

# WebGLによるデータ可視化入門\*

## 簡単なWebGLプログラム

陰山 聡

神戸大学 システム情報学研究科 計算科学専攻

2014.04.22

# 環境設定

- ログイン確認
- WebGL 環境オン
  - Safari 立ち上げ
  - 環境設定
  - メニューバーに開発メニューを表示 にチェック
  - 開発メニューから “WebGL を有効”
- 確認
  - この講義のウェブページ
  - <http://bit.ly/1qXgljB>
  - サンプルをクリック

# 前回の復習

# WebGL とは

WebGL = シェーダを使い、HTML5 の canvas に、JavaScript で 3D CG を書くための API

## WebGL の特徴

- スタンドアロンアプリからウェブアプリへの流れ
- クロスプラットフォーム
- オープンスタンダード
- Web で GPU を使ったレンダリングが可能
- 開発・利用が容易：プラグイン不要
- ソースコードが見える
- グラフィックス（OpenGL）と UI（ウィンドウ管理やイベント処理）の分離が明白

# 線形代数の復習

## 同次座標

同次座標 (homogeneous coordinates) とは 3 次元空間中の位置座標  $x$  と、任意のベクトル  $v$  をあえて 4 成分で表現したもの。

3 次元空間中の位置座標  $x$  を

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (1)$$

ベクトル  $v$  は

$$\begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} \quad (2)$$

# アフィン変換

3次元空間の位置座標  $x$  や、ベクトル  $v$  の変換を考える。

$$x \longrightarrow y \equiv F(x). \quad (3)$$

線形変換 = スケール変換 + 回転 + 剪断。

アフィン変換 = 線形変換 + 平行移動。

平行移動は 3 行 3 列の行列では書けない。

同次座標と 4 行 4 列の行列を使えば書ける。



## 線形代数の復習：内積

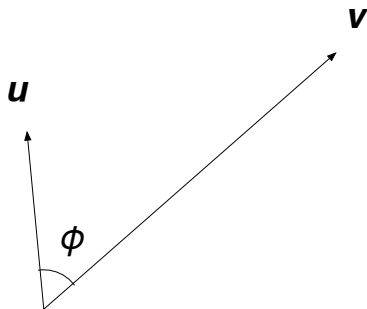
$n$ 次元空間中のベクトルと正方行列

ベクトル  $u$  の大きさ

$$u = |\mathbf{u}|$$

内積

$$\mathbf{u} \cdot \mathbf{v} = u_i v_j = u v \cos \phi$$



## 線形代数の復習：正規直交系

$$e_i \cdot e_j = \delta_{ij} \quad (\text{クロネッカーのデルタ})$$

一般のベクトル  $v$  と正規直交系  $\{e_0, e_1, \dots, e_{n-1}\}$

$v$  の  $i$  成分

$$v_i = v \cdot e_i$$

## 線形代数の復習：外積

## 3次元空間のベクトル

$$\mathbf{w} = \mathbf{u} \times \mathbf{v} \quad w_i = \epsilon_{ijk} u_j v_k \quad (\text{エディントンのイプシロン})$$

$\mathbf{w}$  は  $\mathbf{u}$  と  $\mathbf{v}$  の両方に垂直

$$w = u v \sin \phi$$

$$\mathbf{u} \times \mathbf{v} = -\mathbf{v} \times \mathbf{u}$$

$$\mathbf{u} \cdot (\mathbf{v} \times \mathbf{w}) = (\mathbf{u} \cdot \mathbf{w}) \mathbf{v} - (\mathbf{u} \cdot \mathbf{v}) \mathbf{w}$$

## 線形代数の復習：行列のかけ算

$M$  と  $N$  は行列

行列  $M$  の成分を  $M_{ij}$  ( $i, j = 0, 1, \dots, n-1$ )

行列  $N$  の成分を  $N_{ij}$  ( $i, j = 0, 1, \dots, n-1$ )

とすると

$$L = MN$$

の成分は

$$L_{ij} = \sum_{k=0}^{n-1} M_{ik}N_{kj} = M_{ik}N_{kj}$$

## 線形代数の復習：行列のかけ算

$$(LM)N = L(MN)$$

$$(L + M)N = LN + MN$$

$$MI = IM = M \quad (I : \text{単位行列})$$

一般には非可換：

$$MN \neq NM$$

## 線形代数の復習：逆行列

正方行列  $M$  に対して

$$MN = NM = I$$

という行列  $N$  を逆行列という。

逆行列を

$$M^{-1}$$

と書く。

一般には逆行列を求めるのは大変（計算量が多い）

glMatrix.js（後述）では4行4列の逆行列を求める関数が組み込まれている。

## 線形代数の復習：行列式

正方行列に対して

$$\det(\mathbf{M})$$

$$\det(\mathbf{I}) = 1$$

$$\det(\mathbf{MN}) = \det(\mathbf{M}) \det(\mathbf{N})$$

$$\det(\mathbf{M}^t) = \det(\mathbf{M})$$

## 線形代数の復習：行列の転置

行列  $M$  の成分を  $M_{ij}$  ( $i, j = 0, 1, \dots, n - 1$ )

転置行列を  $M^t$  と書く

$a$  を数、 $M$  と  $N$  を行列として

$$(aM)^t = aM^t$$

$$(M + N)^t = M^t + N^t$$

$$(M^t)^t = M$$

$$(MN)^t = N^t M^t$$



## 線形代数の復習：行列のトレース

$$\text{tr}(M) = \sum_{i=0}^{n-1} M_{ii}$$

## 線形代数の復習：直交行列

$$MM^t = M^tM = I$$

を満たす正方行列  $M$  を直交行列という。

$$M^t = M^{-1}$$

$$\det(M) = \pm 1$$

$M^t$  も直交行列。

直交行列はベクトルの長さを変えない：

$$|Mu| = |u|$$

直交する二つのベクトルを直交行列で変換しても直交したまま。

$$u \cdot v = 0 \iff (Mu) \cdot (Mv) = 0$$

## 3次元空間中の平面

点  $p$  を通り、ベクトル  $u$  とベクトル  $v$  で張られる平面の式：

$$\boldsymbol{x} = \boldsymbol{p} + s\boldsymbol{u} + t\boldsymbol{v}$$

単位ベクトル  $\boldsymbol{n} \equiv \boldsymbol{u} \times \boldsymbol{v} / |\boldsymbol{u} \times \boldsymbol{v}|$  を使えば、

$$\boldsymbol{n} \cdot \boldsymbol{x} + d = 0$$

$\boldsymbol{n}$  を法線ベクトルという。  $f(\boldsymbol{x}) = \boldsymbol{n} \cdot \boldsymbol{x} + d$  とすると

$f(\boldsymbol{x}_0) = 0 \iff$  点  $\boldsymbol{x}_0$  はこの平面の上

$f(\boldsymbol{x}_0) > 0 \iff$  点  $\boldsymbol{x}_0$  は  $\boldsymbol{p} + \boldsymbol{n}$  側にある

$f(\boldsymbol{x}_0) < 0 \iff$  点  $\boldsymbol{x}_0$  は  $\boldsymbol{p} - \boldsymbol{n}$  側にある

## 面積

3点  $p, q, r$  を頂点とする 3 角形の面積

$$S = \frac{1}{2} |(\mathbf{p} - \mathbf{r}) \times (\mathbf{q} - \mathbf{r})|$$

$x$ - $y$  平面上におかれた  $n$  角形の面積

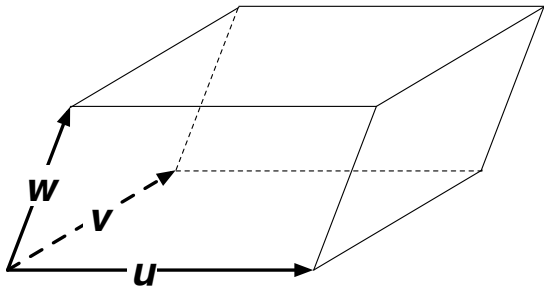
$$S = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - y_i x_{i+1}) = \frac{1}{2} \sum_{i=0}^{n-1} \{x_i (y_{i+1} - y_{i-1})\}$$

添字は  $\text{mod } (n)$  をとる。

## 体積

原点を基点とする3つのベクトル  $u, v, w$  が張る平行6面体の体積

$$V = \mathbf{u} \cdot (\mathbf{v} \times \mathbf{w}) = \mathbf{v} \cdot (\mathbf{w} \times \mathbf{u}) = \mathbf{w} \cdot (\mathbf{u} \times \mathbf{v})$$



## 同次座標

3次元  $\Rightarrow$  4次元

3次元空間の位置座標

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

3次元空間のベクトル

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \Rightarrow \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix}$$

## 同次座標

行列

$$M = \begin{pmatrix} M_{00} & M_{01} & M_{02} & 0 \\ M_{10} & M_{11} & M_{12} & 0 \\ M_{20} & M_{21} & M_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## 平行移動

平行移動行列

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (4)$$



## 回転

 $z$  軸の周りの回転

$$R_z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (5)$$

## スケール変換

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (6)$$

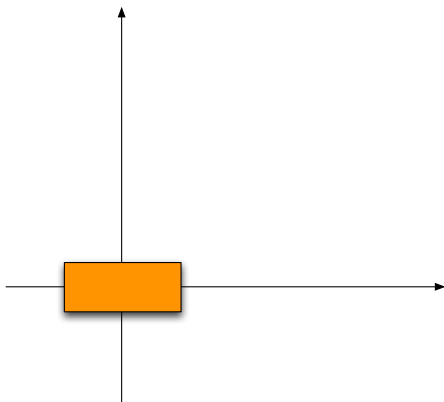
## 剪断

$$H_{xy}(\beta) = \begin{pmatrix} 1 & \beta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (7)$$

## 座標変換の合成

アフィン変換は非可換。一般に

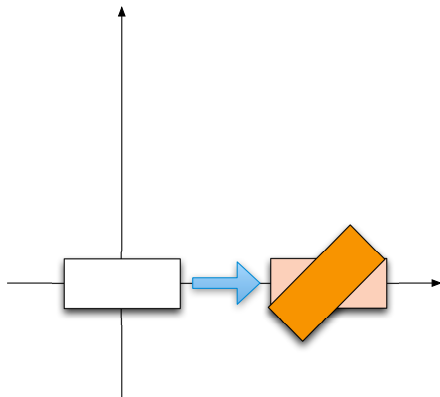
$$M_1M_2 \neq M_2M_1$$



## 座標変換の合成

アフィン変換は非可換。一般に

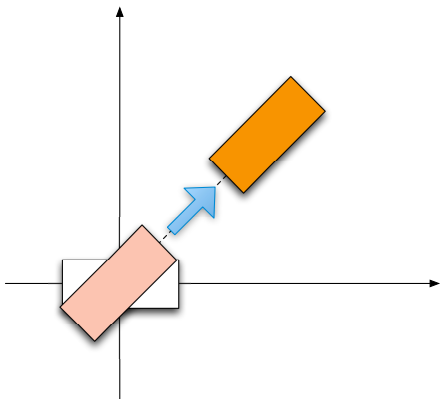
$$RT \neq TR$$



## 座標変換の合成

アフィン変換は非可換

$$RT \neq TR$$



## 演習：機能の確認

WebGL 公式ページ

<http://www.khronos.org/webgl/>

デモ集 [http://www.khronos.org/webgl/wiki/Demo\\_Repository](http://www.khronos.org/webgl/wiki/Demo_Repository)

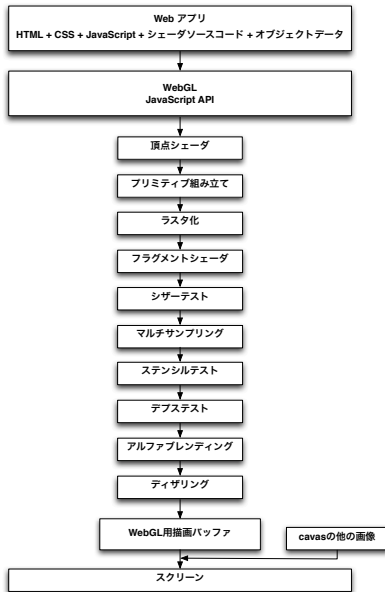
Safari でのソースコード表示方法：

- 開発メニュー
- ページのソースを表示

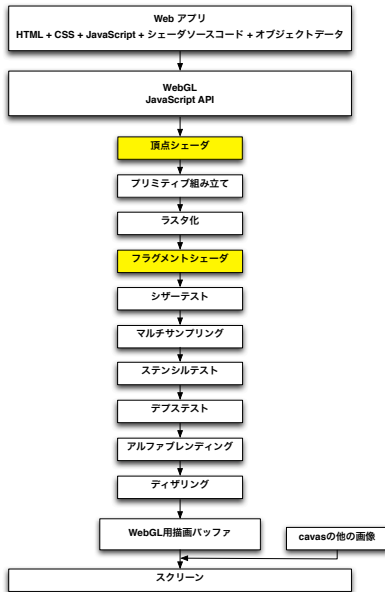
# WebGL グラフィックスパイプライン



# WebGL のグラフィックスパイプライン

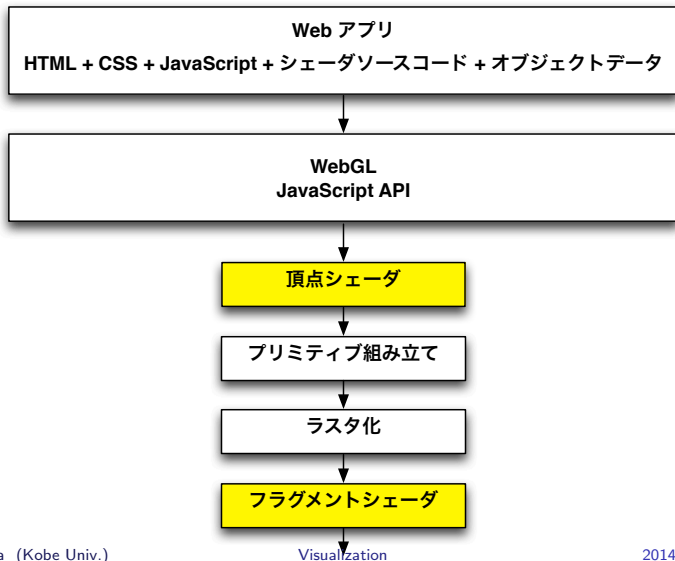


## シェーダ ( 拡大図は次のページ )



# シェーダ

## 頂点シェーダ（バーテックスシェーダ）とフラグメントシェーダ



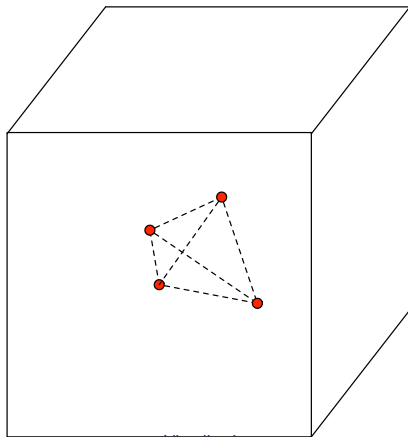
# WebGL アプリケーション

Web アプリ = HTML + CSS + JavaScript

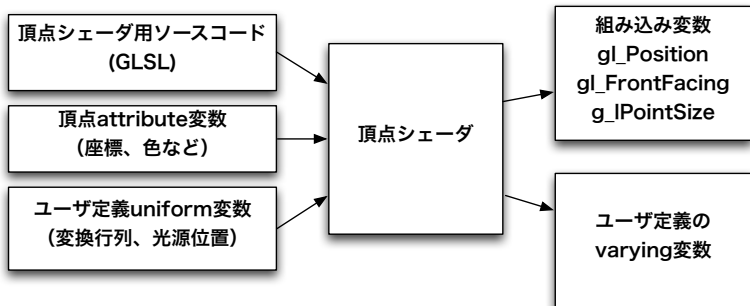
WebGL アプリ = HTML + CSS + JavaScript + シェーダ言語 ( OpenGL SL )

## 頂点シェーダ

- 各頂点に対して処理を行う
- 並列処理
- $n$  個の頂点があれば  $n$  個の頂点シェーダプロセッサを同時に実行させる



## 頂点シェーダの入出力データ



## 頂点シェーダプログラム

- C 言語に似ている。
- OpenGL SL (Shading Language)
- 4 行 4 列の行列ベクトル演算が組み込み関数

```
attribute vec3 aVertexPos;  
attribute vec4 aVertexColor;  
  
uniform mat4 uMVMatrix;  
uniform mat4 uPMatrix;  
  
varying vec4 vColor;  
  
void main() {  
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPos, 1.0);  
    vColor = aVertexColor;  
}
```

# 頂点シェーダプログラム

```
attribute vec3 aVertexPos;  
attribute vec4 aVertexColor;
```

attribute (属性) 変数とは

- ユーザが定義する変数
- 各頂点に固有のデータ (位置や色)

RGBA で 4 成分のベクトル



# 頂点シェーダプログラム

```
uniform mat4 uMVMatrix;  
uniform mat4 uPMatrix;
```

mat4 は、 $4 \times 4$  の行列の型

uniform 変数とは

- ユーザが定義する変数
- (その時刻 (フレーム) に) 全ての頂点で同じ値を持つデータ

# 頂点シェーダプログラム

```
| varying vec4 vColor;
```

varying 変数 (varying variable) とは

- フラグメントシェーダに情報を渡すための変数
- ユーザが定義できる
- 組み込み varying 変数もある
  - gl\_Position
  - gl\_FrontFacing
  - gl\_PointSize

## 頂点シェーダプログラム

```
void main() {  
    gl_Position = uPMatrix * uMVMMatrix * vec4(aVertexPos, 1.0);  
    vColor = aVertexColor;  
}
```

エントリーポイントは main

返値はなし

1. 今処理している頂点の位置（3次元規格化デバイス座標）を4次元にして
2. モデルビュー変換行列をかけて
3. 射影変換行列をかけて
4. 組み込み varying 変数である gl\_Position に代入する

最後にこの頂点の色を varying 変数である vColor に書き込む

# プリミティブ組み立て

primitive assembly

## プリミティブ

- 3 角形<sup>†</sup>
- 線分
- ポイントスプライト

クリッピング処理はここで行われる

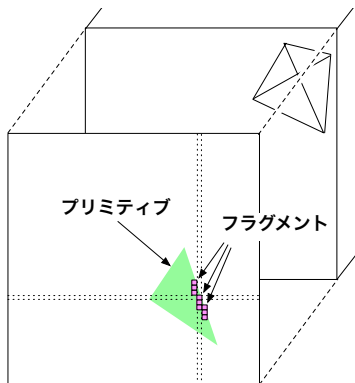
---

<sup>†</sup>OpenGL 1.x では沢山のプリミティブがあったがいまは 3 角形と線分、点のみ。

## ラスタ化

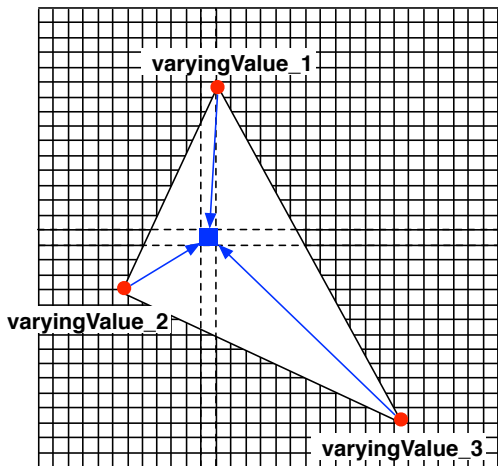
プリミティブからフラグメントを作る処理

フラグメント  $\approx$  ピクセル (様々なテストに合格したフラグメントだけが描画ピクセルになる)



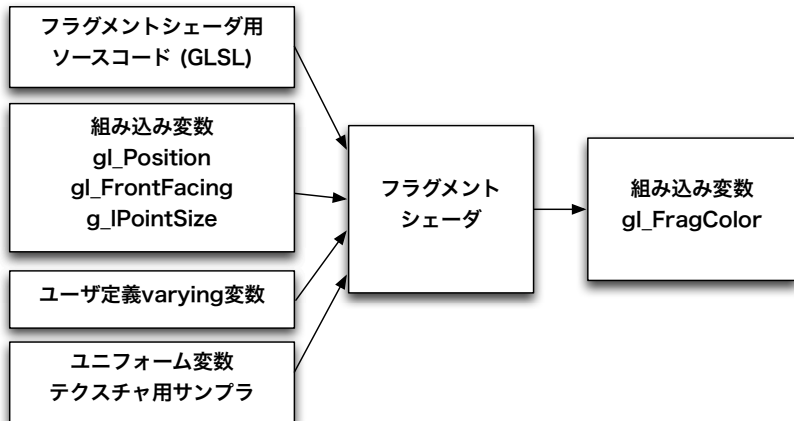
## varying 変数の補間

- 頂点シェーダ からフラグメントシェーダへは varying 変数を通じて情報を送る。
- 各フラグメントの varying 変数値は自動的に線形補間される。



## フラグメントシェーダの入出力

全てのフラグメントで並列処理。シェーディング言語でプログラム。



# フラグメントシェーダプログラム

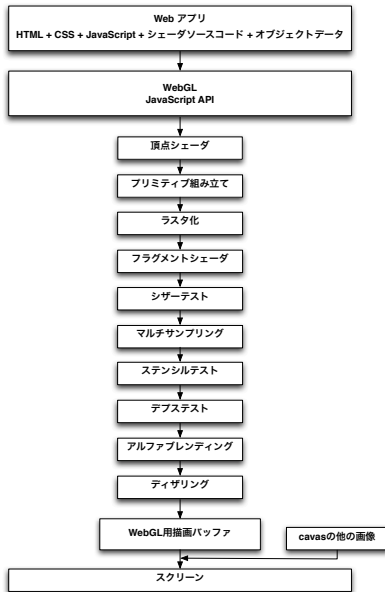
```
precision mediump float; // precision qualifier (精度修飾子)
varying vec4 vColor; // 補間された値

void main() {
    gl_FragColor = vColor;
}
```

精度修飾子：最低保証する精度。



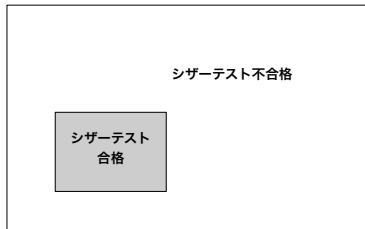
## WebGL のグラフィックスパイプライン (再掲)



## シザーテスト

描画ウィンドウの一部の領域だけを「はさみ ( scissors )」で切り取る<sup>‡</sup>。  
テストに合格したフラグメントだけ描画。不合格フラグメントはそれ以降のパイプラインを通らない 処理の高速化

シザーテストの簡単な例： OpenGL Super Bible (2011, p.112)



<sup>‡</sup>はさみといっても任意の形ではない。長方形のみ。

## マルチサンプリング

アンチエイリアシング = 斜めの線（特にほぼ水平な線）のギザギザをとる方法

マルチサンプリング = 周囲の複数のフラグメントをランダムに選択して色を混ぜる

OpenGL Super Bible (2011, p.382) 参照。

## ステンシルテスト

ステンシルバッファの対応する位置の値と比較テストする。

不合格フラグメントは破棄

OpenGL Super Bible (2011, p.399) 参照。

# デプステスト

既に述べた。

# アルファブレンディング

半透明な物体の表現

後述

# ディザリング

カラーバッファのビット数が少ないとき、中間色を表現する処理。