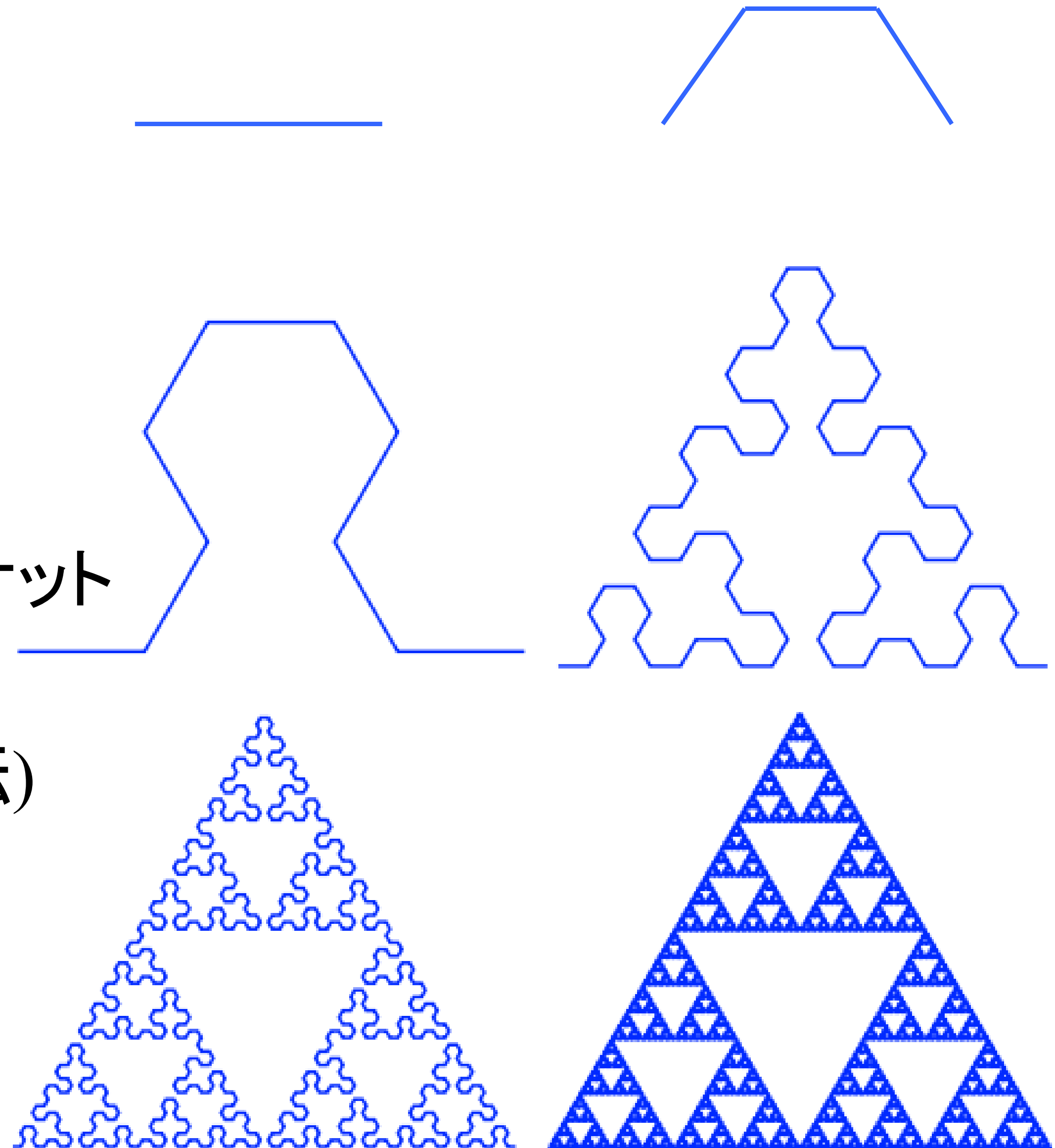
$$T(x)+\frac{1}{2}T(2x)+\frac{1}{4}T(4x)+\frac{1}{8}T(8x)$$

$T(x)$:

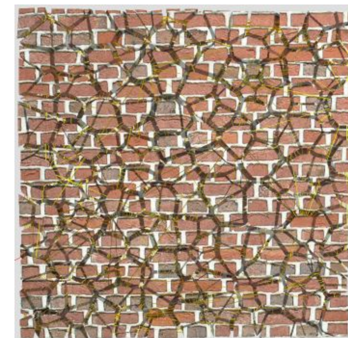


L-System

- **G** = {V, S, ω, P},
  - V: 変数
  - S: 定数
  - ω: 初期値
  - P: 変換
- シェルピンスキーのギャスケット
  - V: A, B (直線描画)
  - S: +, -(左(右)へ60度回転)
  - ω: A
  - P: (A→B-A-B), (B→A+B+A)

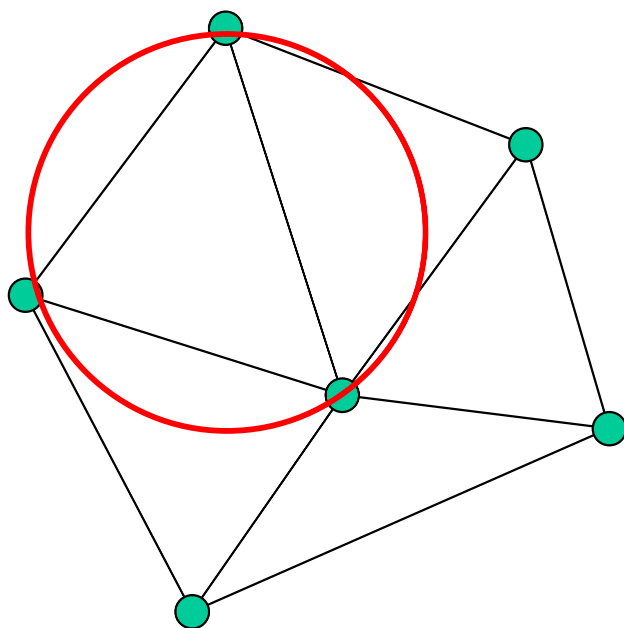


ドロネー図、ボロノイ図


ボロノイ図で亀裂作成

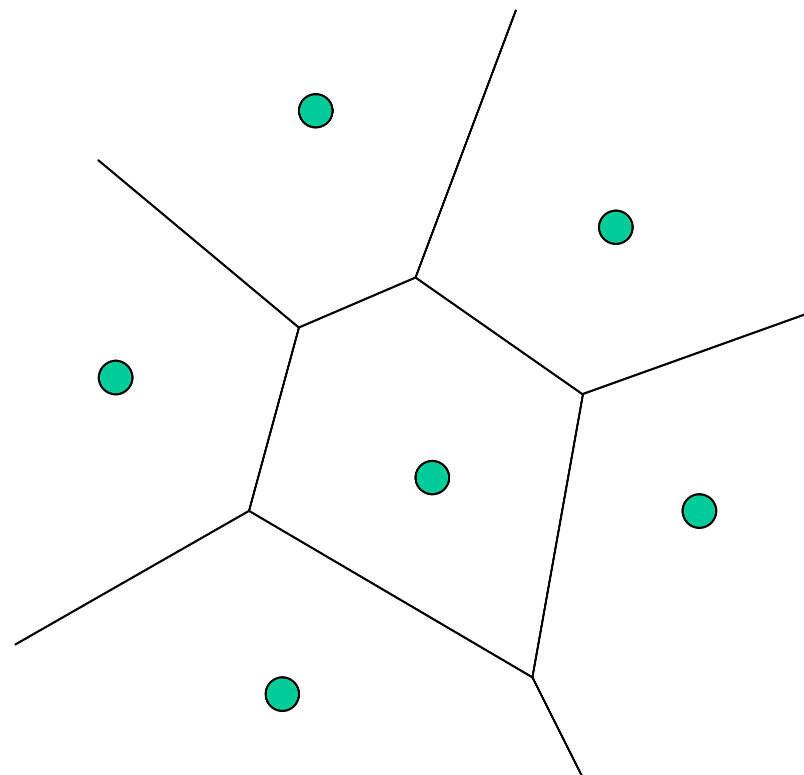
ドロネー図

頂点の集合を与えられた際の、最小の3角形 (3次元なら4面体)での分割


とがらないポリゴン分割

ボロノイ図

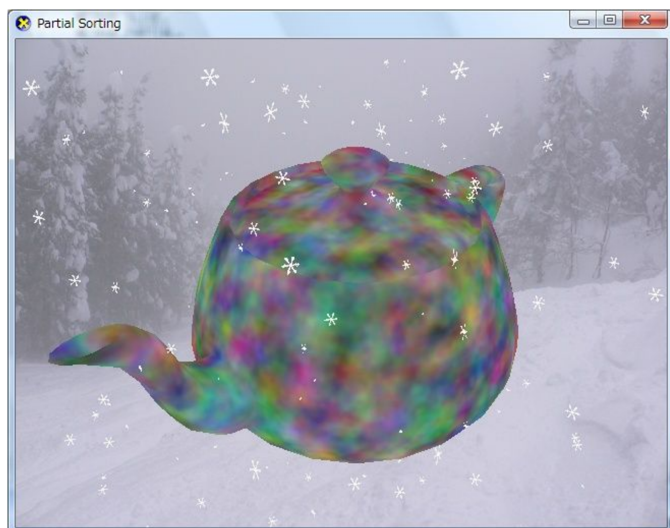
頂点の一番近い領域として空間を分割



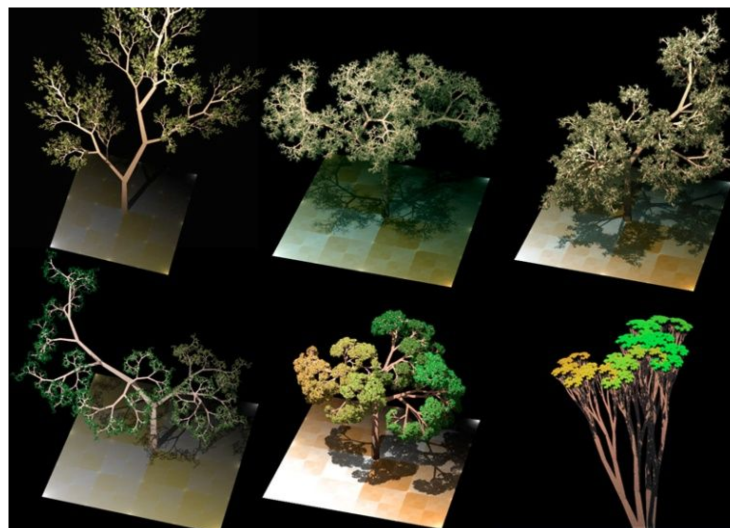
例えばどんな技法

- ノイズ関数
- L-System
- ドロネー図、ボロノイ図

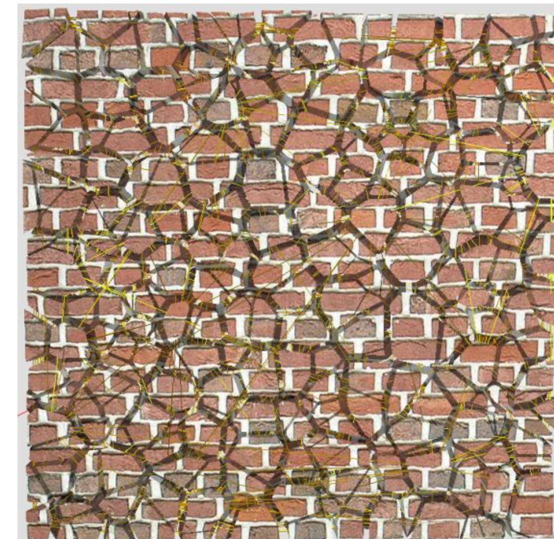
} 擬似フラクタルシステム



３次元ノイズ



L-system による木



ボロノイ図で亀裂作成

擬似って？

- 規則的なのは面白くない。
  - 単なるランダムはつまらないって分かってるでしょ?
- 変化が与えられることが大事
  - もちろん、知識(観察、感性)も重要
    - 人が歩くところは草木が生えない
    - 港が作れない土地は発展しない
    - プログラマーはめがねが多い
- 製作者の意図を組み込んだ自動化がプロシージャル & 次世代ゲーム

ムケテ、未来。

CEDEC 2008

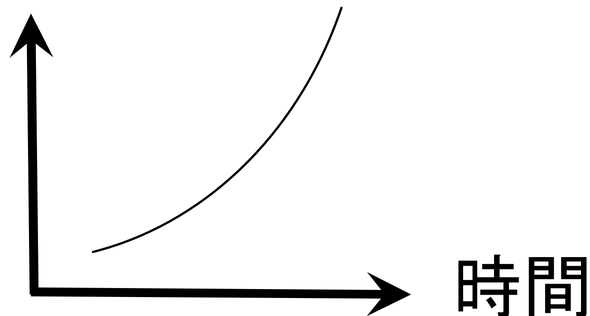
CESA DEVELOPERS CONFERENCE 2008

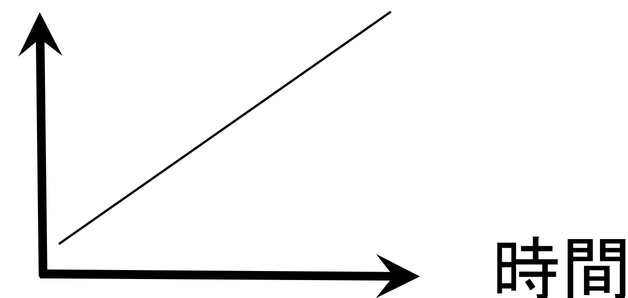
FOR NEXT 10 YEARS

プロシージャルなシェーディング

楽にデータを作りませんか？

- 「スペキュラマップ」、「法線マップ」、「ポリゴン数の増大」作るデータが多すぎ
- 時間かかりすぎじゃないですか？
  - そんなに売り上げが上がるわけではないのに…
- もっと簡単に作る方法を考えましょう。



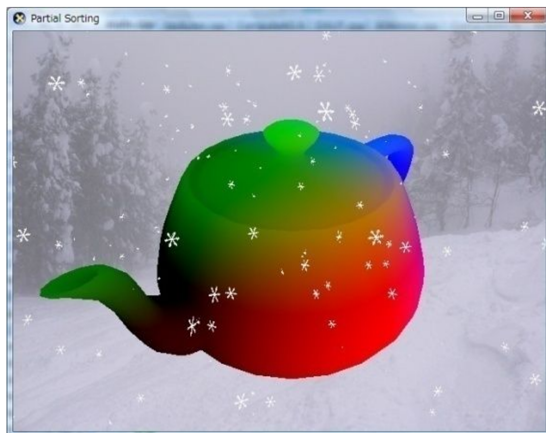


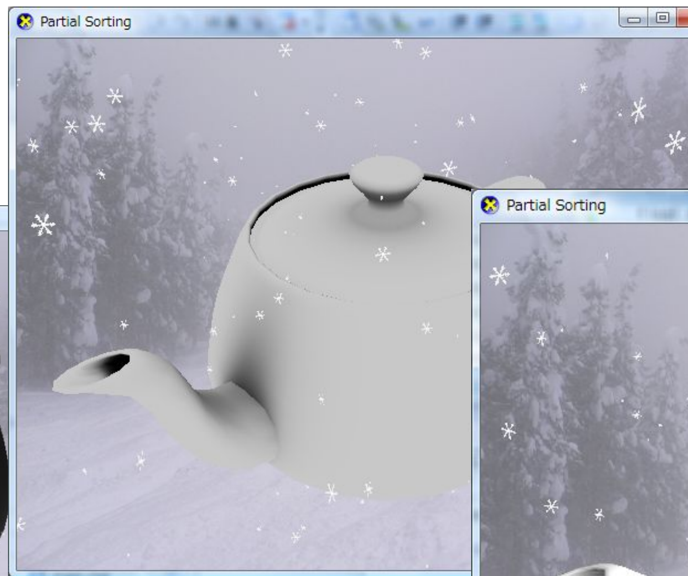
映画業界も困ってる

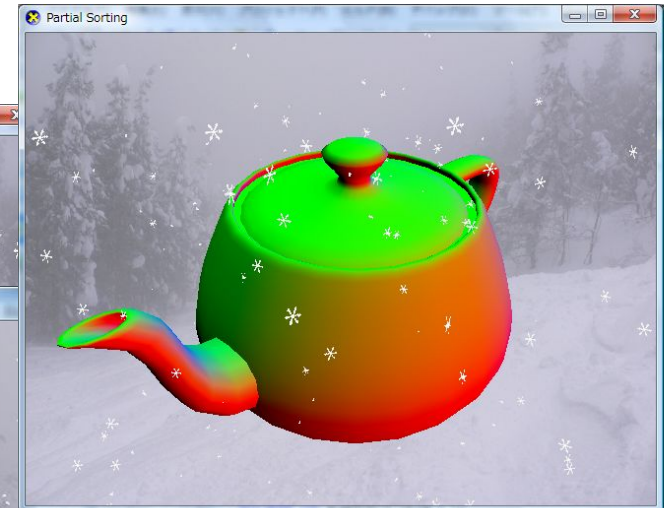
- Shading The Many: Solutions For Shading Crowd Characters on WALL•E, SIGGRAPH 2008 Talks.
  - どうやって短い期間で大量にロボットモデルを作成したか

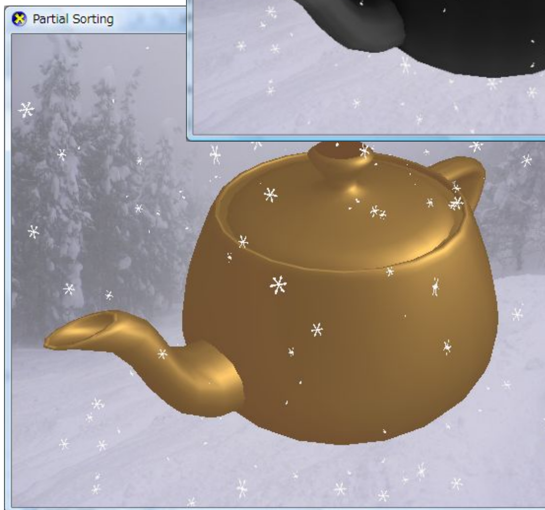
手持ちの武器

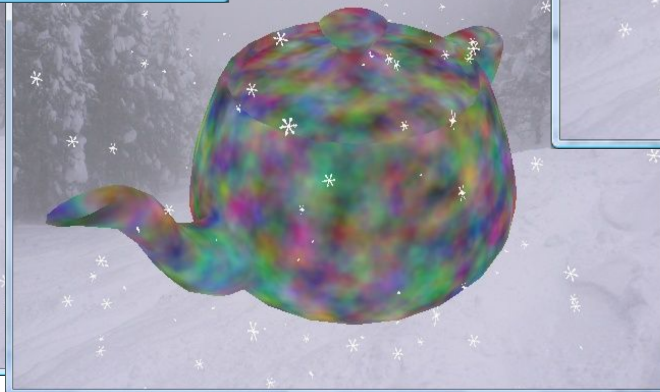
- 自動で作成できるものは？

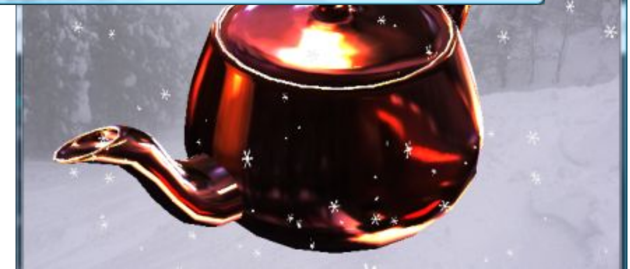




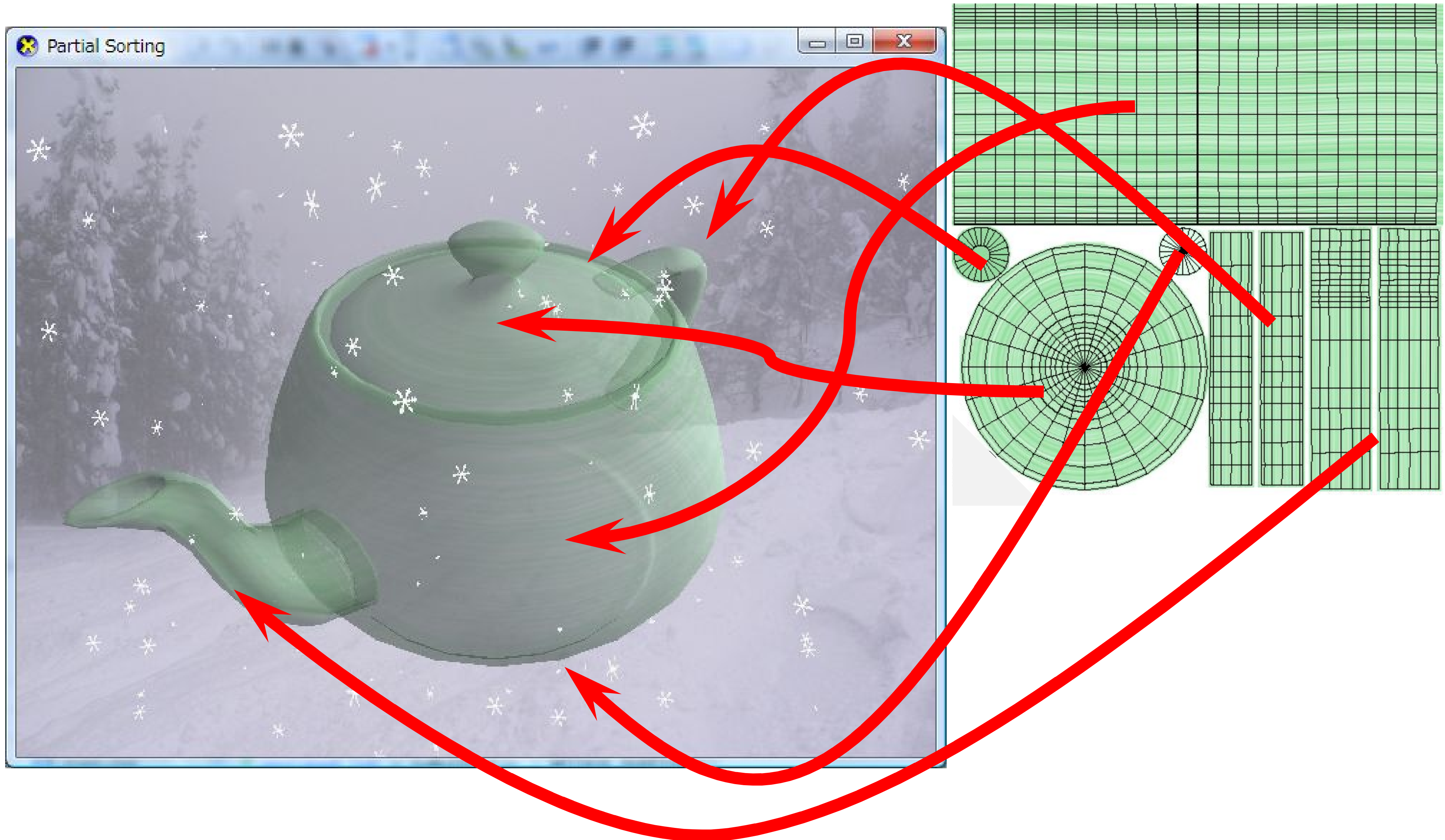








デカールテクスチャとUV座標



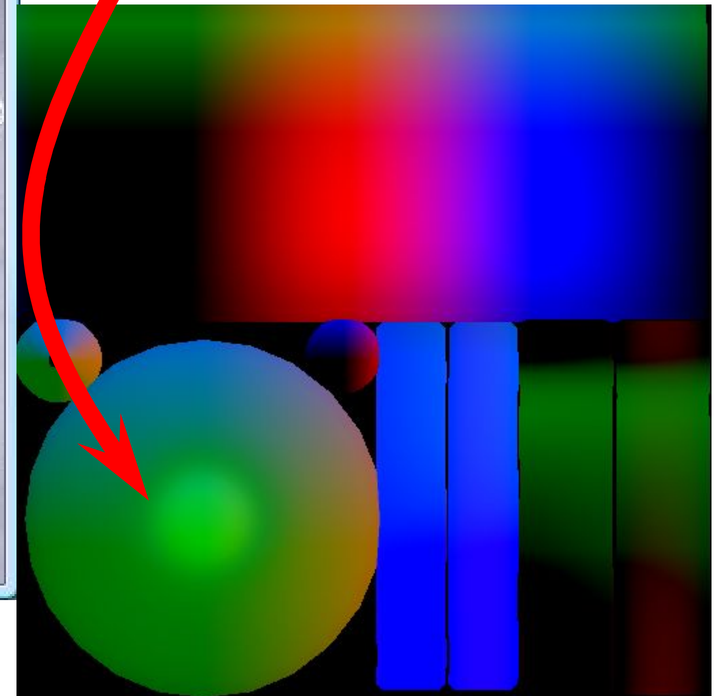
ローカル座標

Partial Sorting

(x, y, z)

⇔

(R, G, B)



法線ベクトル

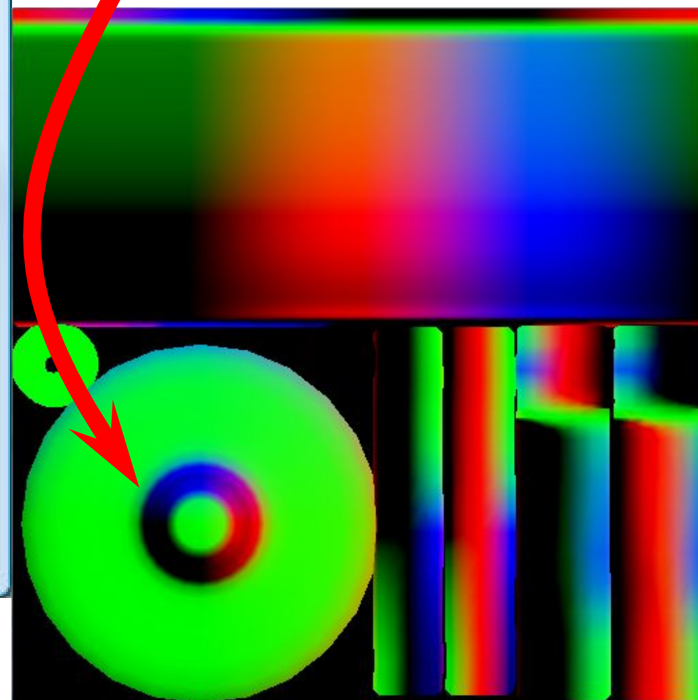
Partial Sorting

法線ベクトル

(n_x, n_y, n_z)

$\Leftrightarrow$

(R, G, B)

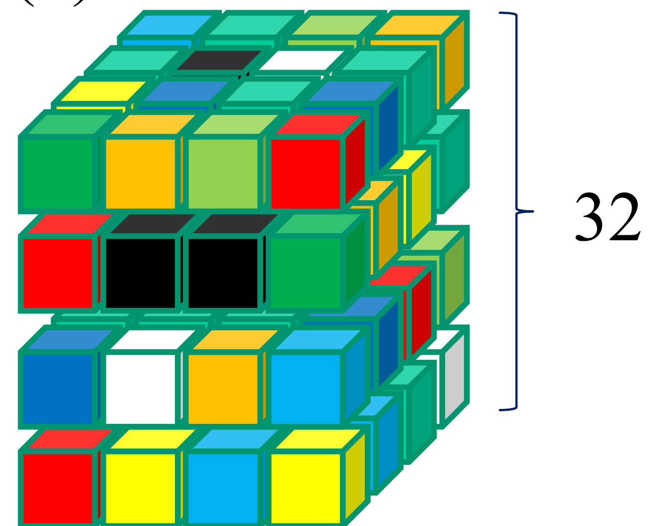


フラクタルノイズ

Partial Sorting

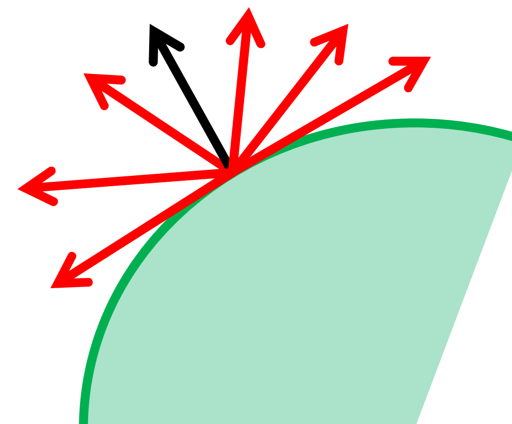
$$T(x)+\frac{1}{2}T(2x)+\frac{1}{4}T(4x)+\frac{1}{8}T(8x)$$

$T(x)$:

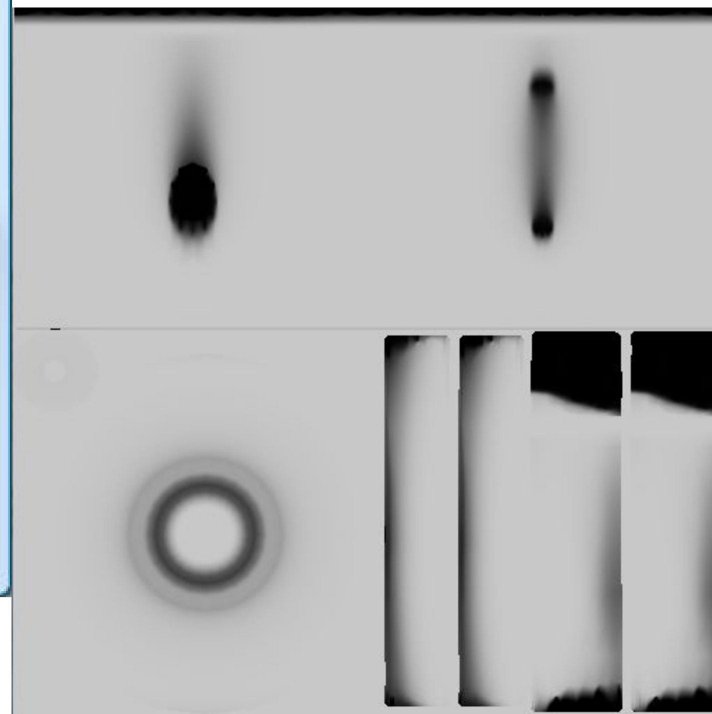


Ambient Occlusion

Partial Sorting

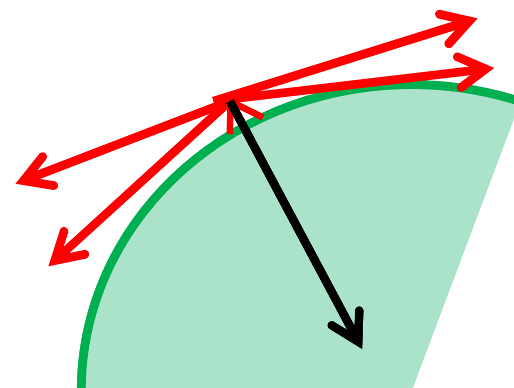


法線と反対方向に
半球分サンプリング

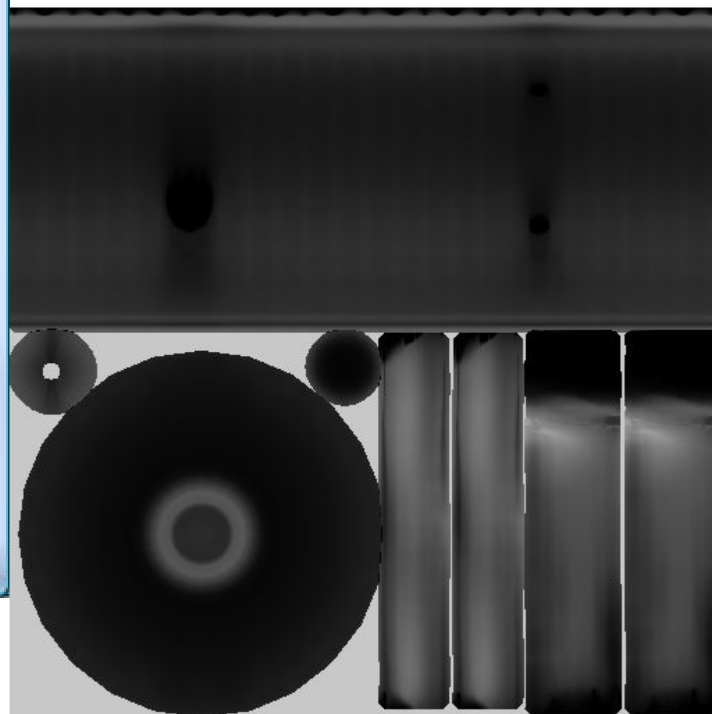


逆向き Ambient Occlusion

Partial Sorting

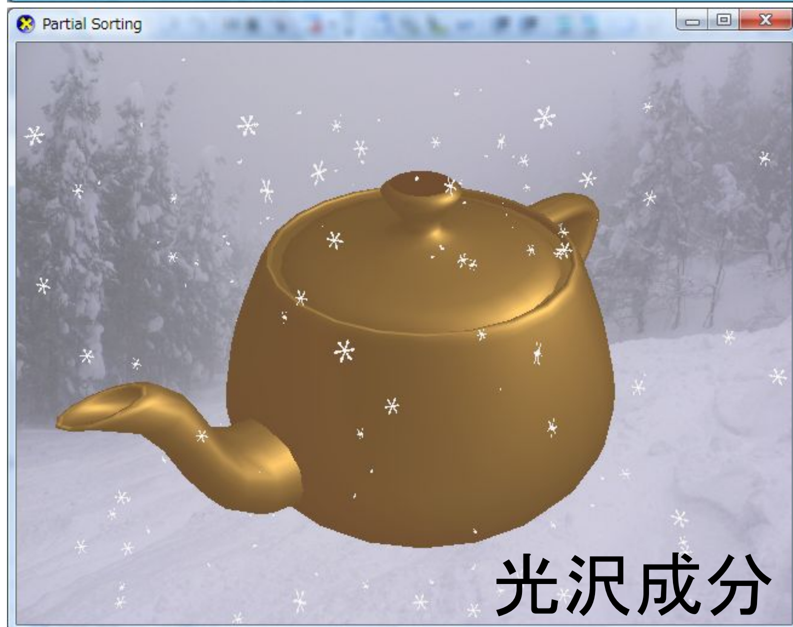


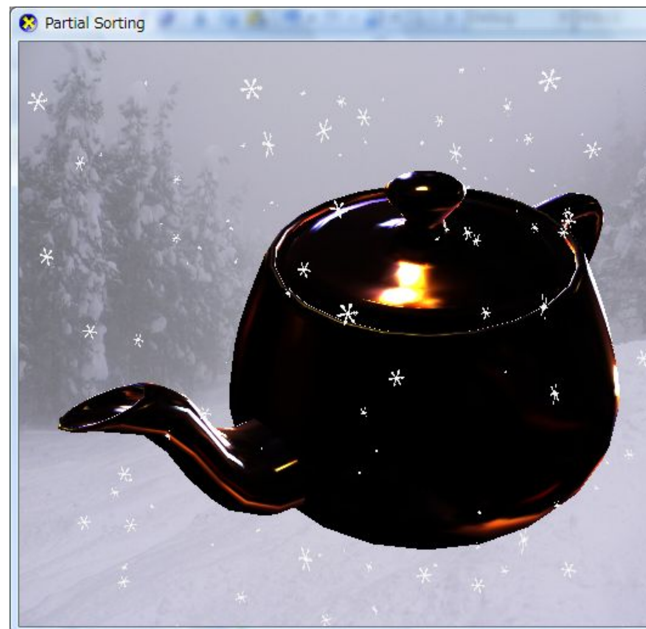
法線と反対方向に
半球分サンプリング



軽いエッジ、曲率検出

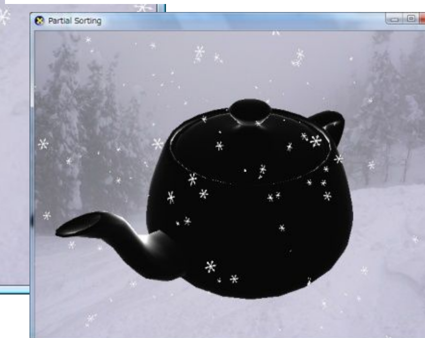
ライティング





環境マップ

元のテクスチャ



フレネル項

スムーズな合成

- なめらかにつなぎ合わせる

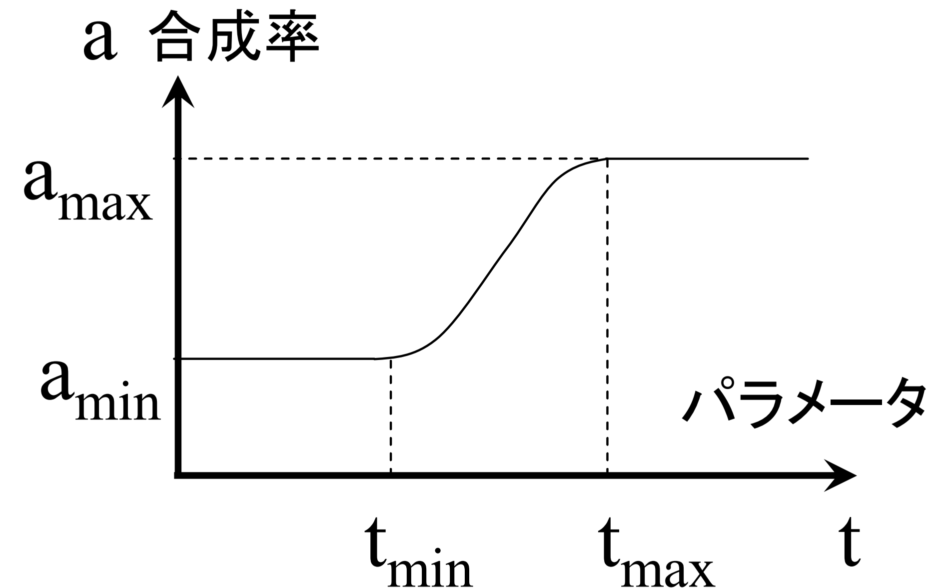
$$a = \begin{cases} a_{\min} & (t \le t_{\min}) \\ (a_{\max} - a_{\min})\, s\left(\dfrac{t - t_{\min}}{t_{\max} - t_{\min}}\right) + a_{\min} & (t_{\min} < t \le t_{\max}) \\ a_{\max} & (t_{\max} < t) \end{cases}$$

$$s(\xi) = 3\xi^2 - 2\xi^3$$

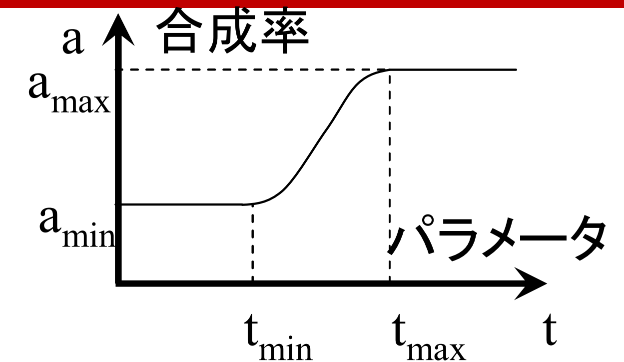
$$s(0) = 0, \quad s(1) = 1,$$

$$s'(0) = 0, \quad s'(1) = 0,$$

$$s'(\xi) = 6(1 - \xi)\xi$$



ソースコード



合成関数の作成

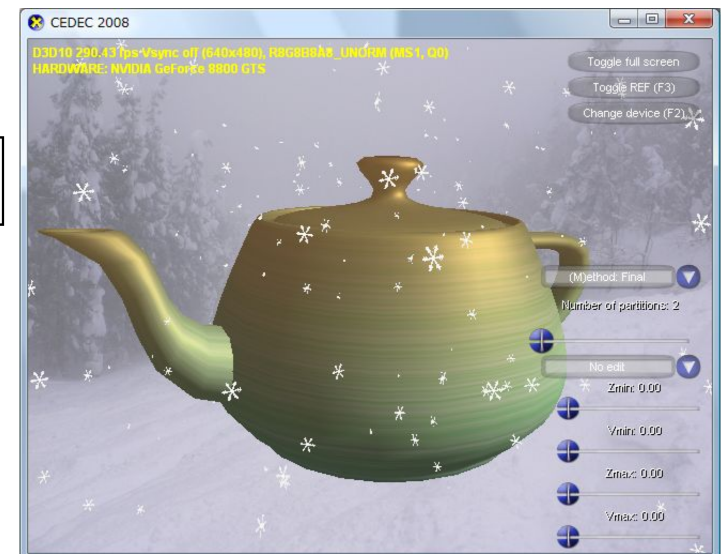
```
float smooth_blend(float t, float t_min, float t_max, float a_min, float a_max)
{
    float xi = (t- t_min)/(t_max - t_min);
    float s = (3-2*xi)*xi*xi;
    return (xi <=0) ? a_min : ((1.0< xi) ? a_max : lerp(a_min, a_max, s));
}
```

合成関数の呼び出し

```
output = lerp(拡散色, 光沢色, smooth_blend(y, 0,0,1,1));
```

拡散色

光沢色



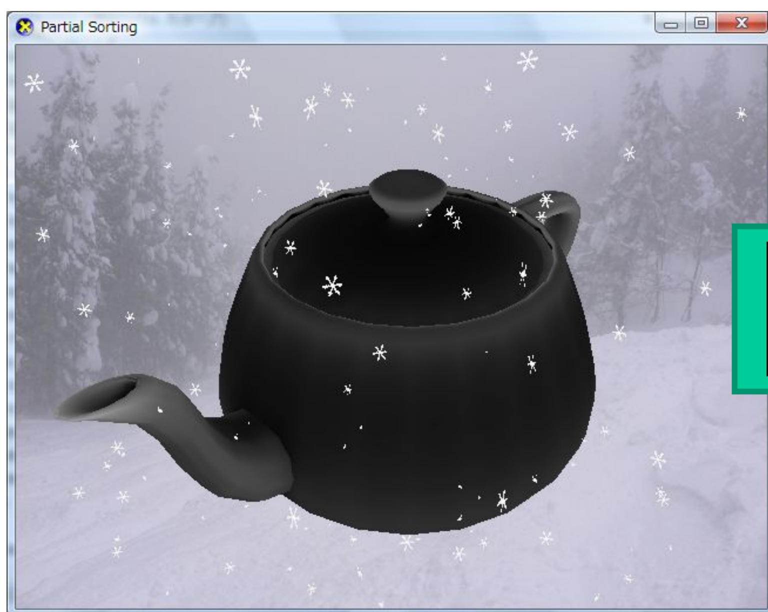
$a(y)$拡散色$+(1-a(y))$光沢色

経年劣化

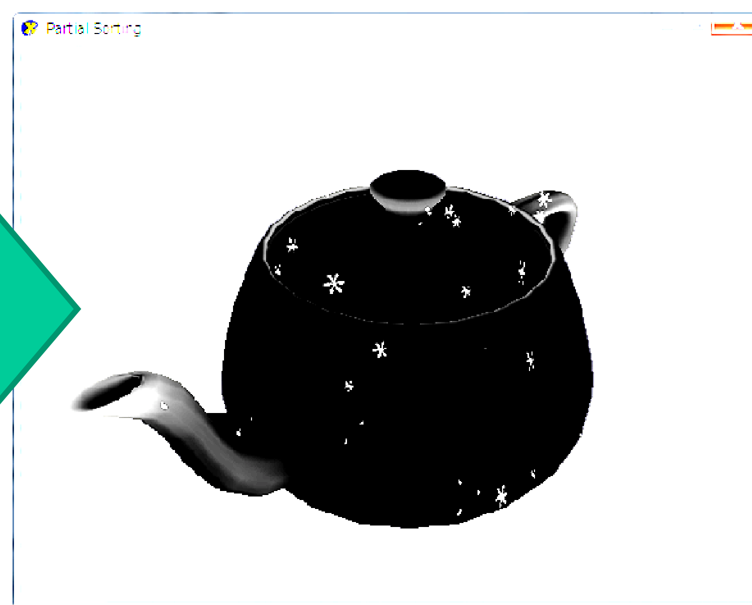
- 使い込まれたものは
  - 傷がついてる
  - 埃がつもっている

傷

- 付きやすい場所は？
  - とがったところ
    - ということは、エッジの情報を使えば、簡単に設定できるかも？



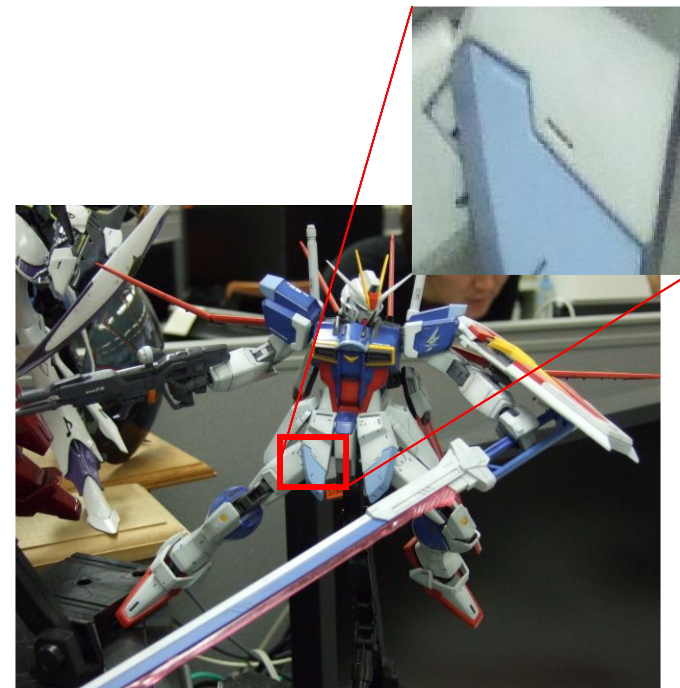
逆向きAO

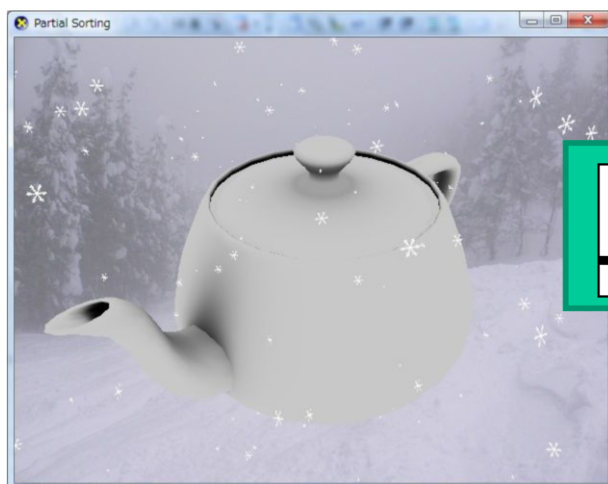


トーンカーブの補正

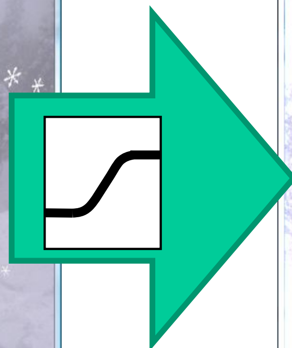
ほこり

- 積もりやすいところは？
  - 上から積もる
    - 法線情報が使えるかも
  - あと、部屋の隅にも埃が積もる
    - ガンプラの墨入れって何を書いていると思います？



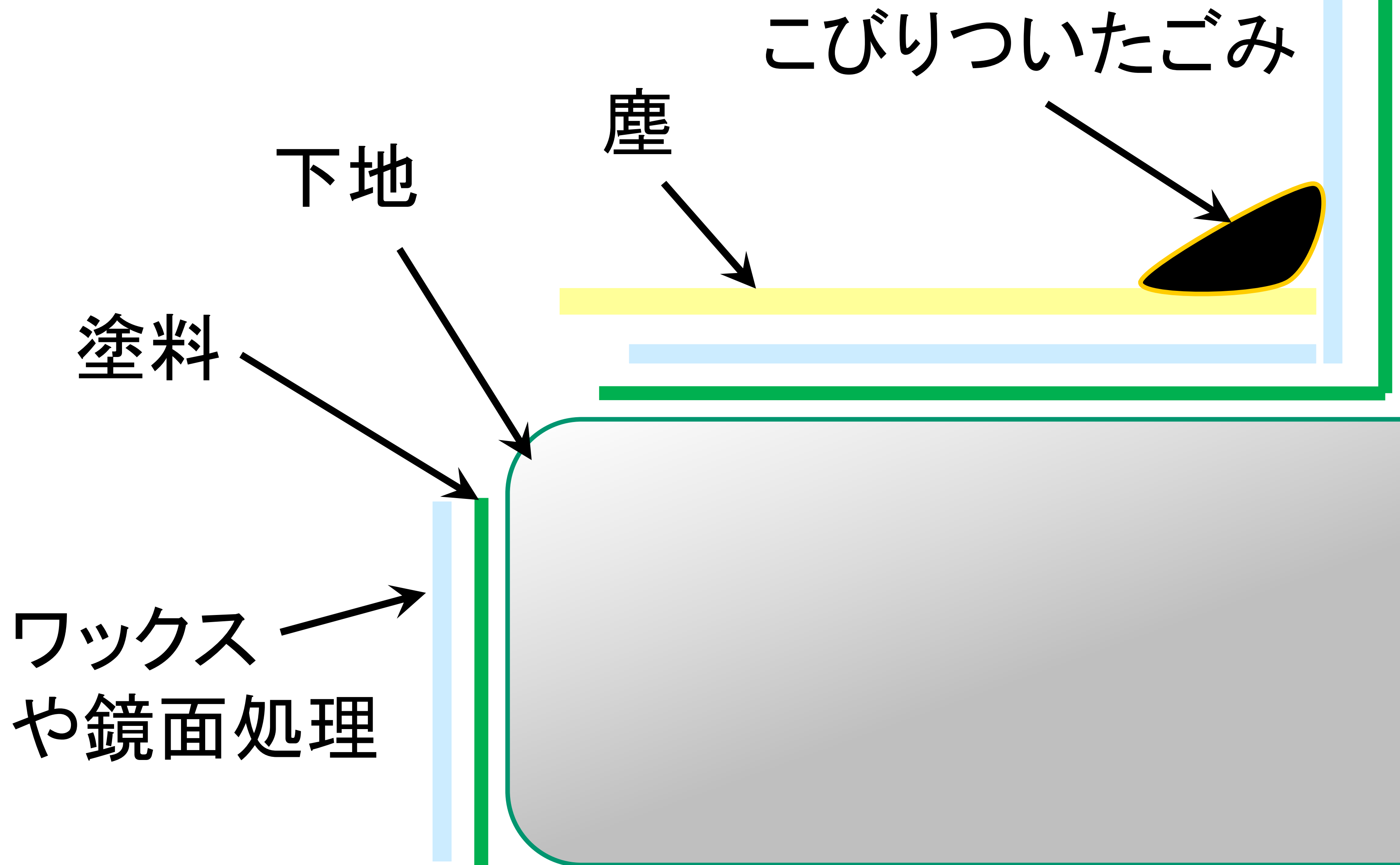


AO

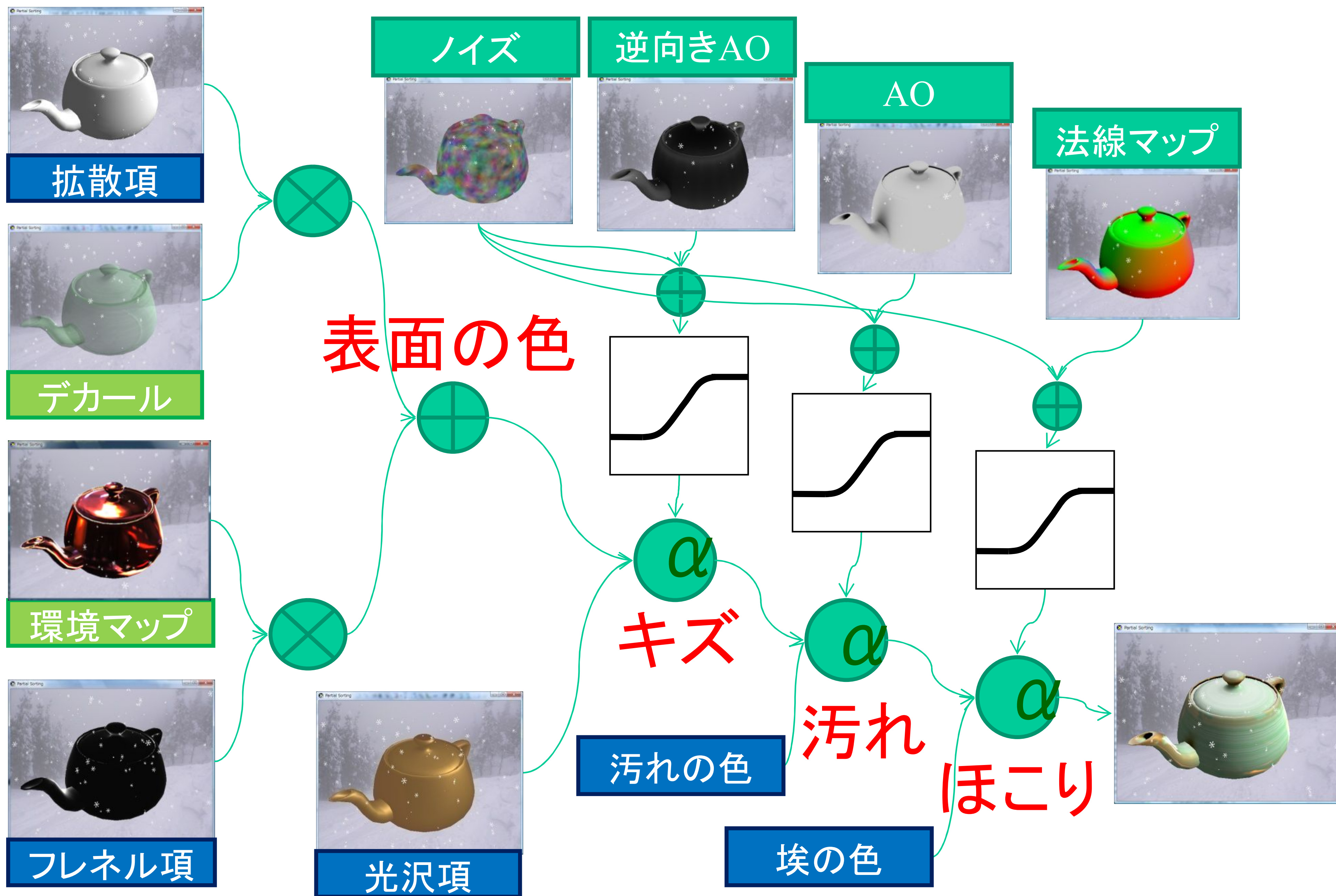


トーンカーブの補正

まとめると

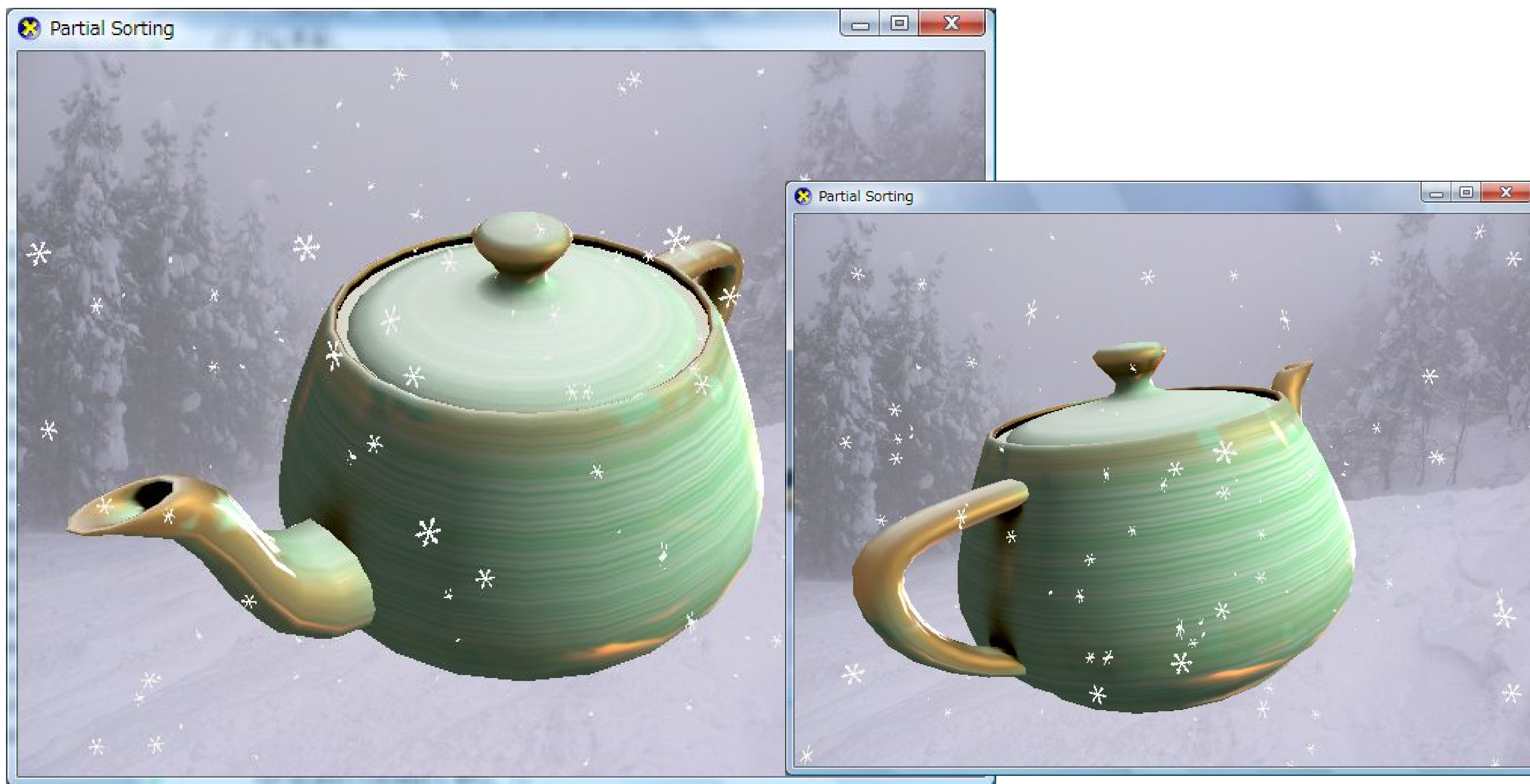


シェーディングネットワーク



結果

- はげて埃をかぶったティーポット



まとめ

- 自動的に作成した情報を使って、楽に作ろう。
- ノイズを加えりゃ何とかなるよ
- 他にも情報はあるかも
  - PRT(高次元の遮蔽情報)
    - 色をつけて簡易カラーブリーディング
  - 血流的な表面の流れ
  - 動的な情報
    - 速度
    - 光源の向き
  - さらに手で修正

ムケテ、未来。

CEDEC 2008

CESA DEVELOPERS CONFERENCE 2008

FOR NEXT 10 YEARS

移流テクスチャ

退屈？

- そんな話は知ってますか？
  - Perlin Noise は、Tron (1982)のころの技術でしょ?
- もうちょっと最近のお話

今年の、SIGGRAPH ネタ

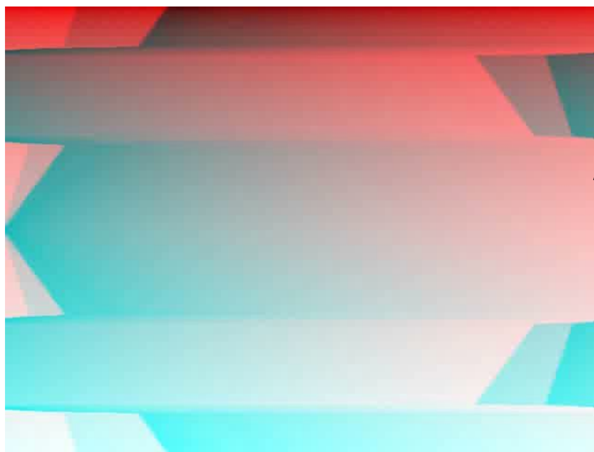
- Theodore Kim, Nils Thurey, Doug James, Markus Gross, "Wavelet Turbulence for Fluid Simulation", ACM-SIGGRAPH, 2008.
  - 流体を細かく見せる方法
    - Procedural の場所じゃないですが…
  - こるもごろふ理論による、分散関係（空間周波数とエネルギーの関係）を利用し、それらしいフラクタルノイズの組み合わせでリアルに見せる

重いので…

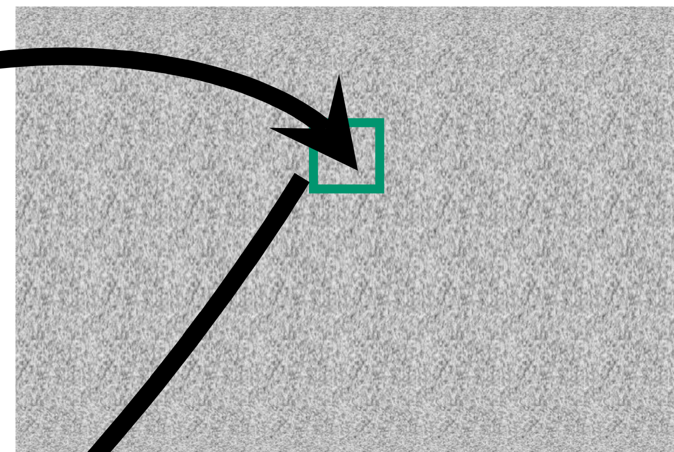
- すぐに使えるもの
- Fabrice Neyret, “Advected Textures”, ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA), 2003.
  - ポイント: 流れるノイズを作る

概要

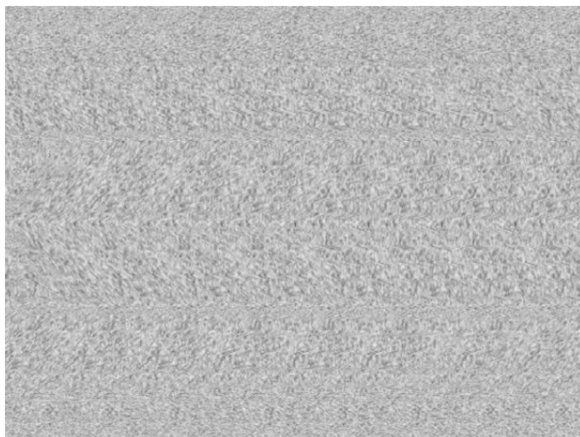
- ノイズを流す = テクスチャ座標を流す



テクスチャ座標アニメーション



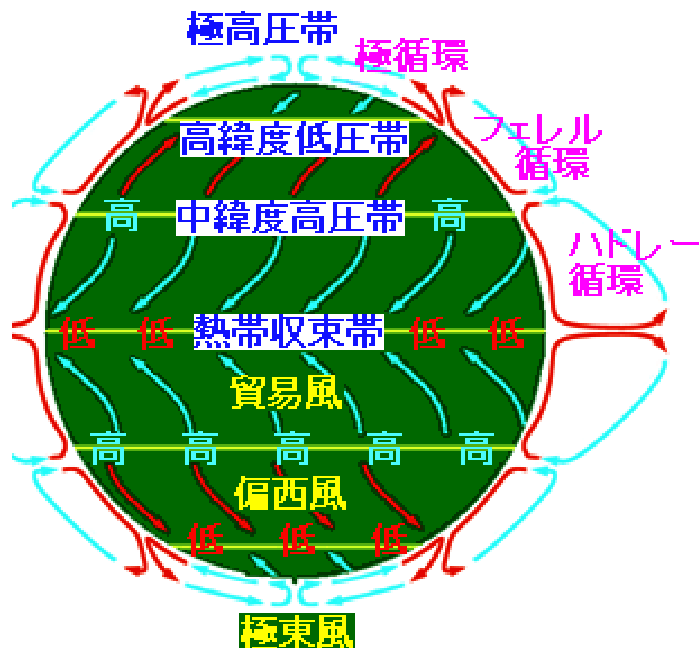
ノイズテクスチャ



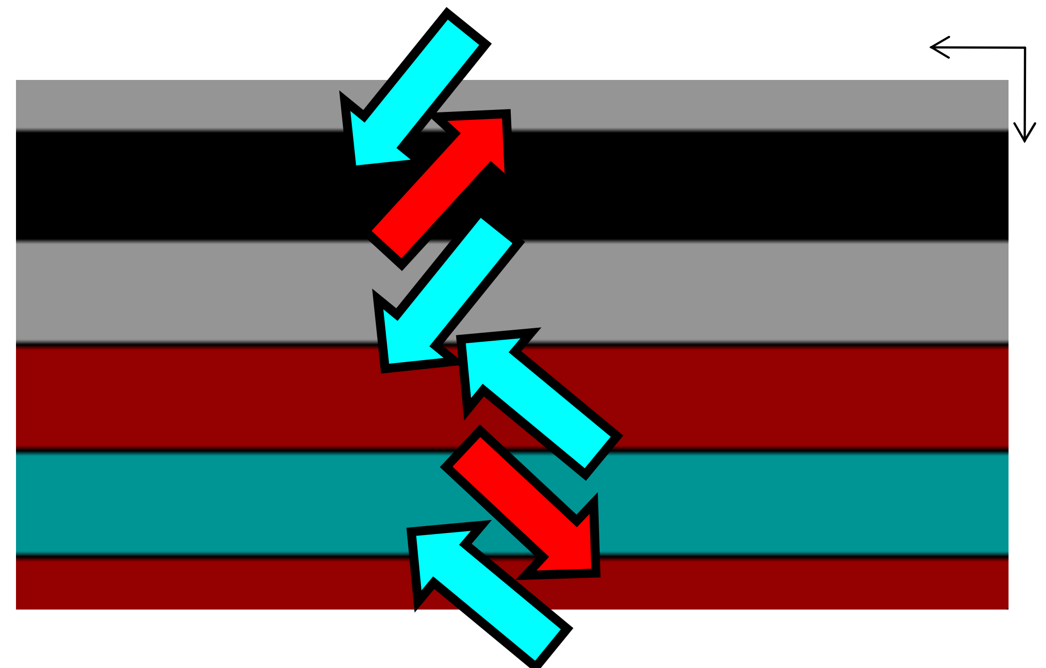
アニメーションしたテクスチャ

ちなみに…

- 今回の流し方
  - 一様な速度場
  - 流体計算を使うとよくなると思います
    - 台風は発生しない



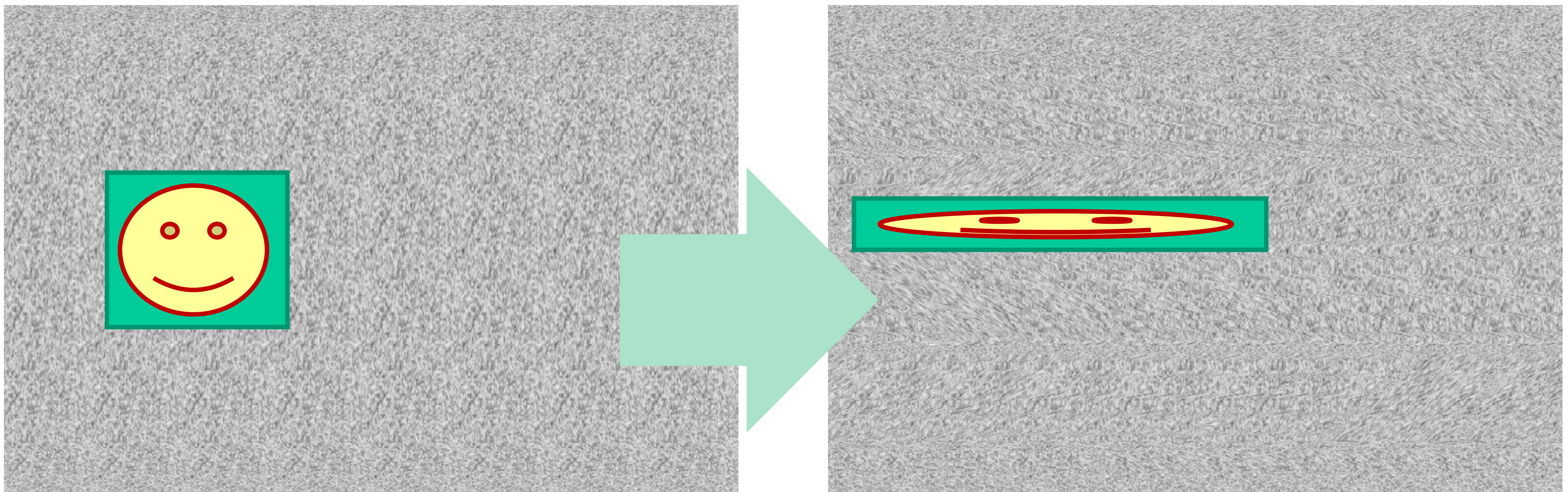
地球の大気循環と偏西風のモデル



速度場(一定…) [読み込んだ後、2s-1]

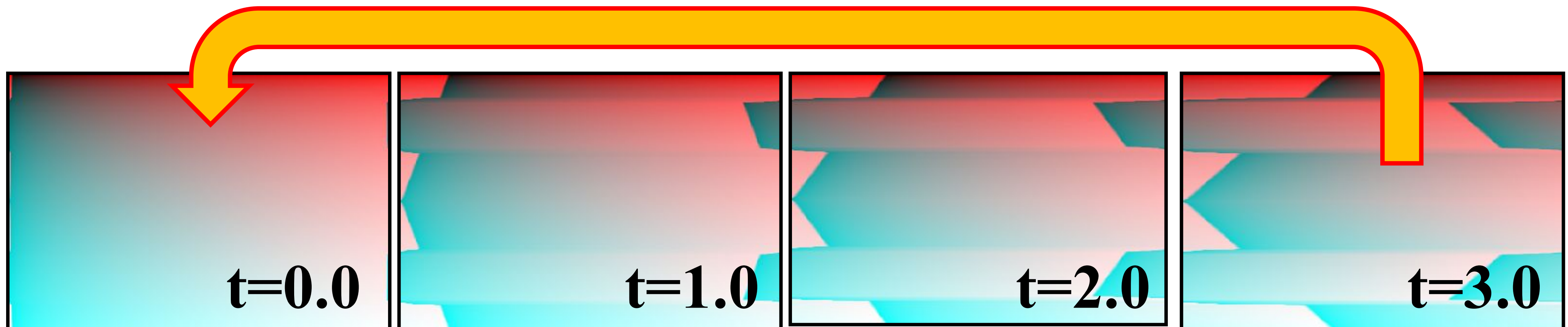
引き伸ばしの問題点

- ノイズをそのまま流すと
  - すごく引き伸ばされる
  - 押しつぶされる

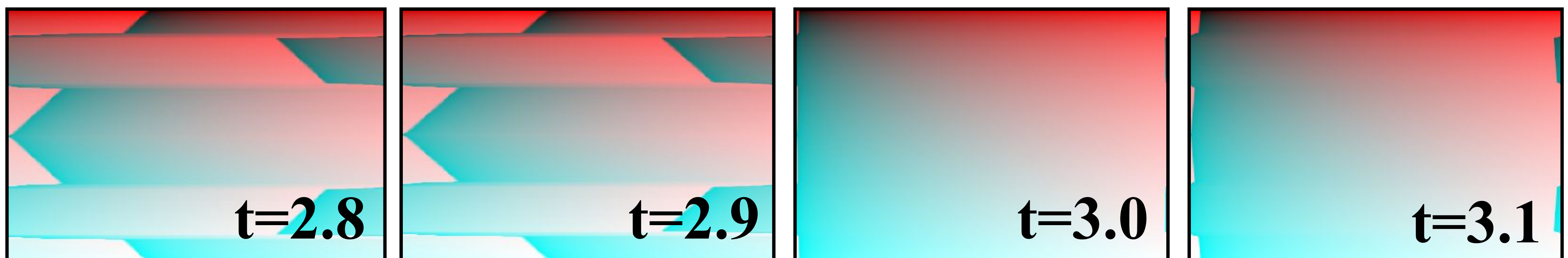


引き伸ばしの解消 (1/2)

- 引き伸ばされるのを防ぐには?
  - 適当なタイミングでリセット

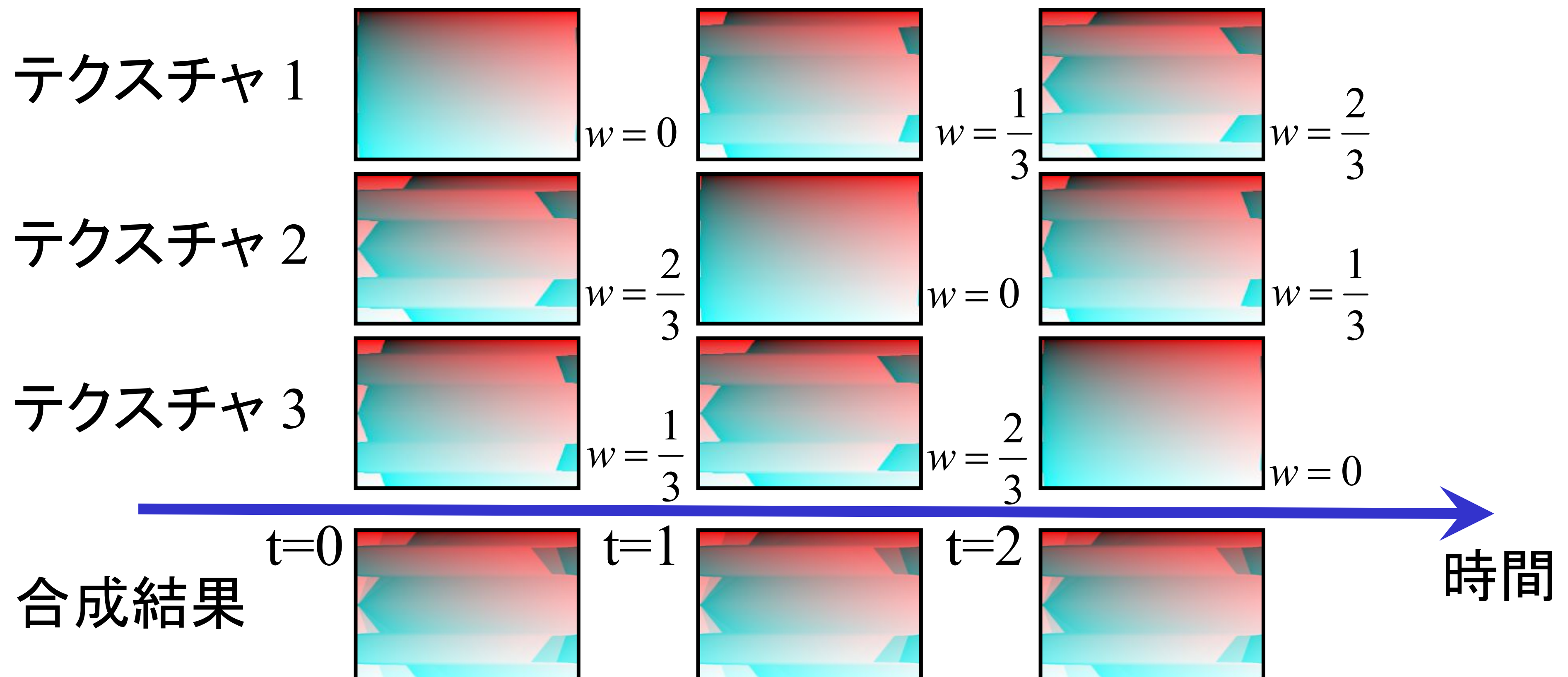


- リセットすると、切り替わりがバレる

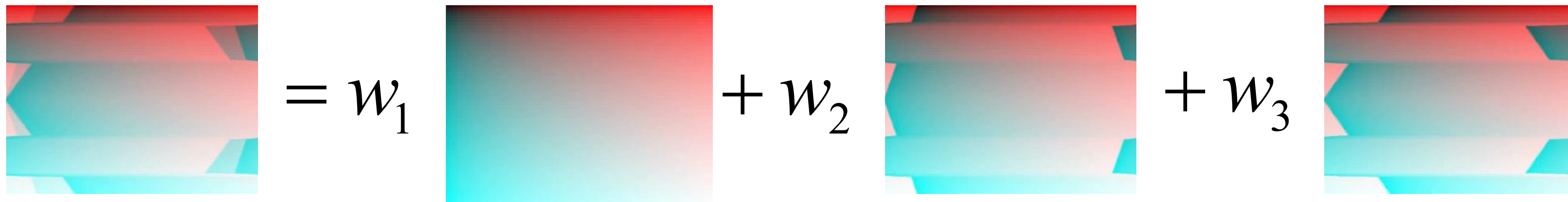


引き伸ばしの解消 (2/2)

- 切り替わりがばれるのなら
  - いくつかのテクスチャを用意して合成
  - リセットする瞬間に重みを0



重み

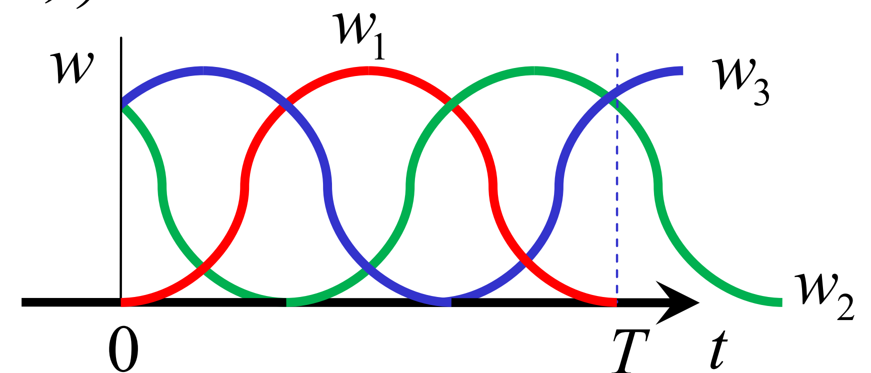


$= w_1 \quad + w_2 \quad + w_3$

$$w_1 = \frac{1}{3}\left(1 - \cos\left(2\pi \frac{t - \theta_1}{T}\right)\right), \qquad \theta_1 = 0$$

$$w_2 = \frac{1}{3}\left(1 - \cos\left(2\pi \frac{t - \theta_2}{T}\right)\right), \qquad \theta_2 = T/3$$

$$w_3 = \frac{1}{3}\left(1 - \cos\left(2\pi \frac{t - \theta_3}{T}\right)\right), \qquad \theta_3 = 2T/3$$

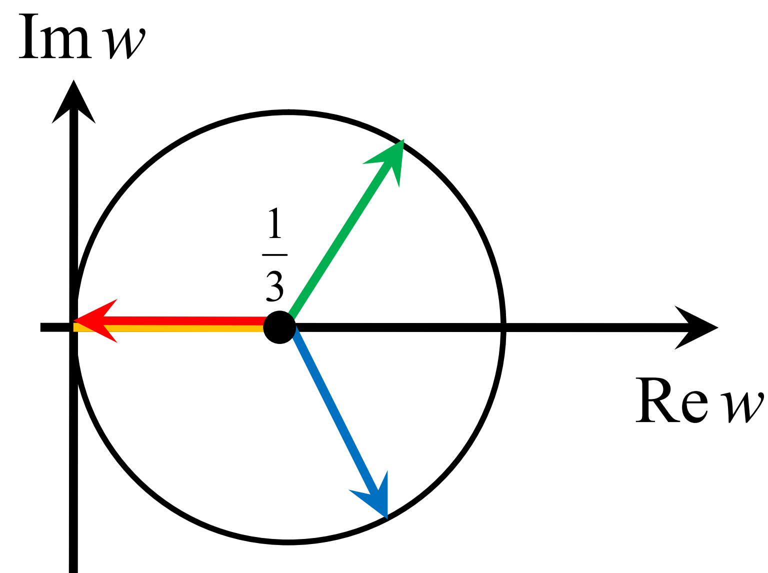


重みの意味

- 複素平面で考えると

$$w_n = \frac{1}{3}\left(1-\cos\left(2\pi\frac{t-\theta_n}{T}\right)\right) = \mathrm{Re}\,\frac{1}{3}\left(1-\exp\left(2\pi i\frac{t-\theta_n}{T}\right)\right)$$

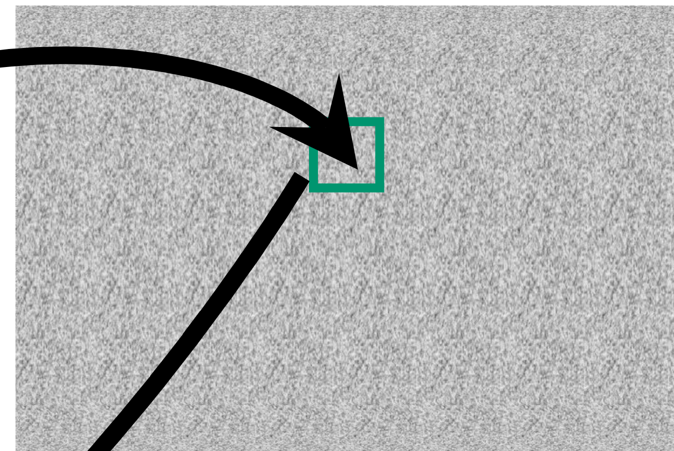
  - ぐるぐる回転
  - 矢印が左向きの時にリセット



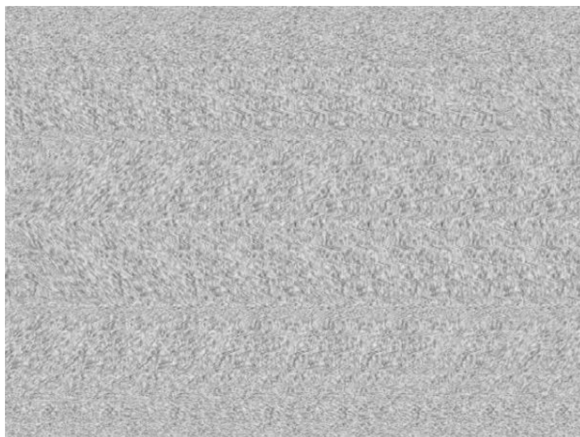
アニメーションノイズ

- ノイズを流す = テクスチャ座標を流す

テクスチャ座標アニメーション



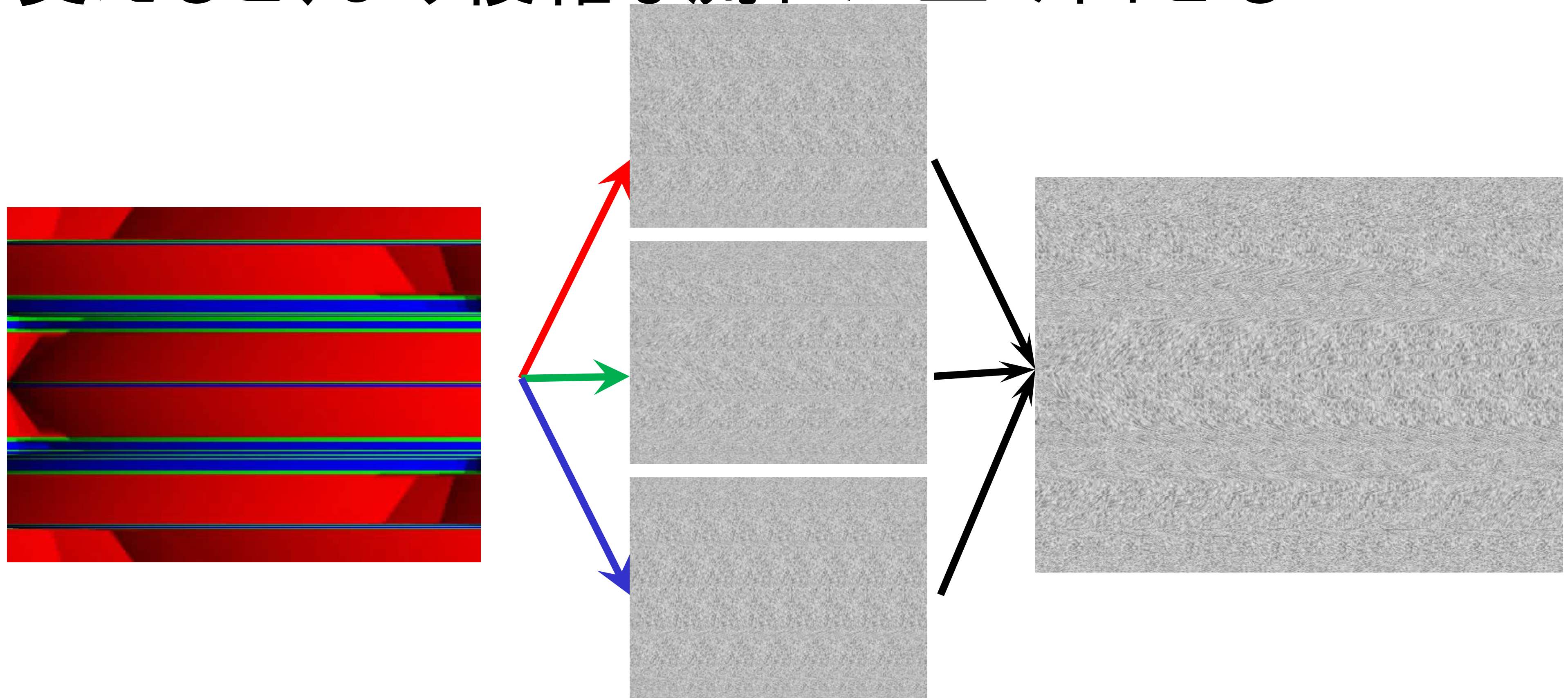
ノイズテクスチャ



アニメーションしたテクスチャ

複数のノイズの合成

- さらに、そのような流れるノイズを複数用意し、場所ごとの速度を見て、切り替える速度を変えると、より複雑な流れが生み出せる



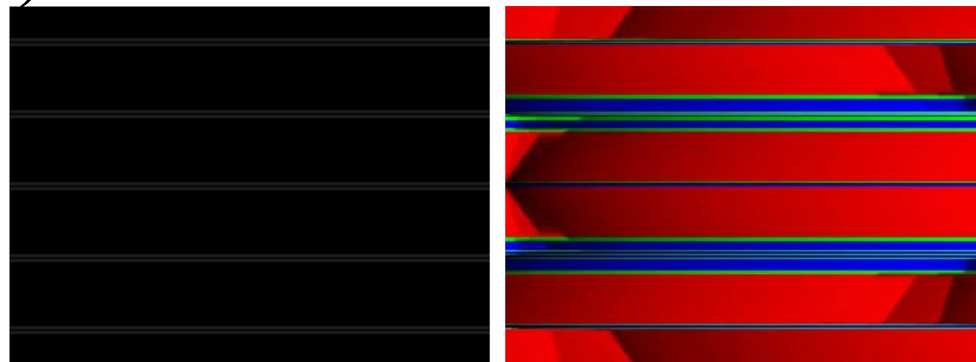
複数のテクスチャの合成率

- ひずみテンソル：連続力学における変形量

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)$$

- 局所変形量の蓄積

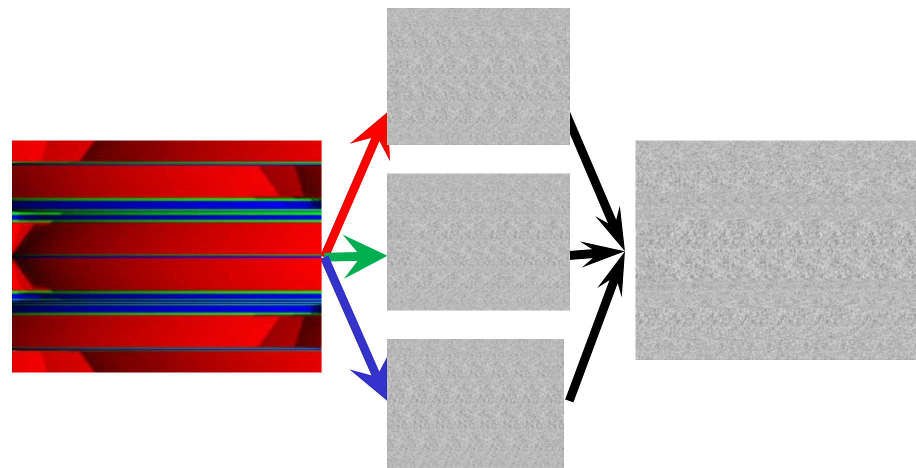
$$d(x,y) += \sqrt{\sum_{i,j} \varepsilon_{ij}^2(x,y)}\,dt$$



```
// 速度場のサンプリング
float2 v  = GetVelocity(In.TextureUV);
float2 vx = GetVelocity(In.TextureUV + float2(lerp(1/textureWidth, 1, 2*abs(In.TextureUV.y-0.5)), 0)) -
v;
float2 vy = GetVelocity(In.TextureUV + float2(0, 1/textureHeight)) - v;
// エネルギーの計算
float e = sqrt(vx.x*vx.x + vy.y*vy.y + 0.5*(vx.y+vy.x)*(vx.y+vy.x));
val += e * dt;
```

テクスチャ合成のソースコード

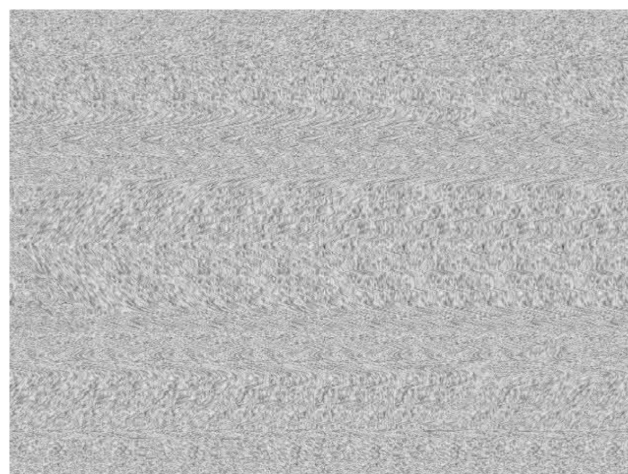
```
float t0 =      ;
float t1 =      ;
float t2 =      ;
float   l =  frac(      );
float noise;
if(l < 1.0/3.0)
{
        noise = lerp(t0, t1, l * 3);
}else if(l < 2.0/3.0){
        noise = lerp(t1, t2, l * 3 - 1.0);
}else{
        noise = lerp(t2, t0, l * 3 - 2.0);
}
```



最終合成

- 適当に濃いところを発生し、雲っぽく

雲の濃度



流れたノイズ

ノイズのかかった雲

結果

雲なし

雲あり

まとめ

- ノイズが動くと楽しいかも

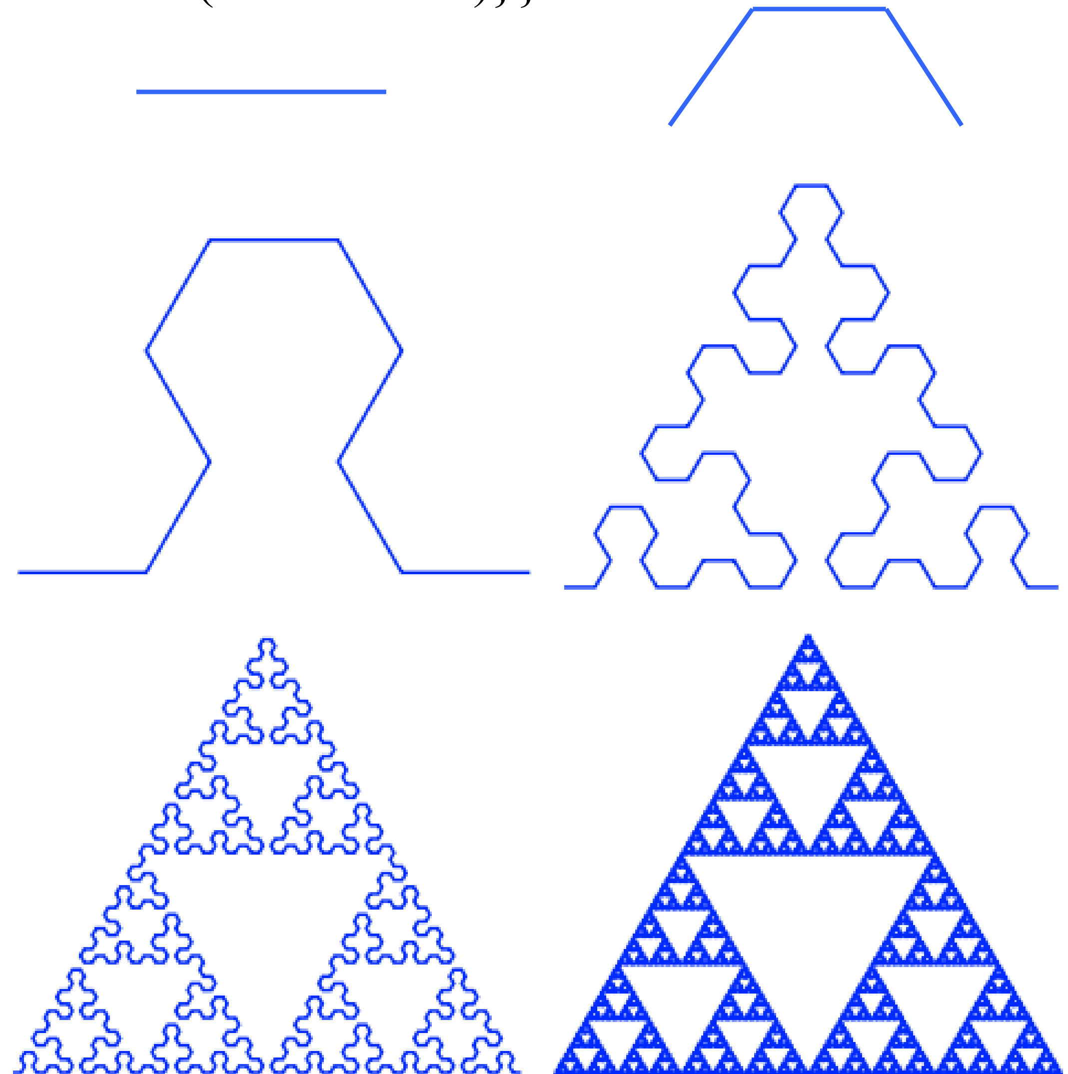
樹木生成

樹木生成

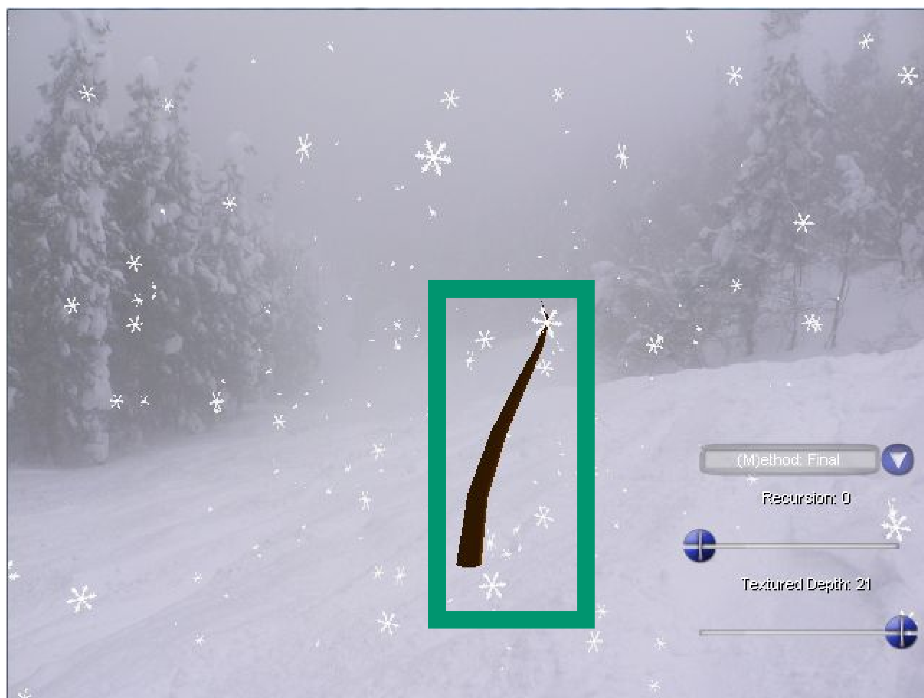
- L-System の有名な例

シェルピンスキーのギャスケット

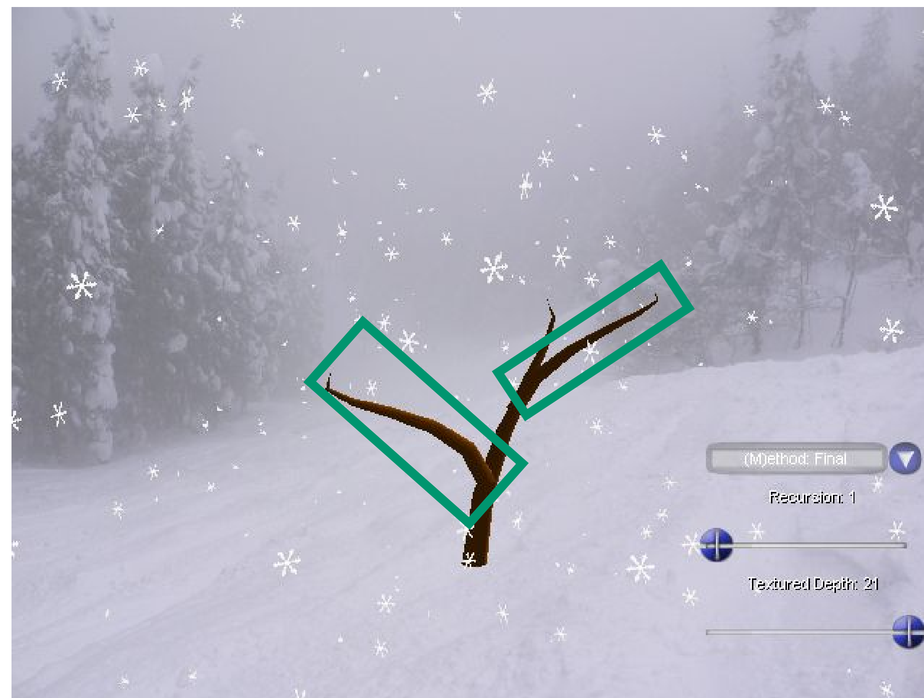
G = {{A, B}, {+, -}, A, {(A→B-A-B), (B→A+B+A)}}



木の作り方



大元の素材を用意



縮小したものを合成

再帰的に繰り返す

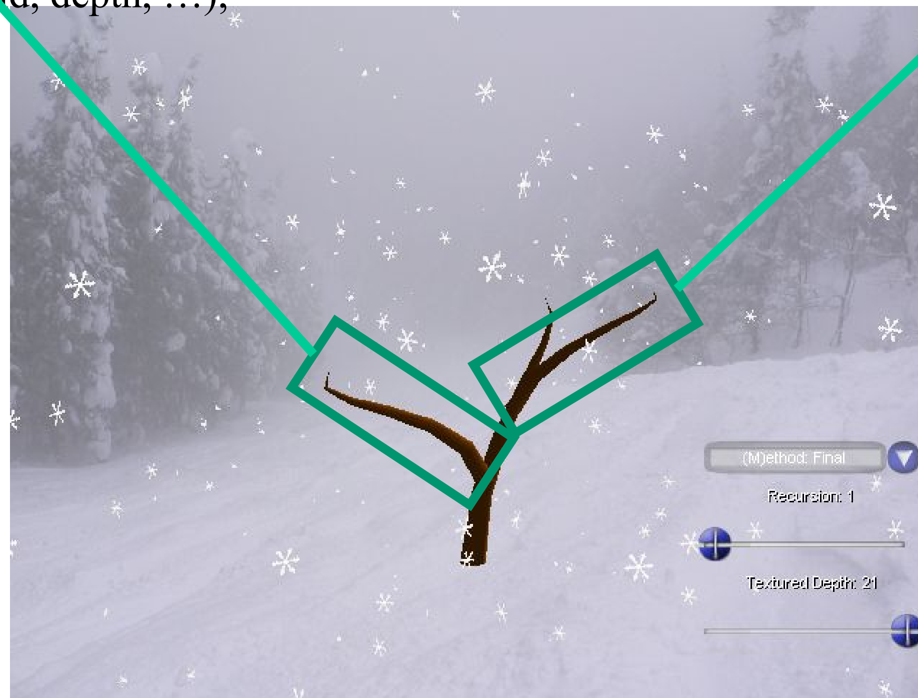
具体的な配置

縮小率　軸のひねり　そり具合　伸びる方向　張り付く位置

```
D3DXMatrixScaling(&mS, 0.8f, 0.8f, 0.8f);
D3DXMatrixRotationY(&mY1, D3DX_PI);
D3DXMatrixRotationX(&mX, -D3DX_PI*0.2f);
D3DXMatrixRotationY(&mY2, D3DX_PI*0.1f);
GetTreePosition(&x, 0.2f);// 下から2/10の位置を調べる
D3DXMatrixTranslation(&mT, x.x, x.y, x.z);
mChild = mS * mY1 * mX * mY2 * mT * mParent;
DrawTree(pd3dDevice, mChild, depth, …);
```

```
D3DXMatrixScaling(&mS, 0.7f, 0.7f, 0.7f);
D3DXMatrixRotationY(&mY1, D3DX_PI*0.2f);
D3DXMatrixRotationX(&mX, -D3DX_PI*0.2f);
D3DXMatrixRotationY(&mY2, D3DX_PI*0.2f);
GetTreePosition(&x, 0.4f);
D3DXMatrixTranslation(&mT, x.x, x.y, x.z);
mChild = mS * mY1 * mX * mY2 * mT * mParent;
DrawTree(pd3dDevice, mChild, depth , …);
```

親の行列



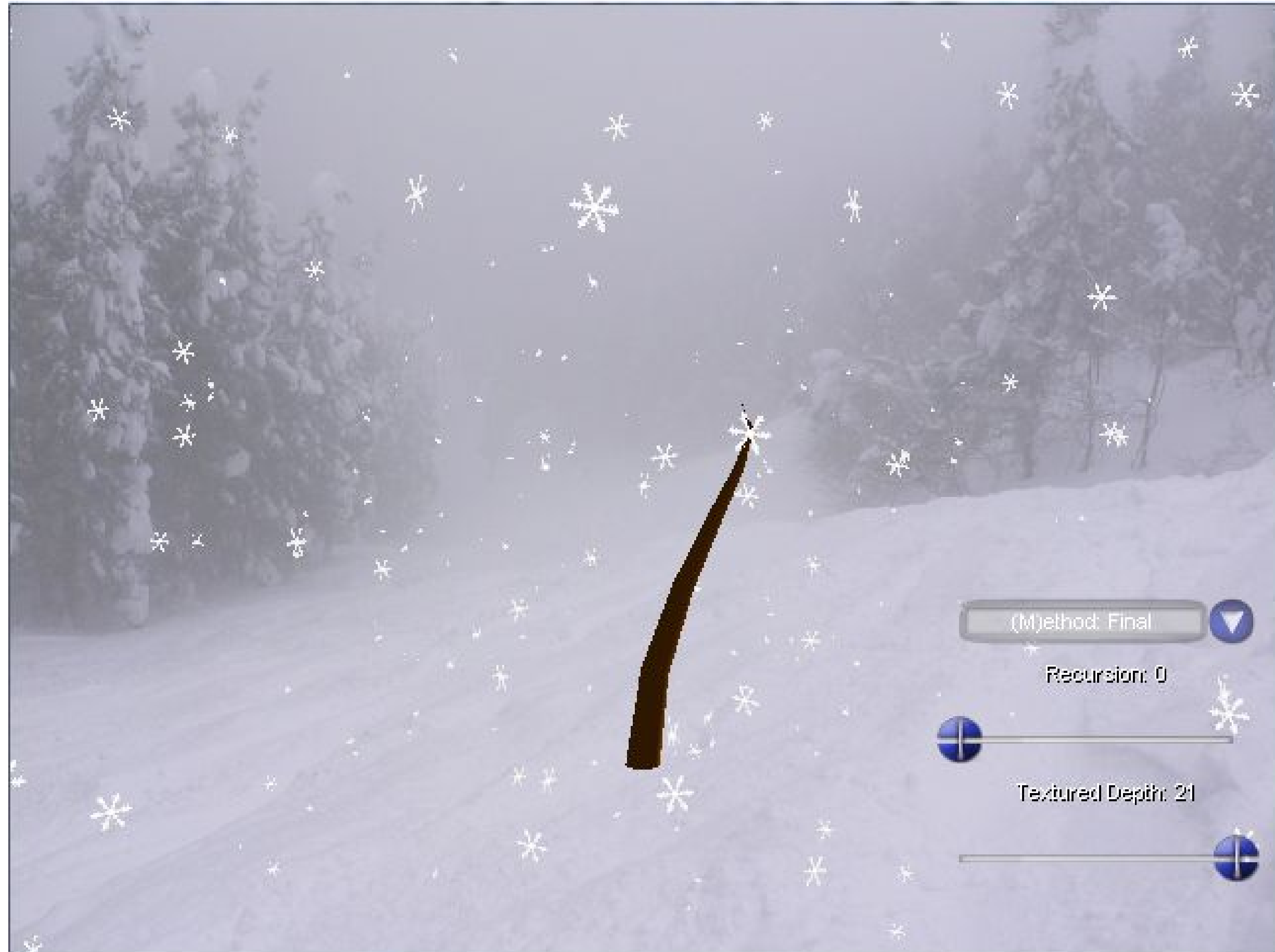
時間があったら
ちゃんとした
UI使おうね…

再帰コード

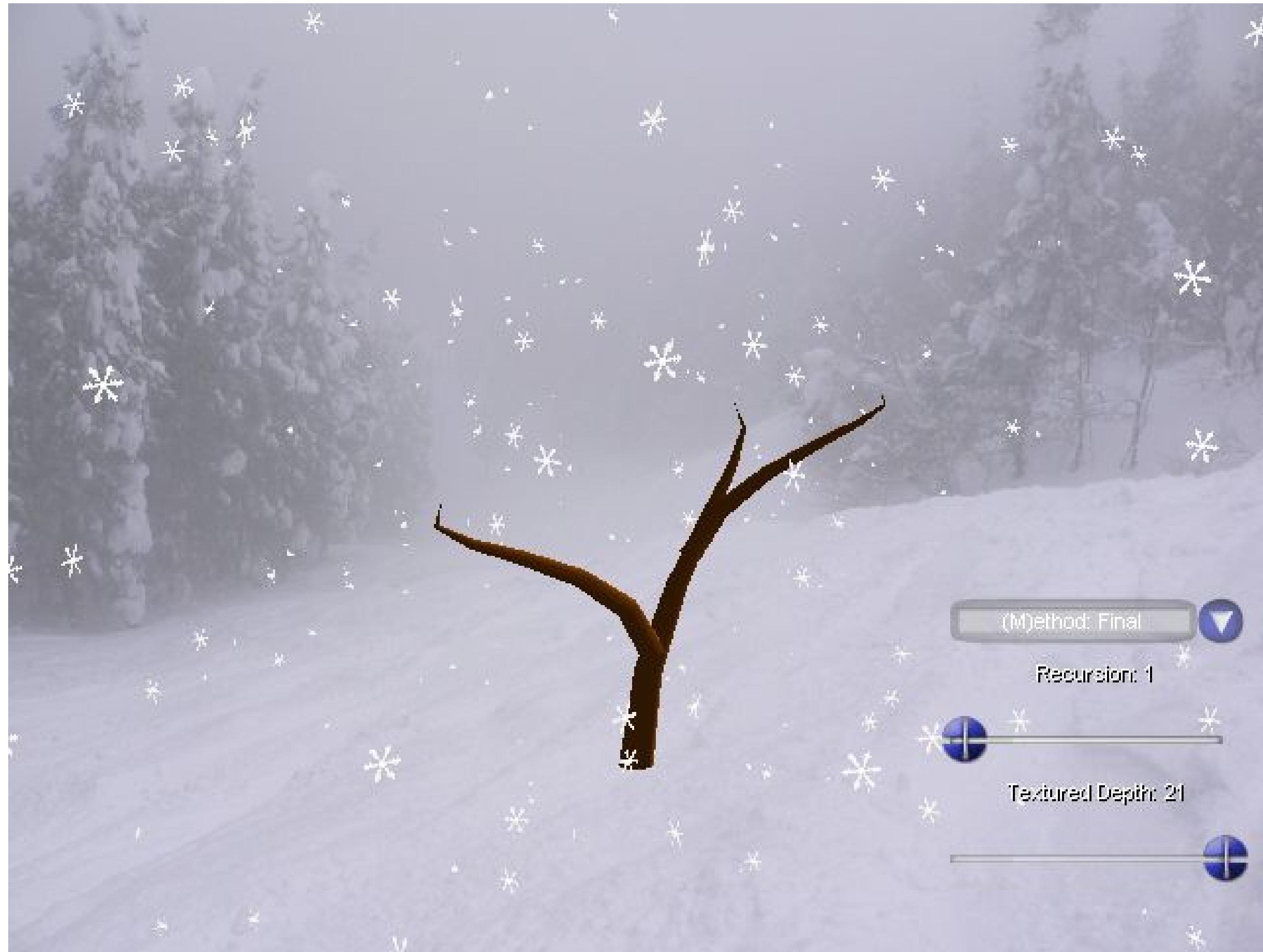
```
void DrawTree(ID3D10Device* pd3dDevice, const D3DXMATRIX &mParent, unsigned int depth,
unsigned int max_depth, ID3D10EffectTechnique* pTech)
{
        g_pmWorld->SetMatrix( (float*)&mParent );
        RenderTreeMesh(pd3dDevice, pTech);

        if(++depth <= max_depth) {
                // 十分細かく描画した
                D3DXMATRIX mChild;
                …
                mChild = mS * mY1 * mX * mY2 * mT * mParent;
                DrawTree(pd3dDevice, mChild, depth, max_depth, pTech);


                …
                mChild = mS * mY1 * mX * mY2 * mT * mParent;
                DrawTree(pd3dDevice, mChild, depth, max_depth, pTech);
        }
}
```

```
DrawTree(pd3dDevice, mIdentitiy, 0, max_depth, pTech)
```

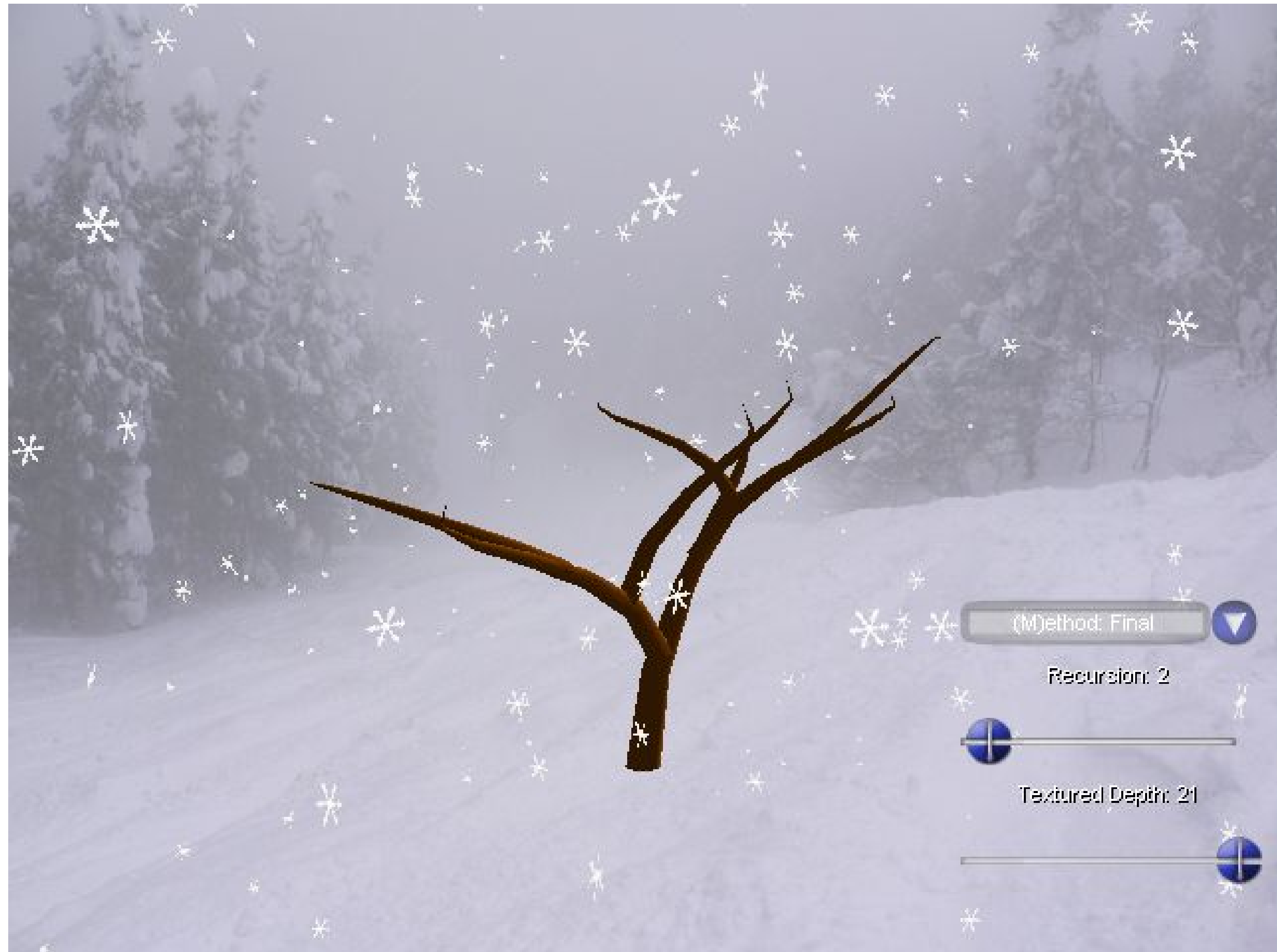
再帰0回



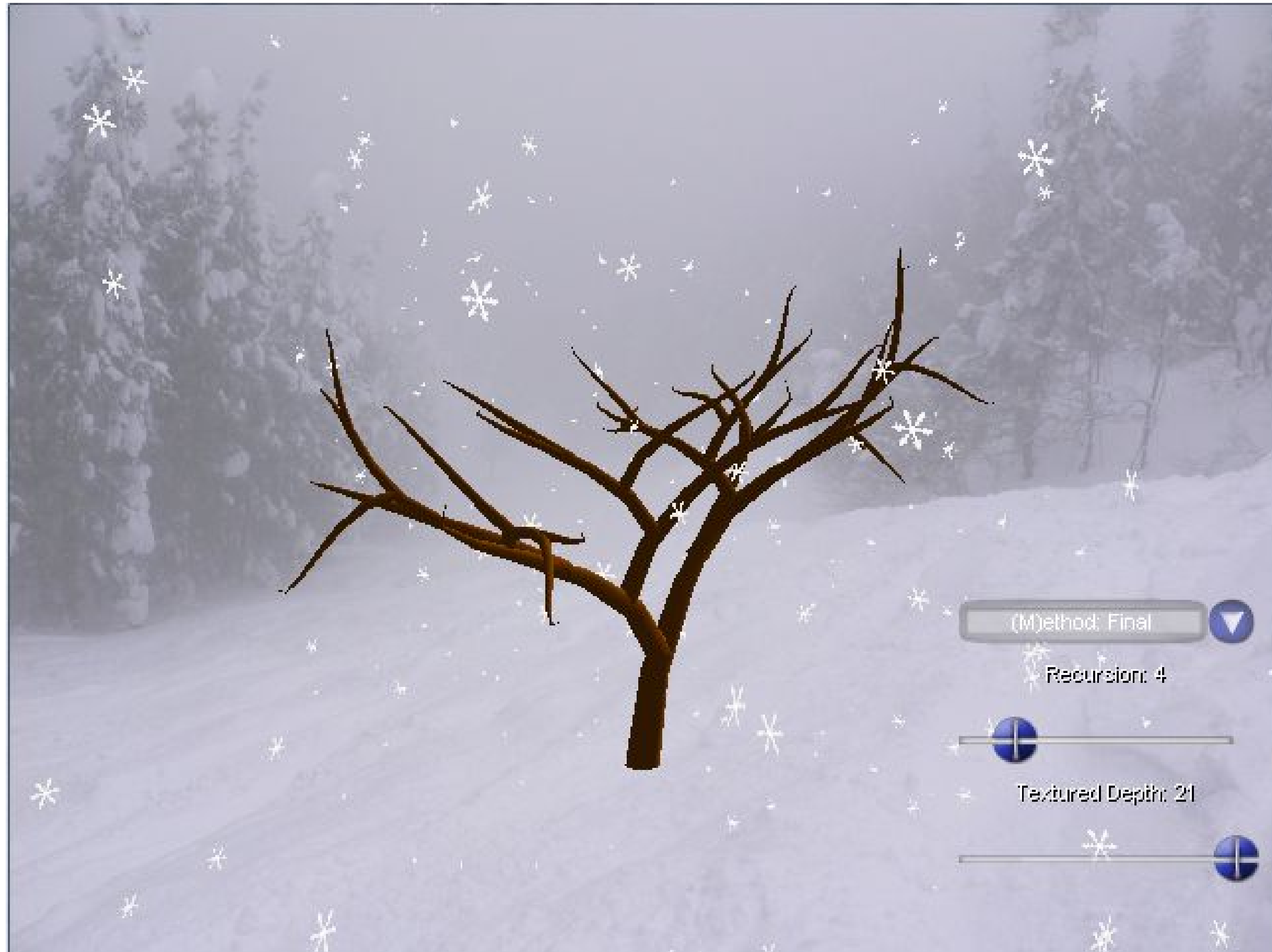
再帰1回



再帰2回



再帰4回

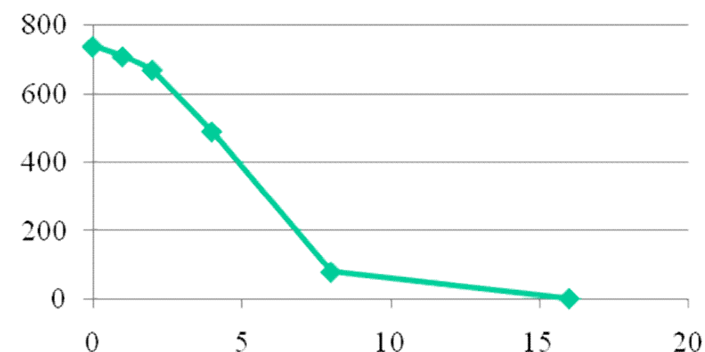


再帰8回

再帰16回

実行速度

- Core 2 Extreme X6800 (2.93GHz)
- GeForce 8800 GTX

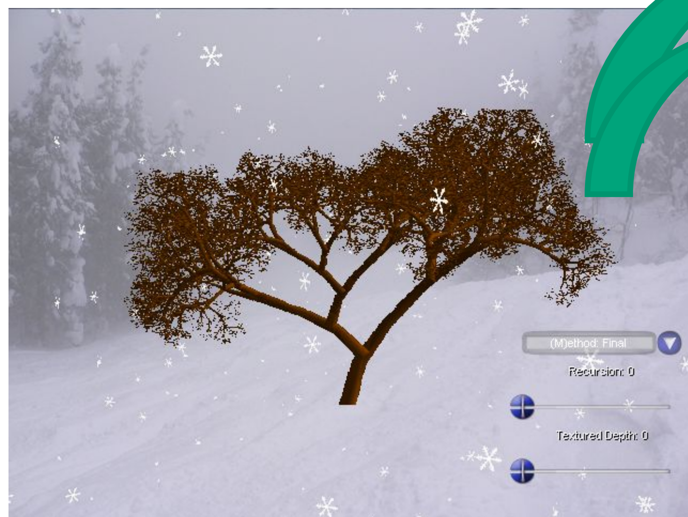


再帰回数	描画命令呼び出し回数	実行速度 (FPS)
0	1	740
1	3	710
2	7	670
4	31	490
8	511	78
16	131072	0.3

$$2^{n+1} - 1$$

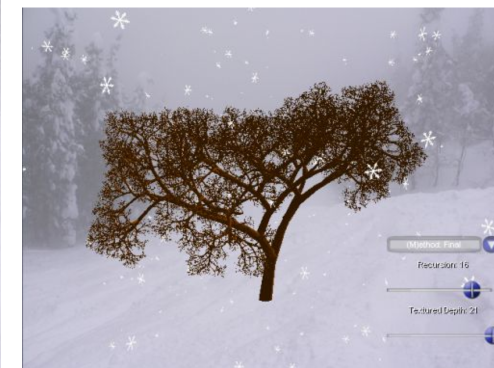
高速化したい

- 描画する数が多くなると処理が重くなる
- 分岐した部分は、フラクタル構造
- ビルボードを作成して、まとめて描画!



ビルボード(一枚絵)作成

1つの枝と2枚の板



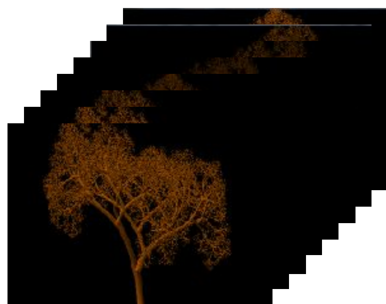
まじめな絵

再帰コードの変更

```
void DrawTree(ID3D10Device* pd3dDevice, const D3DXMATRIX &mParent, unsigned int depth,
unsigned int max_depth, unsigned int tex_depth, ID3D10EffectTechnique* pTech)
{
        if(tex_depth  < depth){
                RenderTreeSprite(pd3dDevice);
        }else{
                g_pmWorld->SetMatrix( (float*)&mParent );
                RenderTreeMesh(pd3dDevice, pTech);

                if(++depth <= max_depth) {
                        // 十分細かく描画した
                        D3DXMATRIX mChild;
                        …
                        mChild = mS * mY1 * mX * mY2 * mT * mParent;
                        DrawTree(pd3dDevice, mChild, depth, max_depth, pTech);


                        …
                        mChild = mS * mY1 * mX * mY2 * mT * mParent;
                        DrawTree(pd3dDevice, mChild, depth, max_depth, pTech);
                }
        }
}
```

ビルボードの作り方

- たくさんの方向から
  レンダリング
  (今回は、8方向)
- 3Dテクスチャに格納
  (読むとき楽)



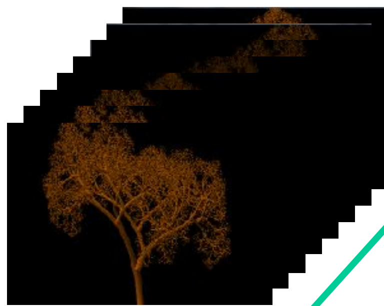
みんなは、法線マップもつくろうね

ビルボードの計算

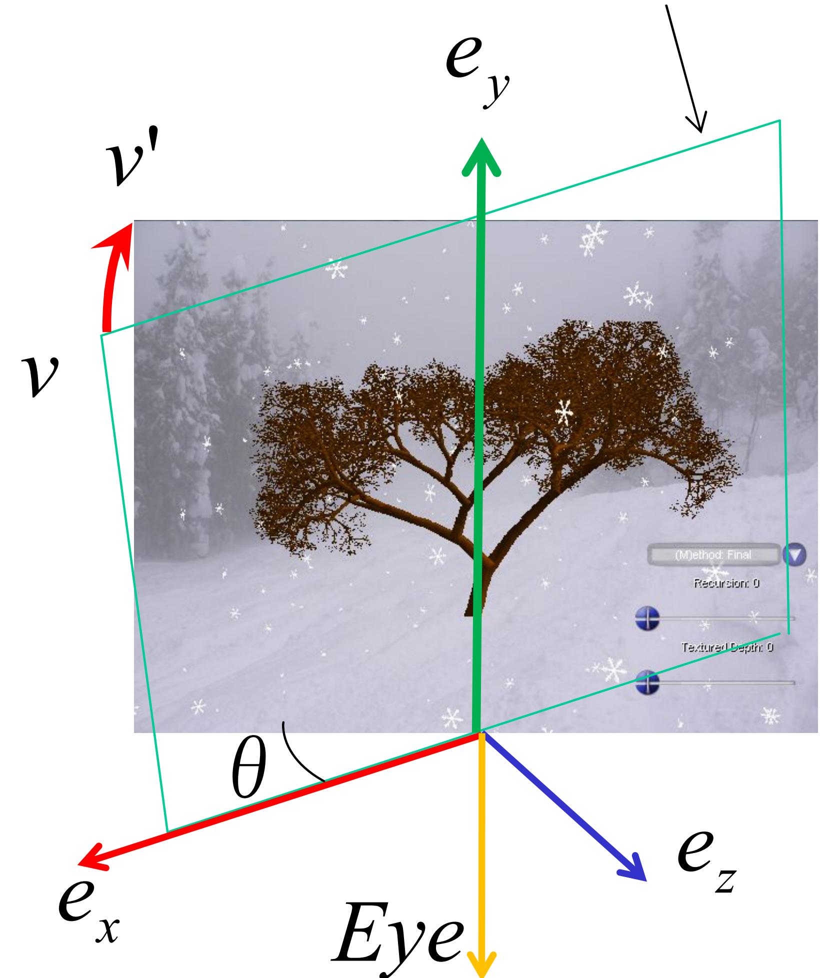
$$v' = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} v$$

$$= \begin{pmatrix} dot(Eye, e_z) & 0 & -dot(Eye, e_x) \\ 0 & 1 & 0 \\ dot(Eye, e_x) & 0 & dot(Eye, e_z) \end{pmatrix} v$$

$$\theta = \arccos(dot(Eye, e_z))$$

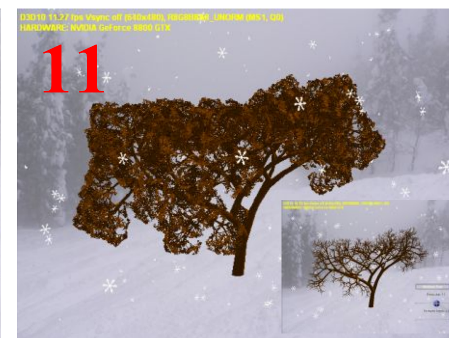


$$z = \frac{\theta}{2\pi}$$



結果

テクスチャを使用した階層	実行速度(FPS)	テクスチャ未使用時の速度
0	705	740
1	691	710
2	643	670
3	565	592
4	462	490
5	332	362
6	216	240
7	126	143
8	70	78
9	36	41
10	18	21
11	9.3	10



まとめ

- フラクタル構造を持っているなら、途中で計算済みデータに置き換えることによって、高品質で高速な処理が可能
- LOD処理の中にフラクタル構造がないか探してみよう

ありがとうございました

- 質問はございませんか？