WebGL によるデータ可視化入門[*1]

頂点シェーダの初歩

陰山 聡

神戸大学 システム情報学研究科 計算科学専攻

2013.05.07[*2]

[*1] 情報可視化論 X021（2013 年前期） LR301 →演習室

[*2] 05/08: レポート提出先アドレス訂正

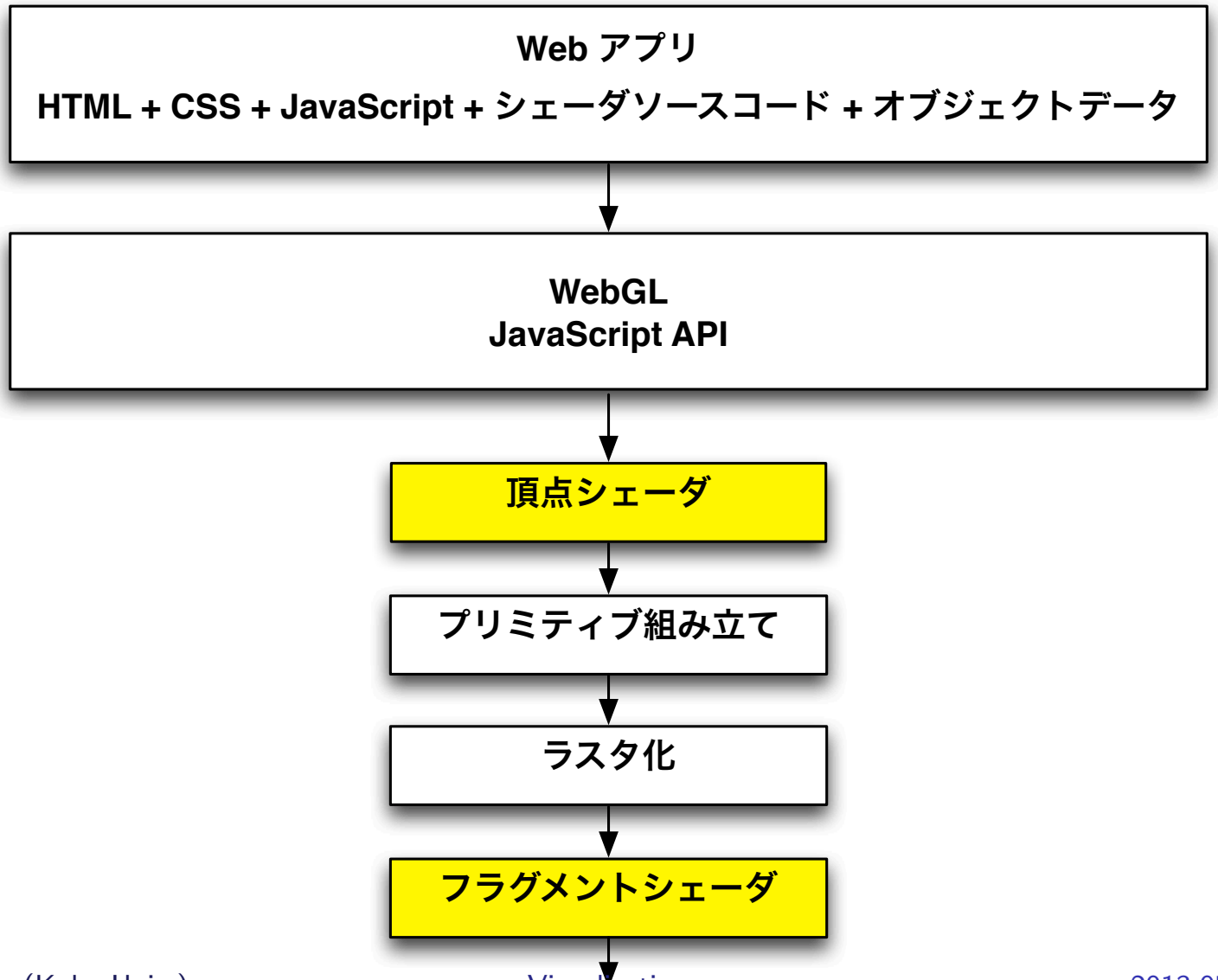
事務連絡

- ブラウザの確認
  - Chrome ブラウザ
  - Firefox ブラウザ
- チェック
  - `http://www.khronos.org/webgl/wiki/Demo_Repository`
- ID とパスワード：先週述べた

前回の復習

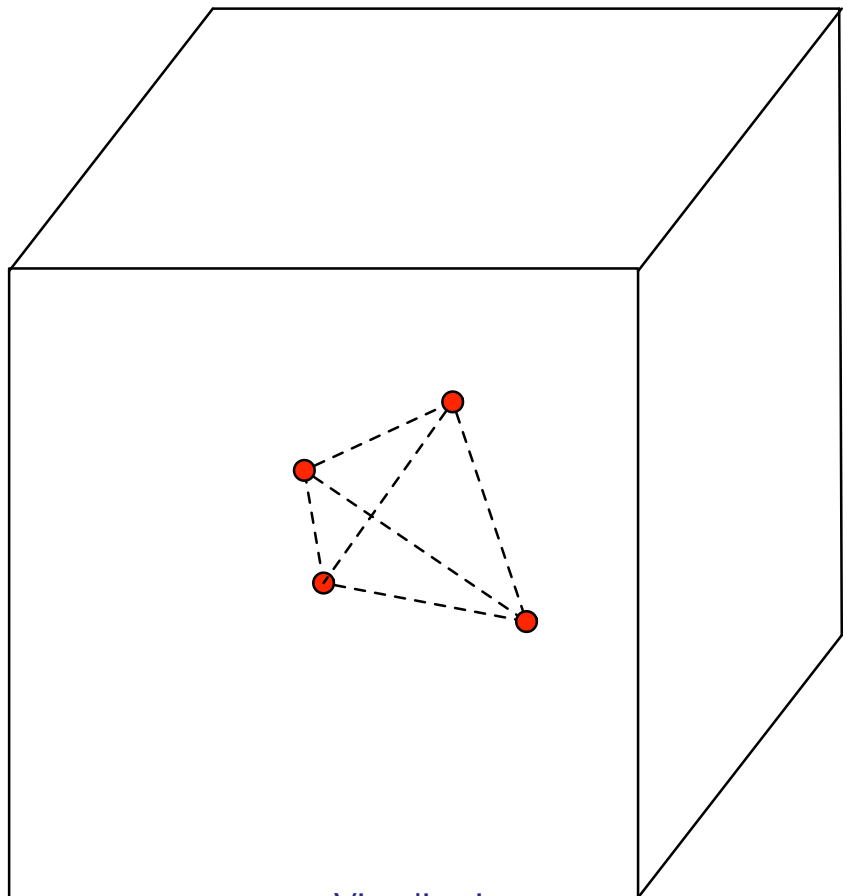
シェーダ

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

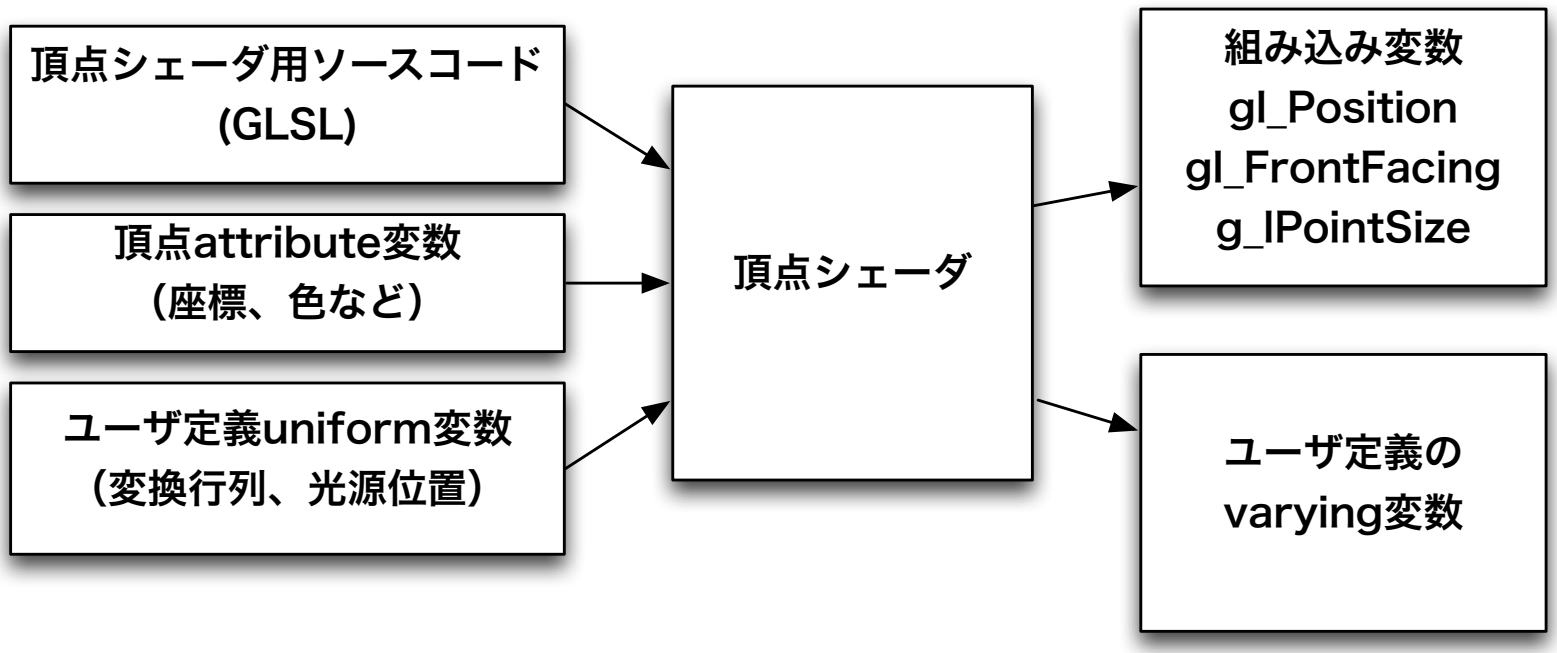


頂点シェーダ

- 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;
atrribute 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;
atrribute 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、4×4 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;
```

varing 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 * uMVMatrix * 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[*15]
- 線分
- ポイントスプライト

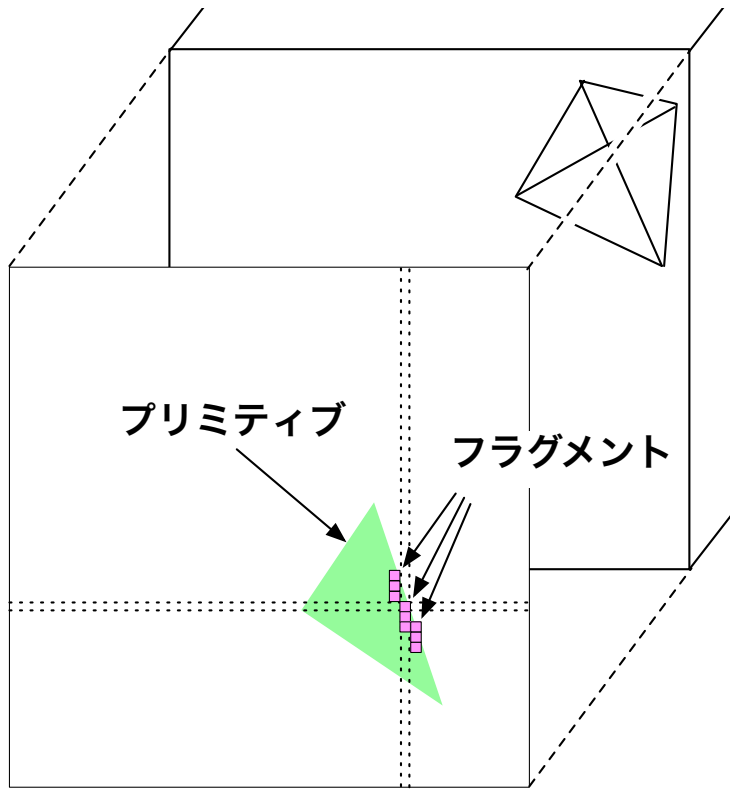
クリッピング処理はここで行われる

[*15] 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

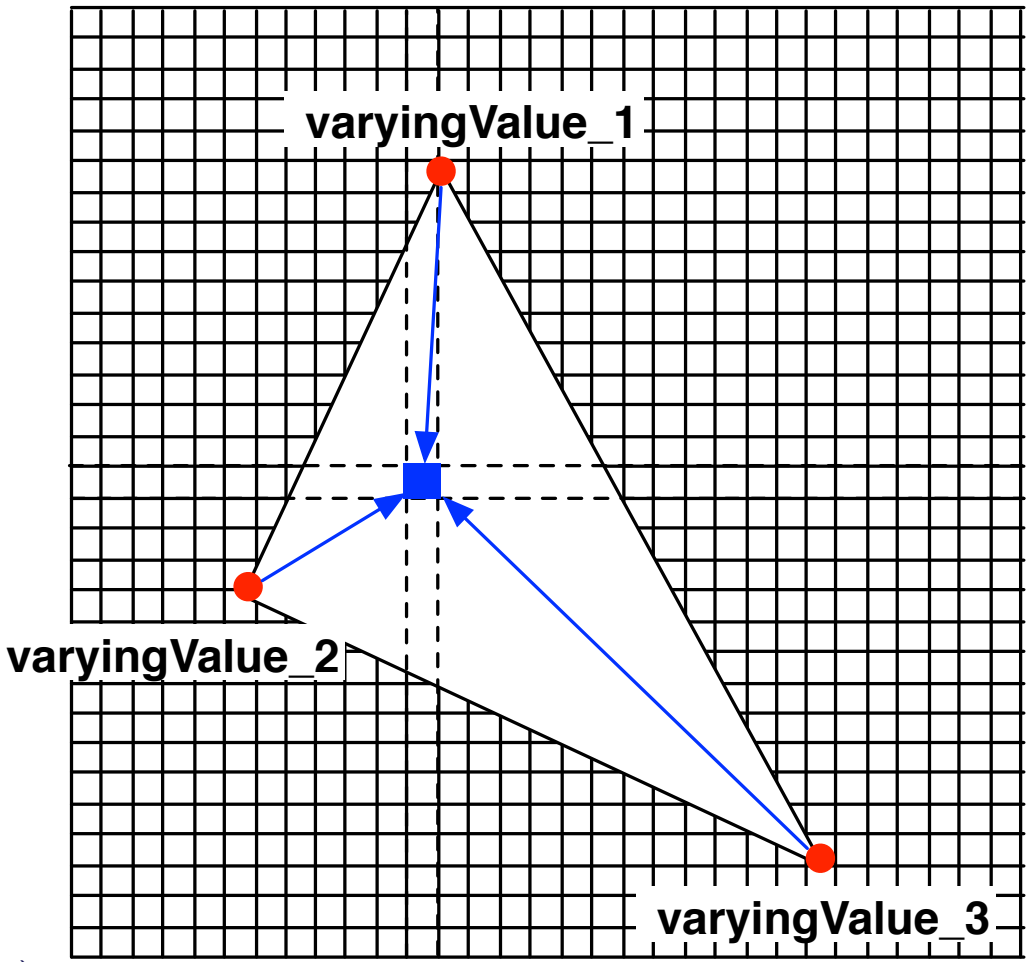
プリミティブからフラグメントを作る処理

フラグメント ≈ 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）



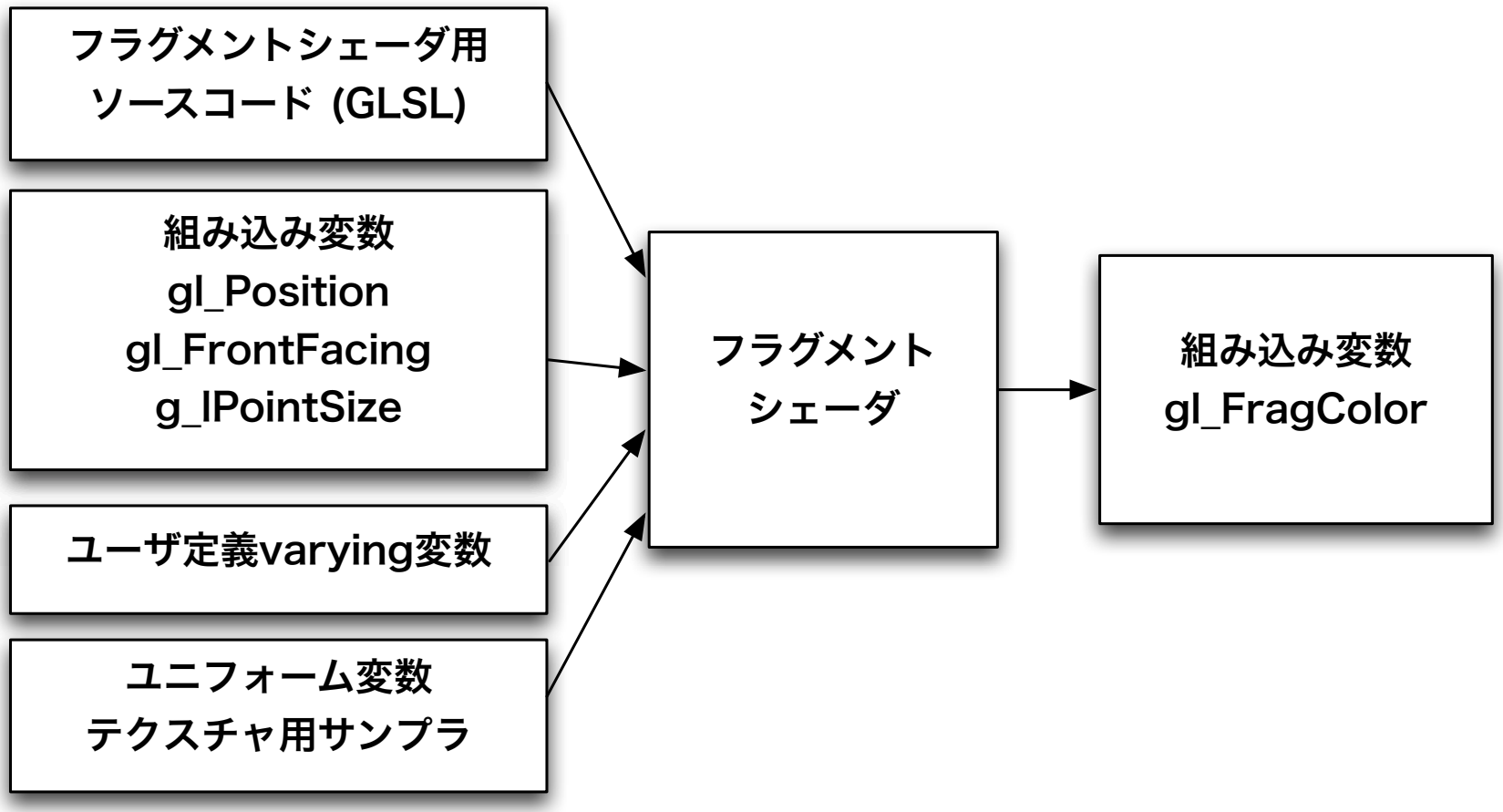
varying 変数の補間

- 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

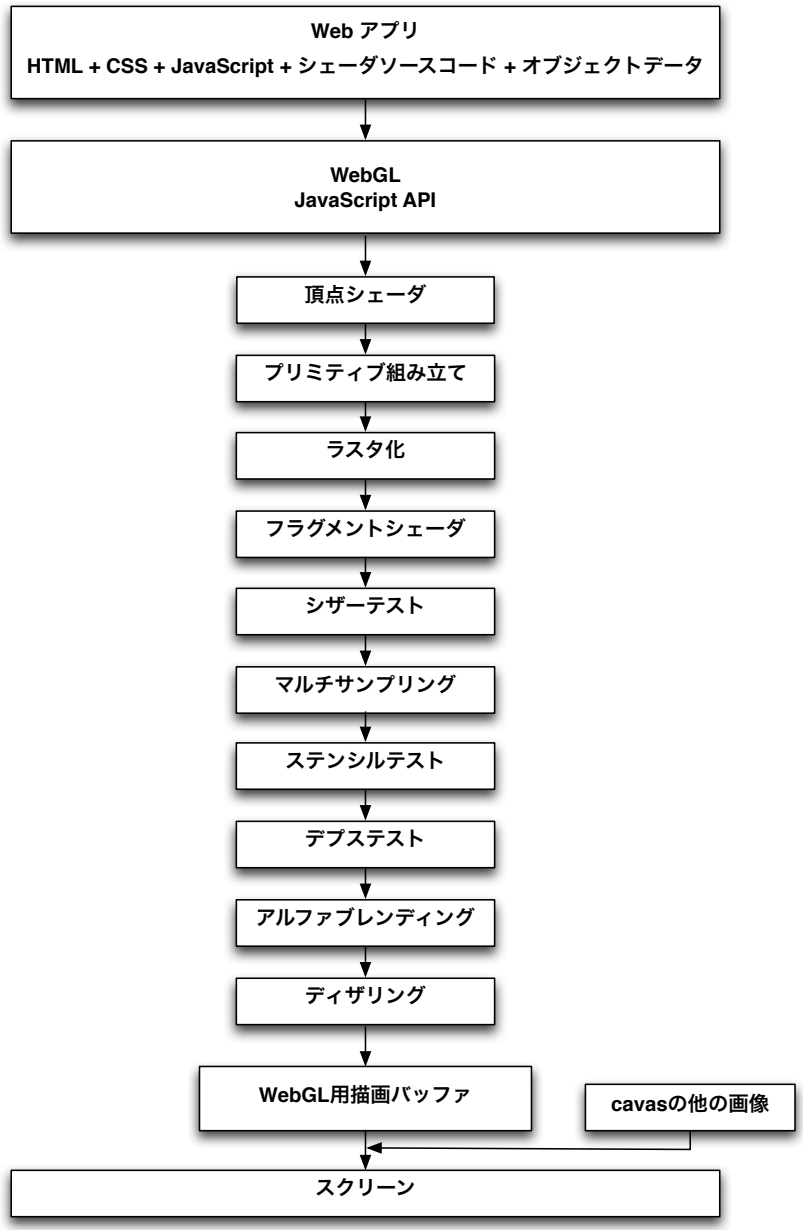
```
precision mediump float; // precision qualifier （精度修飾子）

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）

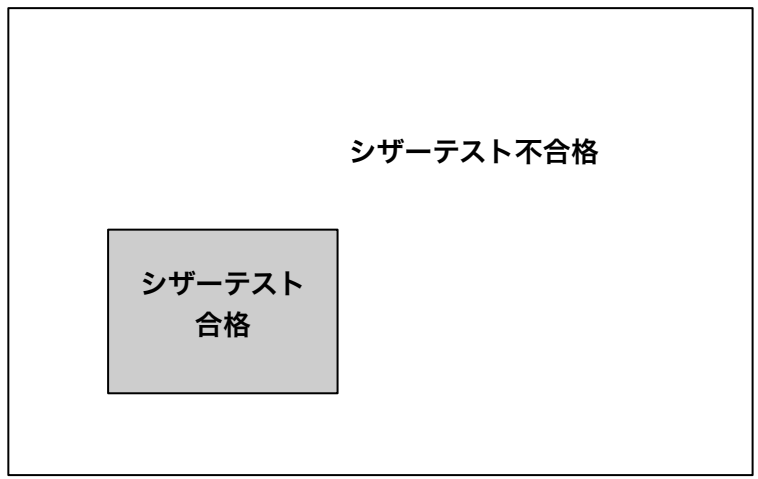


シザーテスト

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[*22]。

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 (Richard S. Wright et al., 2011, p.112)



[*22]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

アンチエイリアジング＝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 (Richard S. Wright et al., 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 (Richard S. Wright et al., 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

【省略】

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。

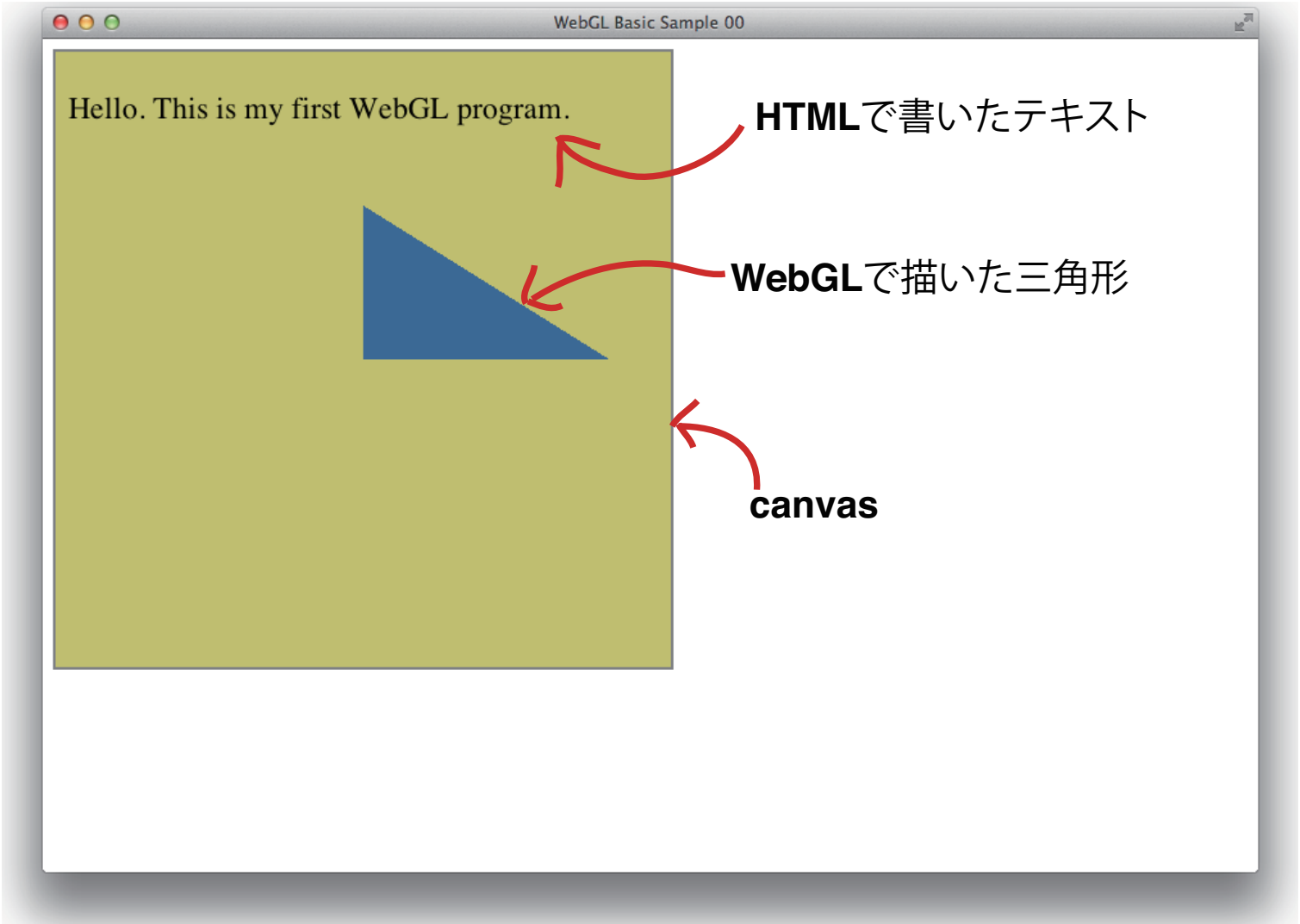
WebGL による三角形の描画

ソースコード

`http://tinyurl.com/kageyama2013v`

“WebGL での三角形の描画” webgl_sample_triangle_00.html

3 角形の描画


WebGL Basic Sample 00
Hello. This is my first WebGL program.
HTMLで書いたテキスト
WebGLで描いた三角形
canvas

webgl_sample_triangle_00.html

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>WebGL Sample Triangle 00</title>
<meta charset="utf-8">

<style type="text/css">
    canvas {
        border: 2px solid grey;
    }
    .text {
        position:absolute;
        top: 40px;
        left: 20px;
        font-size: 1.5em;
```

webgl_sample_triangle_00.html

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>WebGL Sample Triangle 00</title>
<meta charset="utf-8">

<style type="text/css">
    canvas {
        border: 2px solid grey;
    }
    .text {
        position:absolute;
        top: 40px;
        left: 20px;
        font-size: 1.5em;
```

```
        color: black;
    }
</style>

<script type="text/javascript">
var gl;
var canvas;
var shaderProgram;
var vertexBuffer;

function createGLContext(canvas) {
  var names = ["webgl", "experimental-webgl"];
  var context = null;
  for (var i=0; i<names.length; i++) {
    try {
```

```
      context = canvas.getContext(names[i]);
    } catch(e) {}
    if (context) {
      break;
    }
  }
  if (context) {
    context.viewportWidth = canvas.width;
    context.viewportHeight = canvas.height;
  } else {
    alert("Failed to create context.");
  }
  return context;
}
```

```
function loadShader(type, shaderSource) {
  var shader = gl.createShader(type);
  gl.shaderSource(shader,shaderSource);
  gl.compileShader(shader);

  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    alert("Error compiling shader" + gl.getShaderInfoLog(
        shader));
    gl.deleteShader(shader);
    return null;
  }
  return shader;
}

function setupShaders() {
  var vertexShaderSource =
```

```
var fragmentShaderSource =
  "precision mediump float; \n" +
  "void main() {     \n" +
  "   gl_FragColor = vec4(0.2, 0.4, 0.6, 1.0); \n" +
  "} \n";

var vertexShader = loadShader(gl.VERTEX_SHADER,
    vertexShaderSource);
var fragmentShader = loadShader(gl.FRAGMENT_SHADER,
    fragmentShaderSource);

shaderProgram = gl.createProgram();
```

```
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
        {
      alert("Failed to setup shader.");
    }

    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute =
      gl.getAttribLocation(shaderProgram, "aVertexPosition");
}

function setupBuffers() {
```

```
  vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  var triangleVertices = [
        0.0, 0.0, 0.0,
        0.8, 0.0, 0.0,
        0.0, 0.5, 0.0
  ];
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(
      triangleVertices),
                gl.STATIC_DRAW);
  vertexBuffer.itemSize = 3;
  vertexBuffer.numberOfItems = 3;
}

function draw() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
```

```
  gl.clear(gl.COLOR_BUFFER_BIT);

  gl.vertexAttribPointer(shaderProgram.
      vertexPositionAttribute,
                         vertexBuffer.itemSize, gl.FLOAT,
                             false, 0, 0);
  gl.enableVertexAttribArray(shaderProgram.
      vertexPositionAttribute);
  gl.drawArrays(gl.TRIANGLES, 0, vertexBuffer.numberOfItems);
}

function startup() {
  canvas = document.getElementById("myGLCanvas");
  gl = createGLContext(canvas);
  setupShaders();
  setupBuffers();
  gl.clearColor(0.8, 0.8, 0.4, 1.0);
  draw();
```

```
}

</script>
</head>

<body onload="startup();">
  <canvas id="myGLCanvas" width="480" height="480"></canvas>
  <div class="text">Hello. This is my first WebGL program.</
      div>
</body>
</html>
```

コンテキスト作成

- HTML5 の canvas 要素の getContext method を呼ぶ
- 引数（文字列）は “experimental-webgl” を渡す[*42]。
- getContext method が返すのは WebGLRenderingContext オブジェクト。
  - WebGL の変数や関数はこの WebGLRenderingContext のメンバー。
  - このサンプルプログラムでは gl という変数にこのオブジェクトを代入。

[*42] 将来これは “webgl” に変更されるので、どちらにも対応できるようにしておく。

シェーダプログラム

- 頂点シェーダ用の（GLSL 言語の）ソースコードは、GPU に送り、GPU にコンパイルさせる。
- HTML（あるいは JavaScript）の文脈では GLSL ソースコードは単なる文字列。
- ここでは文字列変数 fragmentShaderSource にソースコード文字列を代入して送っている。
- 【注意】普通はこうはしない。もっと便利な方法がある。後述。

シェーダプログラムの作成

1. 頂点シェーダオブジェクトを作る
   1.1 頂点シェーダソースコードを頂点シェーダオブジェクトにロードする
   1.2 コンパイルする
2. フラグメントシェーダオブジェクトを作る
   2.1 フラグメントシェーダソースコードをフラグメントシェーダオブジェクトにロードする
   2.2 コンパイルする
3. プログラムオブジェクトを作る
4. プログラムオブジェクトに頂点シェーダオブジェクトをアタッチする（こちらが先）
5. プログラムオブジェクトにフラグメントシェーダオブジェクトをアタッチする
6. リンクする
7. このプログラムオブジェクトを WebGLRenderingContext に登録する

（GLSL の）頂点属性について[*47]

頂点シェーダ内で定義できる attribute 変数

float, vec2, vec3, vec4,

mat2, mat3x2, mat4x2,

mat2x3, mat3, mat4x3,

mat4, mat2x4, mat3x4

宣言されていても使われていない attribute 変数 → アクティブでない

それ以外 → アクティブ

アクティブな attribute 変数全体はまとめて整数インデックスで管理される[*46] それぞれの attribute 変数にはこのインデックスで参照する。

[*46]最大サイズは MAX_VERTEX_ATTRIBS

[*47]詳細は「OpenGL 4.0 グラフィックスシステム」 (Segal et al., 2010, p.60) を参照。

`int BindAttribLocation(unit program, unit index, const char *name);`

- name という attribute 変数を index にバインドする（上書き）。
- program はリンクしたプログラムオブジェクトの名前。
- 自分でバインドしない場合は GL が自動的にバインド する。
- そのときの index は以下の関数で取得できる。

`int GetAttribLocation(unit program, const char *name);`

attribute 変数名 (name) を指定して、その変数がどのインデックスにバインドされているかを問い合わせる。

シェーダプログラム

gl.useProgram(shaderProgram);

shaderProgram . vertexPositionAttribute =
gl.getAttribLocation(shaderProgram, "aVertexPosition ");

上で、シェーダプログラムの vertexPositionAttribute というプロパティ名は任意。

```
shaderProgram.anyNameIsOK =
  gl.getAttribLocation(shaderProgram, "aVertexPosition");
```

バッファの作成

```
function setupBuffers() {
  vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  var triangleVertices = [
        0.0, 0.0, 0.0,
        0.8, 0.0, 0.0,
        0.0, 0.5, 0.0
  ];
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(
      triangleVertices),
                gl.STATIC_DRAW);
  vertexBuffer.itemSize = 3;
  vertexBuffer.numberOfItems = 3;
}
```

バッファの作成

- 三角形の頂点データをシェーダに送るためにはバッファオブジェクトを使う。
- バッファオブジェクトは WebGLRenderingContext の createBuffer() メソッドで作成する。
- 作成した WebGLBuffer オブジェクトを JavaScript のグローバル変数 vertexBuffer に代入
- vertexBuffer を「現在の配列バッファオブジェクト」にバインドする。
- 後で使えるようにこの頂点配列の情報（頂点数等）も書いておく（どうやって渡してもよい）。

描画

```
function draw() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT);

  gl.vertexAttribPointer(shaderProgram.
      vertexPositionAttribute,
                         vertexBuffer.itemSize, gl.FLOAT,
                             false, 0, 0);
  gl.enableVertexAttribArray(shaderProgram.
      vertexPositionAttribute);
  gl.drawArrays(gl.TRIANGLES, 0, vertexBuffer.numberOfItems);
}
```

演習

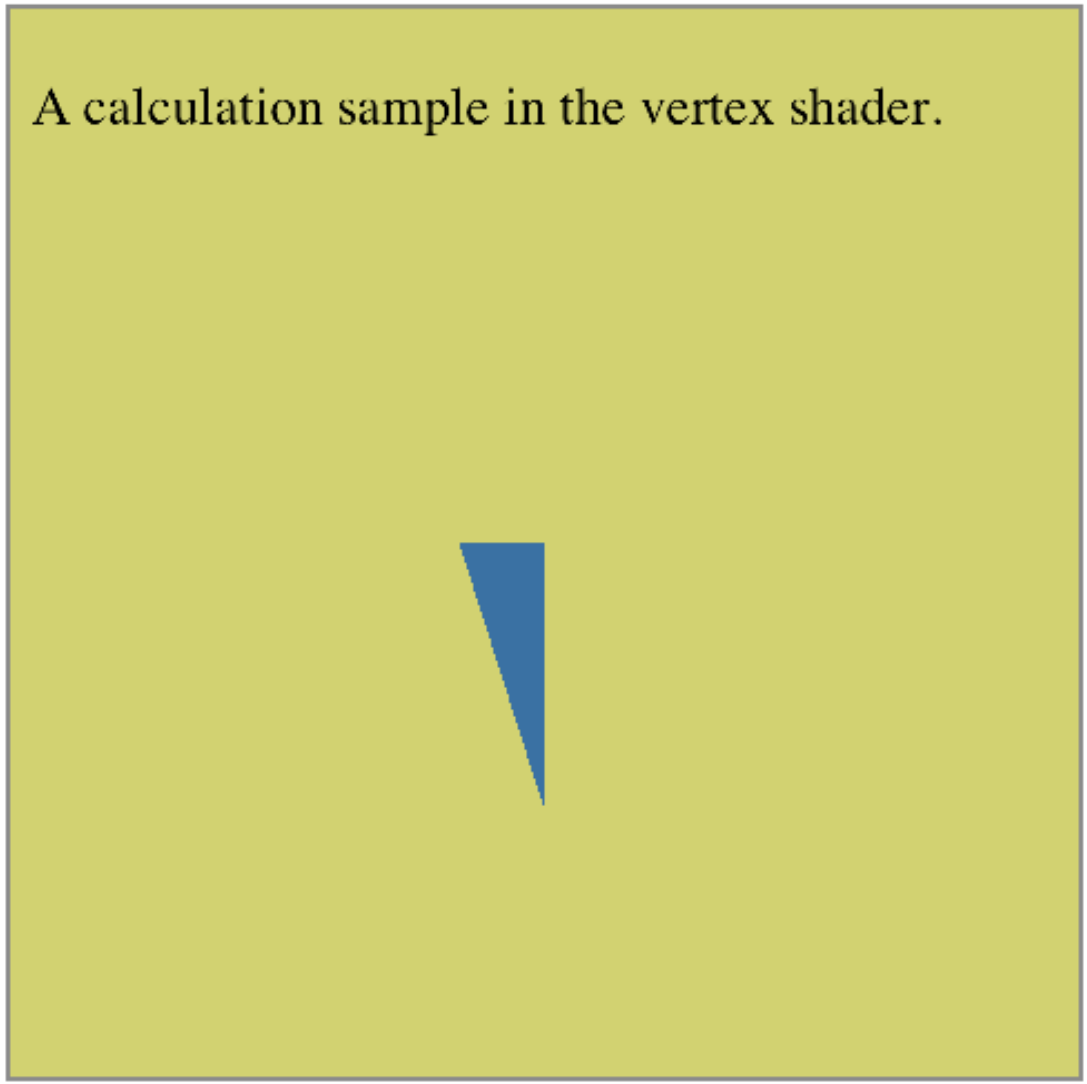
頂点シェーダでの演算

- 頂点シェーダの中で簡単な数値計算をしてみよう。
- 例題：webgl_sample_triangle_01.html

webgl_sample_triangle_01.html

```
function setupShaders() {
  var vertexShaderSource =
    "attribute vec3 aVertexPosition;  \n" +
    "void main() { \n" +
    "   gl_Position = vec4(aVertexPosition, 1.0); \n" +
    "   gl_Position *= vec4(-0.2,-1.0,1.0,1.0); //  Here !! \n
        " +
    "} \n";
```

頂点シェーダでの演算



レポート課題

- webgl_sample_triangle_00.html の頂点シェーダを変更して元の三角形とは別の図形（図）を描け。
- 提出はメールで。添付ファイルは二つ。
  1. レポートの PDF ファイル
  2. 作成した HTML ファイル
- gmail アドレス：~~kageyama_lecture@...~~ 【訂正】kageyama.lecture@...
- メールのタイトル： 学籍番号_氏名
- レポート（PDF ファイル形式に限る）には以下を記述すること
  - 学籍番号と氏名
  - どのような図形を描いたか
  - 描いた図形のキャプチャ図
  - 自分の HTML ファイルをウェブで公開された場合の希望名（フルネーム／名字のみ／イニシャル）
- 締め切り： 5/13（月）正午

索引

References

Richard S. Wright, J., Haemel, N., Sellers, G., and Lipchak, B. (2011).
OpenGL SuperBible: Comprehensive Tutorial and Reference.
Pearson Education, Inc, Boston, MA, 5th edition.

Segal, M., Akeley, K., and 松田晃一他訳 (2010).
OpenGL 4.0 グラフィックスシステム.
カットシステム.