DrawArrays と DrawElements

情報可視化論 第 05 回

陰山 聡

神戸大学 システム情報学研究科 計算科学専攻
［情報基盤センター分館 第 1 演習室］

2015.05.19

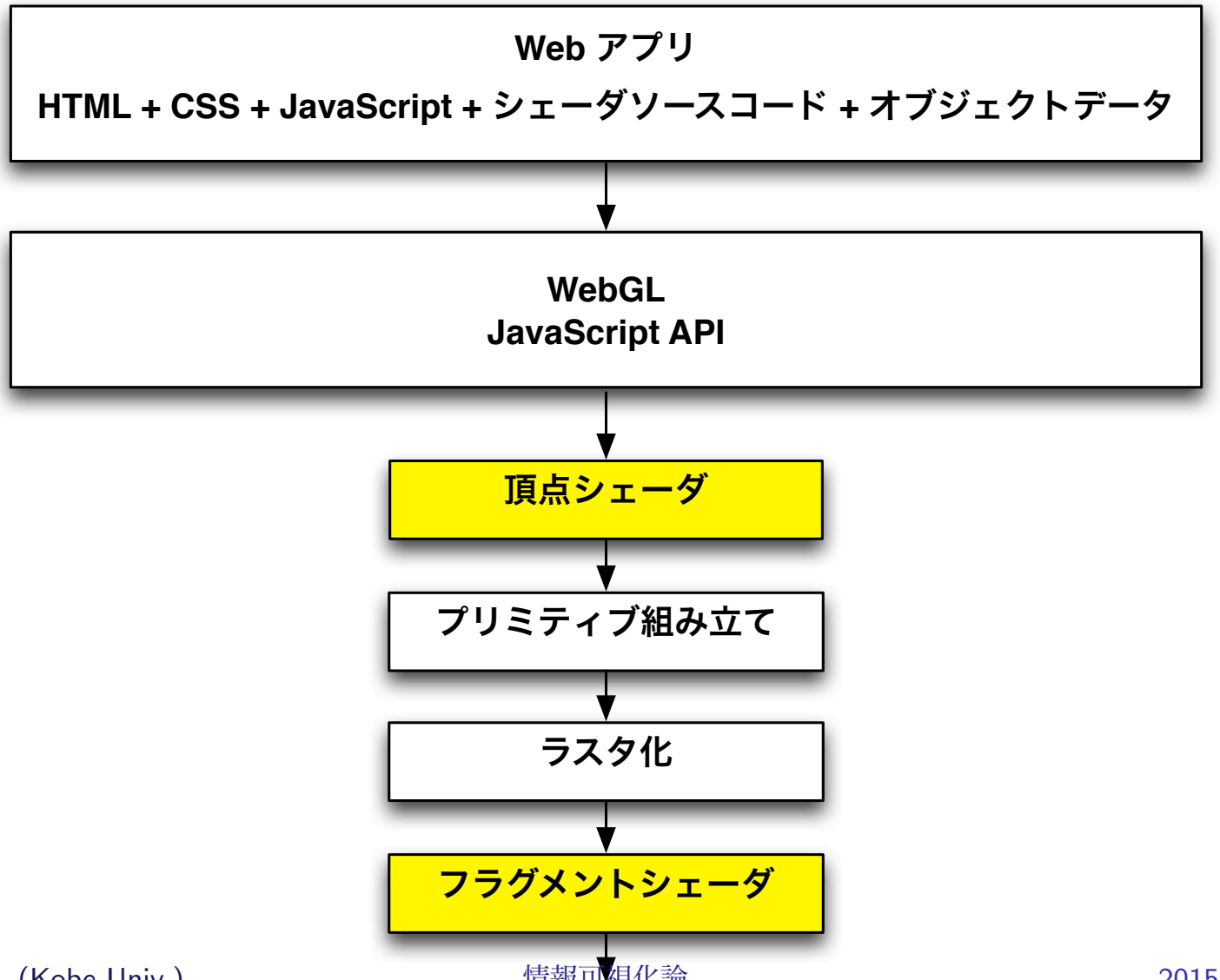
事務連絡

- レポートメールには全員返事をした。
- 本講義のレポートに時間をかけすぎないように（修士論文研究とのバランスを考えましょう）。
- 作品の一部をウェブに掲載予定
- 来週休講

前回の復習

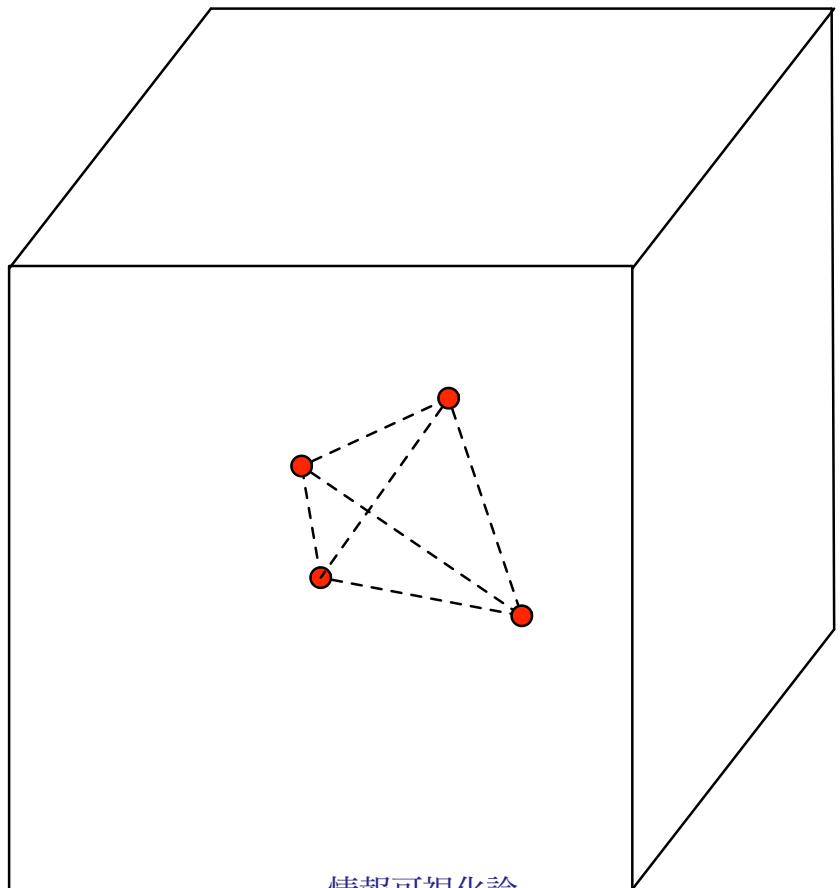
シェーダ

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

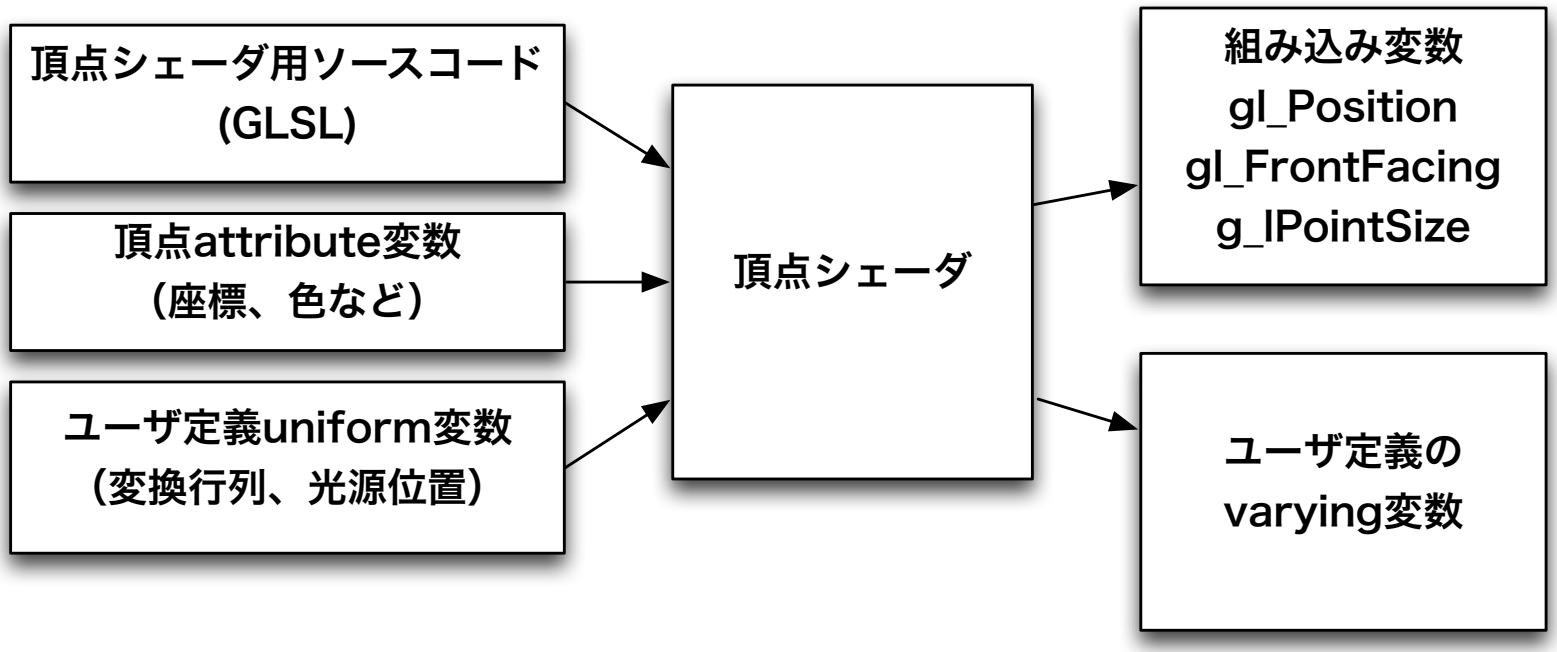


頂点シェーダ

- 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;
}
```

描画：DrawArraysとDrawElements

最も基本的な描画

指定した色で全てのピクセルを描く

`gl.clear()`

色の指定 `gl.clearColor()`

「背景色」

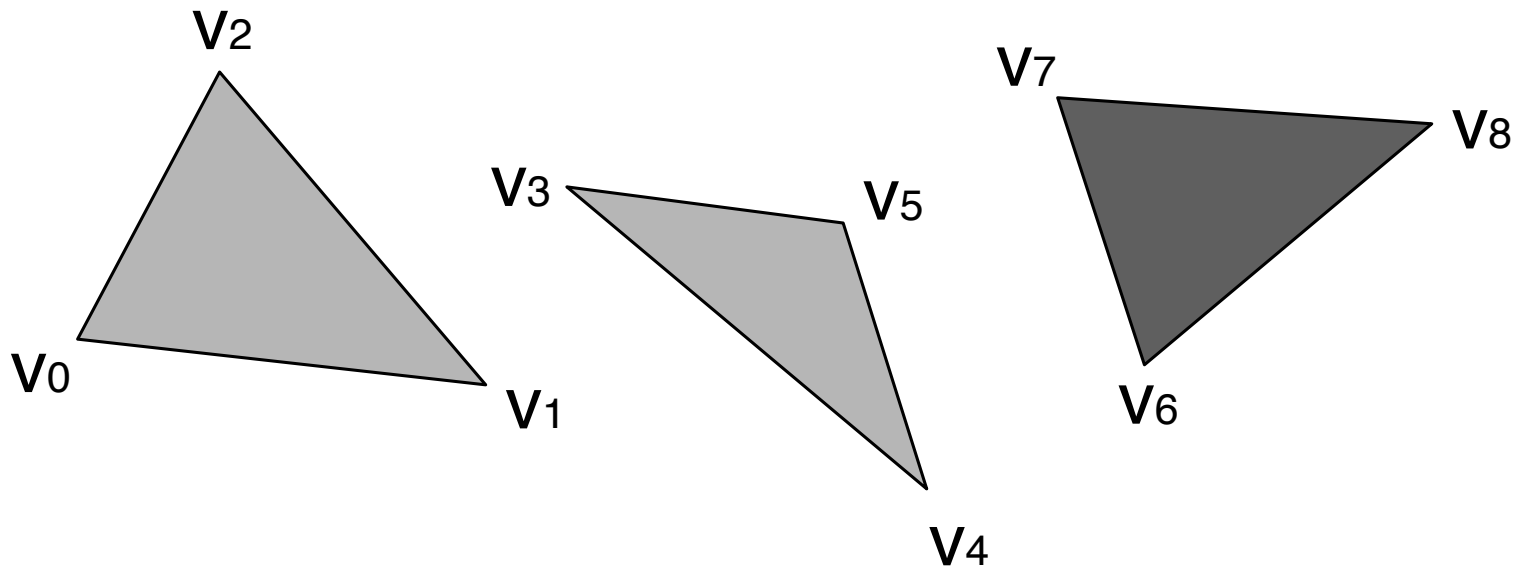
プリミティブ

面は 3 角形で作る

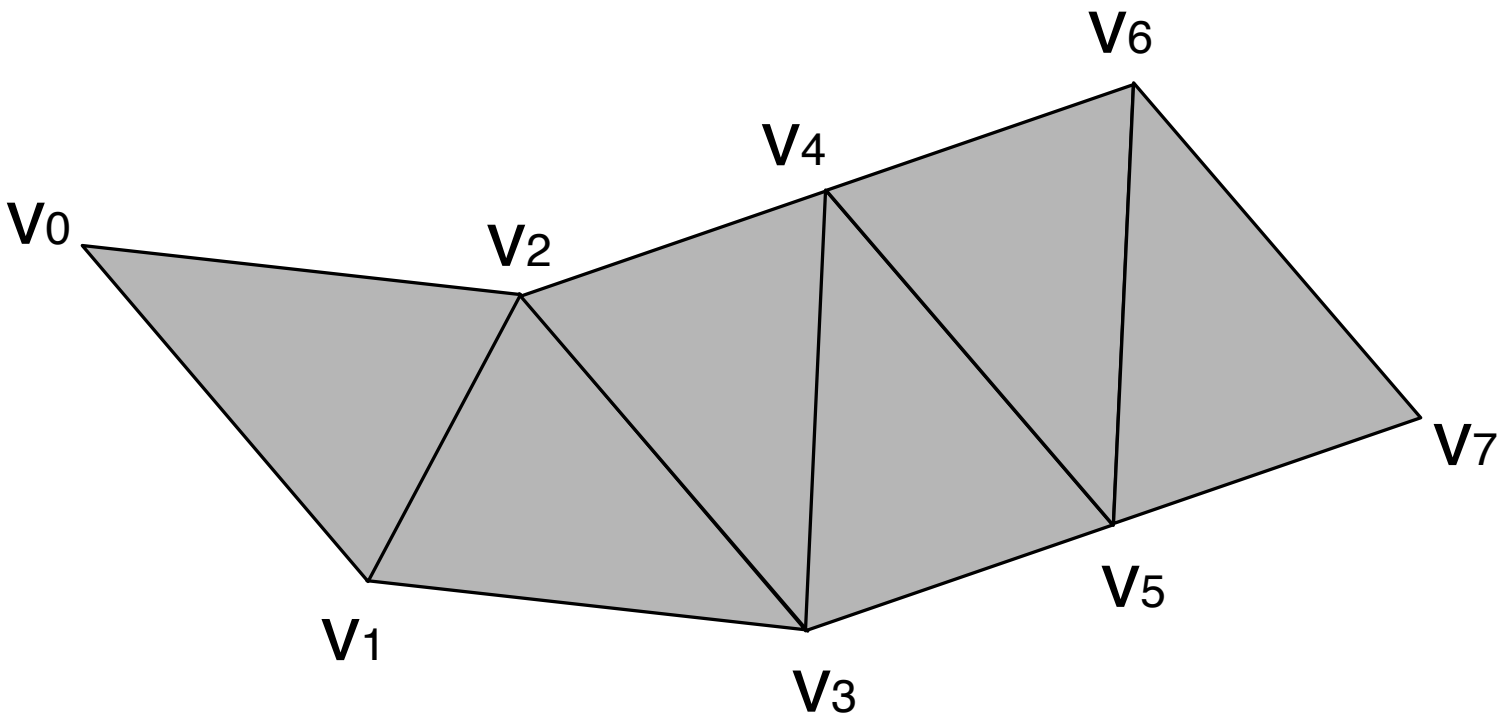
OpenGL 1.x と基本は同じ（ただしプリミティブは三角形だけになった。）

TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN

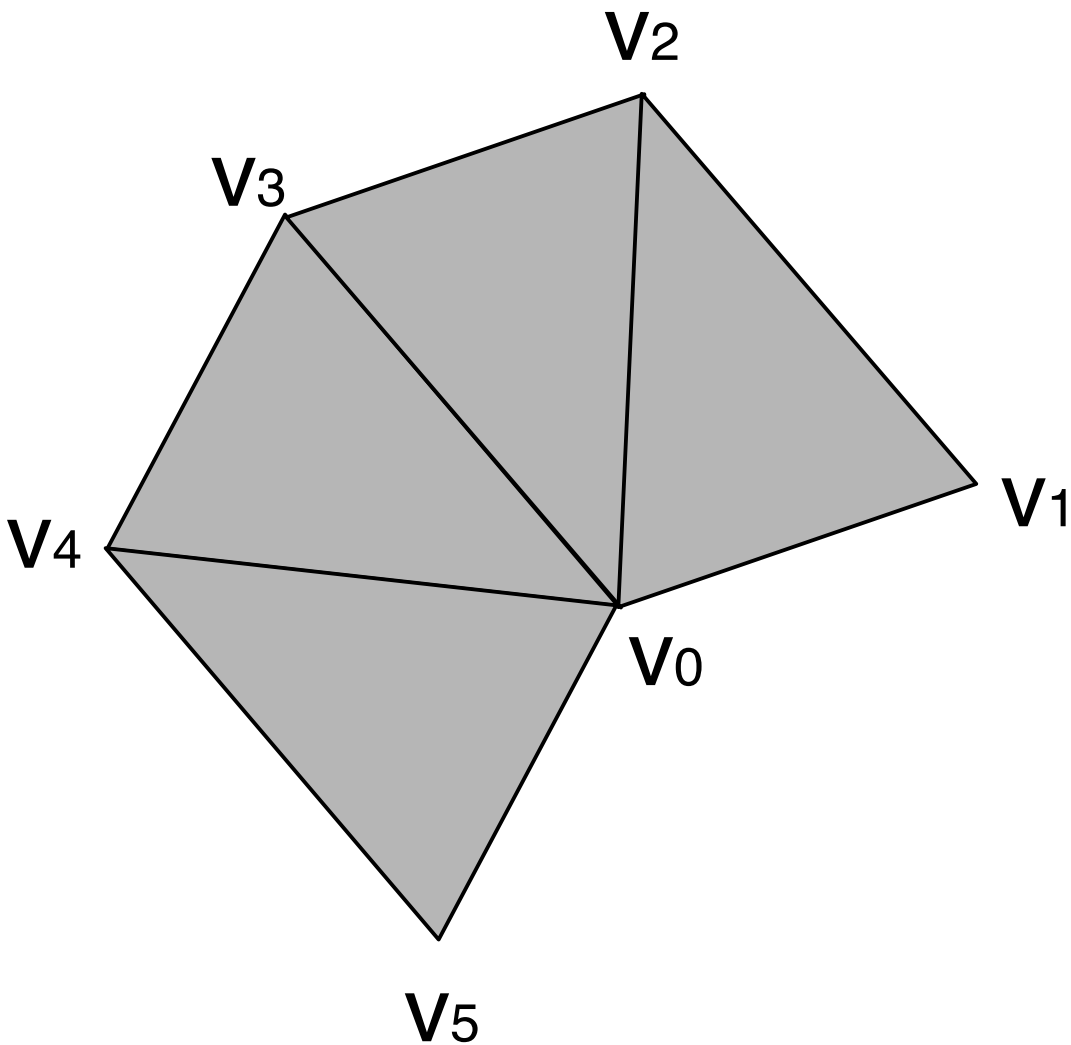
gl.TRIANGLES



gl.TRIANGLE_STRIP



gl.TRIANGLE_FAN



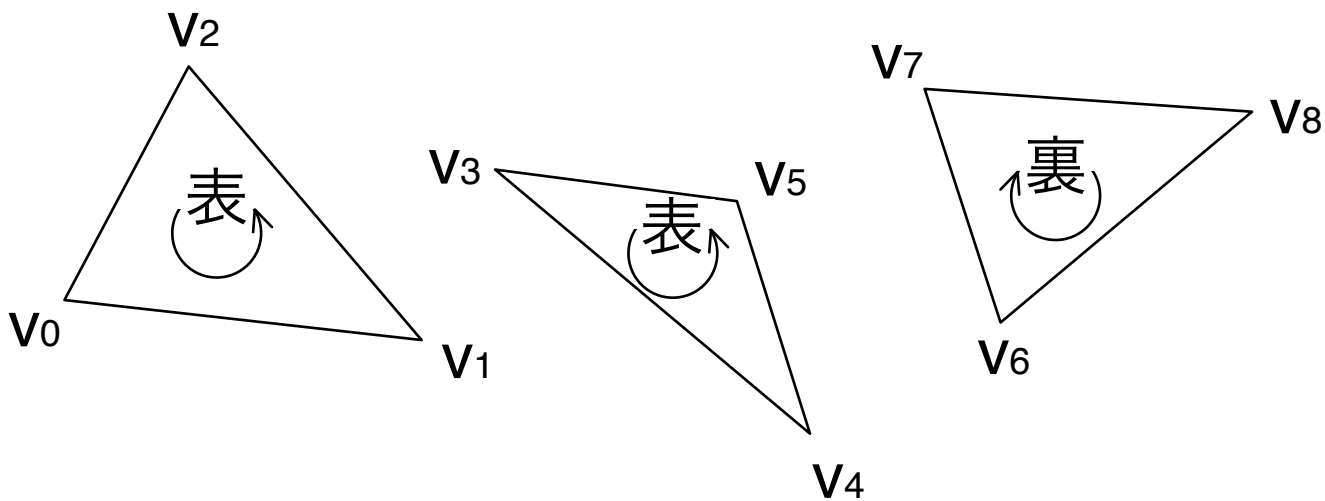
Front Face と Winding Order

三角形には表面と裏面がある

表面は頂点の番号順で決まる。デフォルトは “右手系”。

反時計方向（Counter Clock Wise, CCW）

逆は時計方向（Clock Wise, CW）



裏面のカリング

裏面を見ることがない場合、裏面のラスター処理は省略すればいい。

高速化。

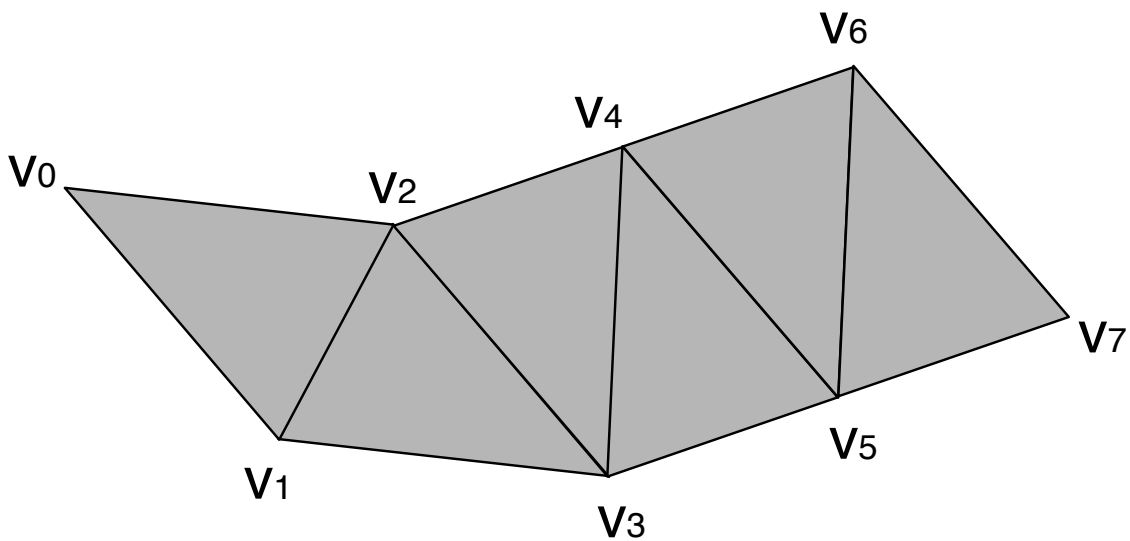
カリング （culling）

```
gl.frontFace(gl.CCW);        // デフォルト
gl.enable(gl.CULL_FACE);     // デフォルトでは disabled
gl.cullFace(gl.BACK);        // デフォルト
```

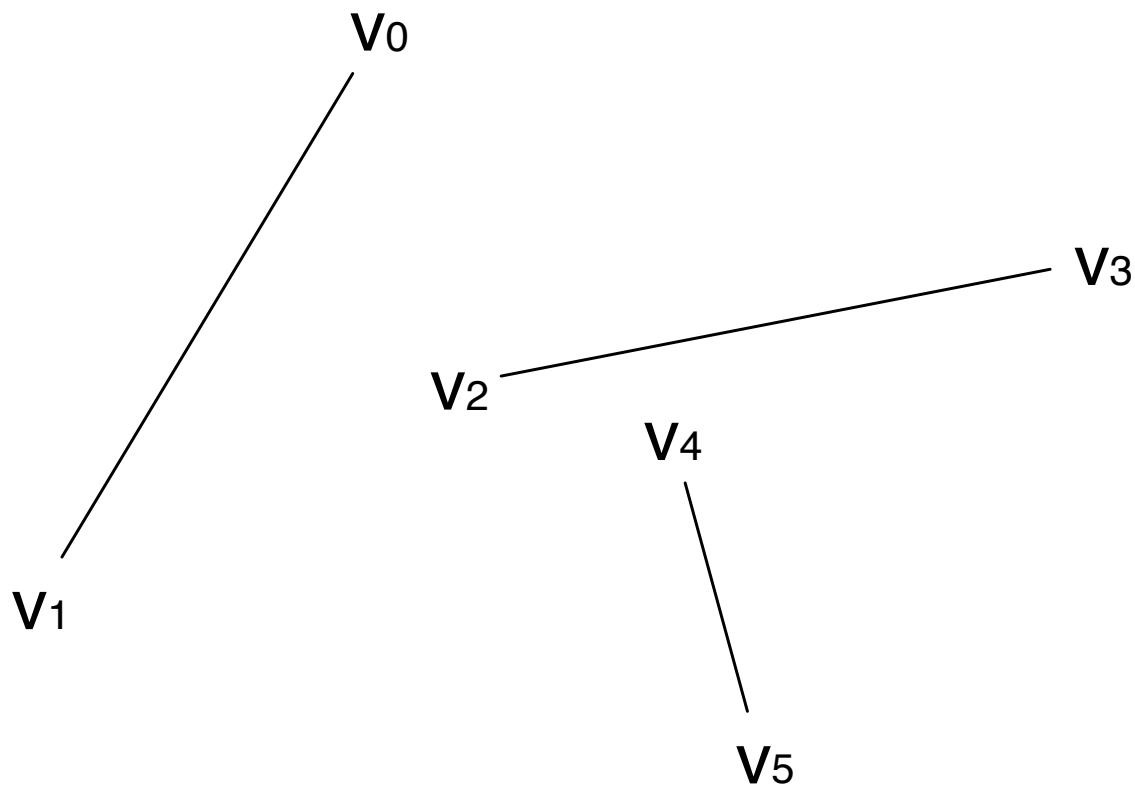
TRIANGLE_STRIP

ワインディングオーダーを保存した三角形列を自動的に構成

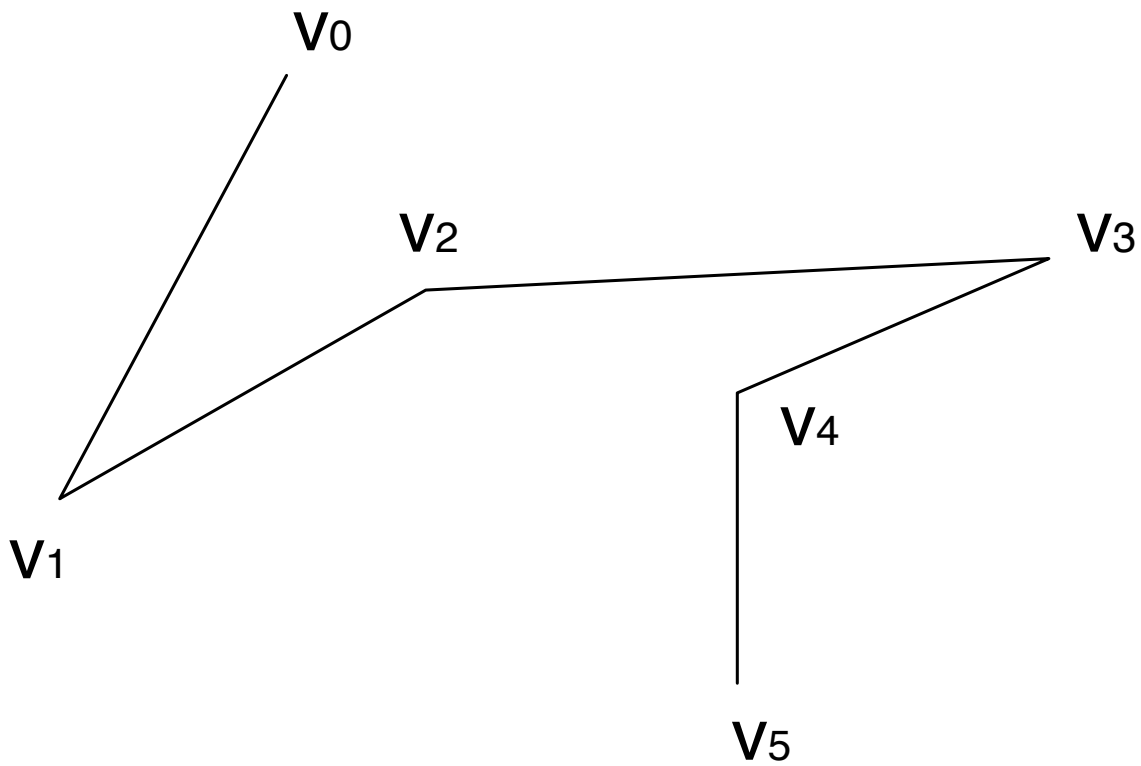
三角形 1	v_0	v_1	v_2				
三角形 2		v_2	v_1	v_3			
三角形 3			v_2	v_3	v_4		
三角形 4				v_4	v_3	v_5	
三角形 5					v_4	v_5	v_6
三角形 6						v_6	v_5	v_7



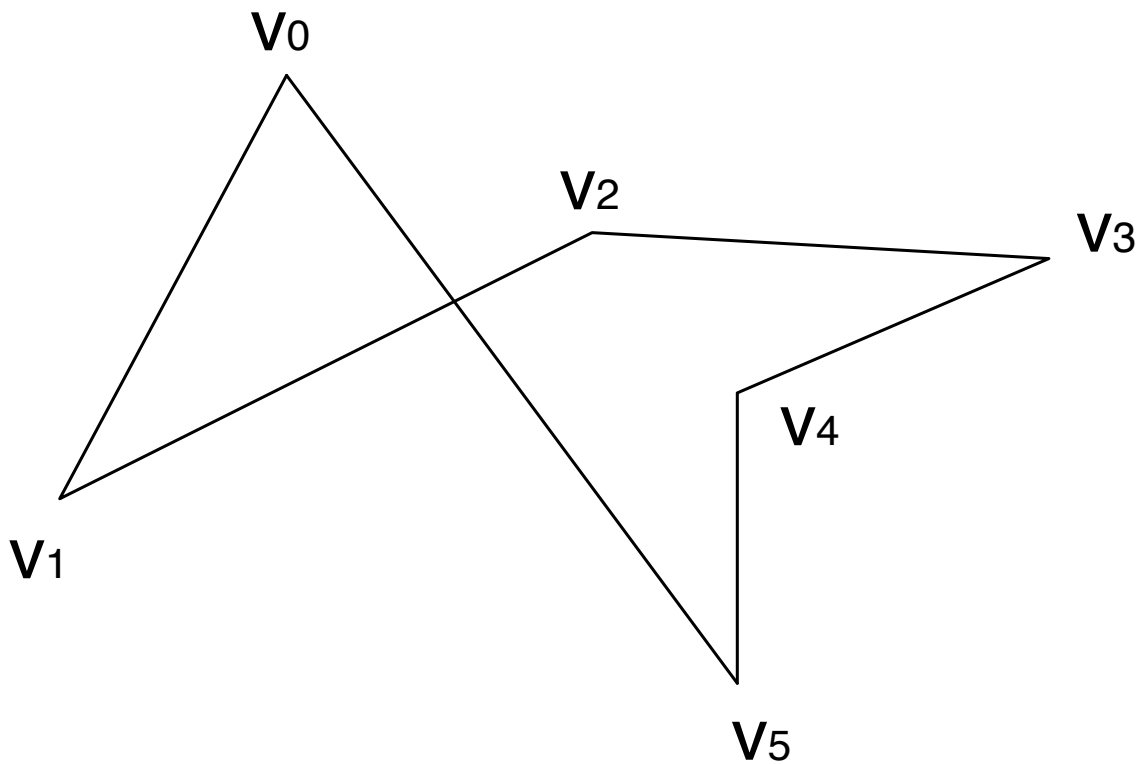
gl.LINES



gl.LINE_STRIP



gl.LINE_LOOP



点

ポイントスプライト（point sprite）

“大きさを持った点”

`gl_PointSize` で指定（シェーダで）

二つの描画メソッド

`gl.drawArrays()`： 配列に納められた順番通りに頂点からプリミティブを構成する

`gl.drawElements()`：別の配列（要素配列）を使って頂点を再利用する

gl.drawArrays()

```
void drawArrays(GLenum mode, GLint first, GLsizei count)
```

ここで mode は gl.X: X={POINTS, LINES, LINE_LOOP, LINE_STRIP, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN}

first は頂点データ配列のうち、最初に使用する要素のインデックス

count は何個の頂点を使うか

何の配列を使うかの指定がないことに注意。

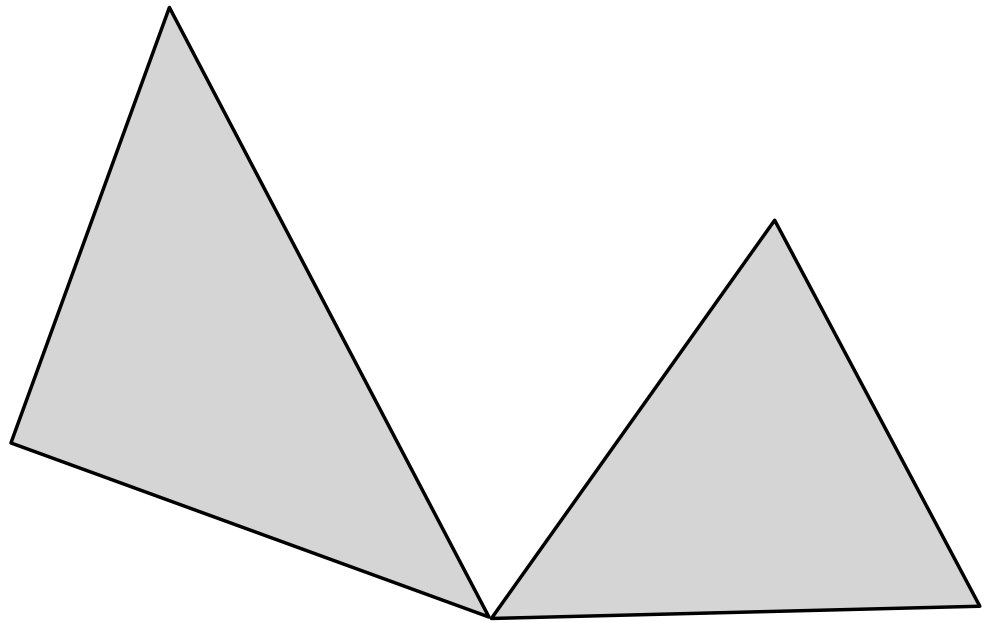
描画する頂点データ配列は、gl.ARRAY_BUFFER と決まっている。

gl.ARRAY_BUFFER にバインドされた配列バッファに頂点の座標を入れる。

ソースコード

```
// バッファオブジェクト作成
  vertexBuffer = gl.createBuffer();
// それをバインドする
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
//  （普通の）配列で座標を用意
  var triangleVertices = [0.0, ...
// 普通の配列から型付き配列を作り、それをバッファにアップロード
  gl.bufferData(gl.ARRAY_BUFFER,
                new Float32Array(triangleVertices)...
// 頂点シェーダの属性としてこのバッファを使うことを指定する
  gl.vertexAttribPointer(shaderProgram.
      vertexPositionAttribute,...
// 頂点シェーダの頂点属性配列を使うことを宣言
  gl.enableVertexAttribArray(shaderProgram.
      vertexPositionAttribute);
// 三角形描画
  gl.drawArrays(gl.TRIANGLES, ...
```

頂点の重複



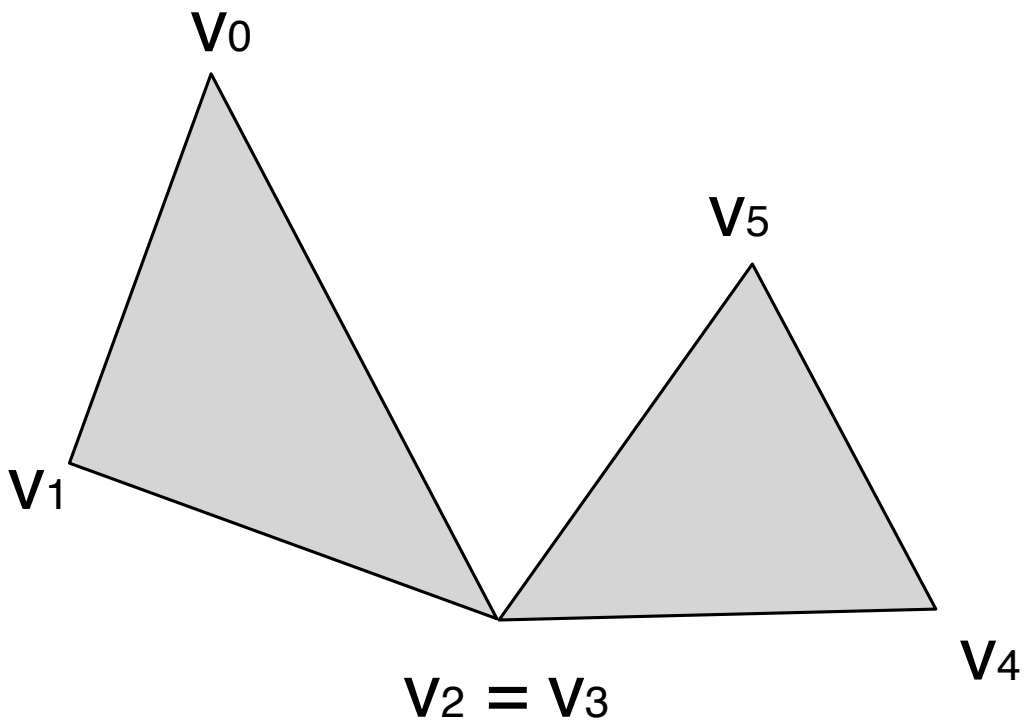
drawArrays() で描く場合

配列バッファ

v_0	v_1	v_2	v_3	v_4	v_5

を用意して gl.TRIANGLES で描く。

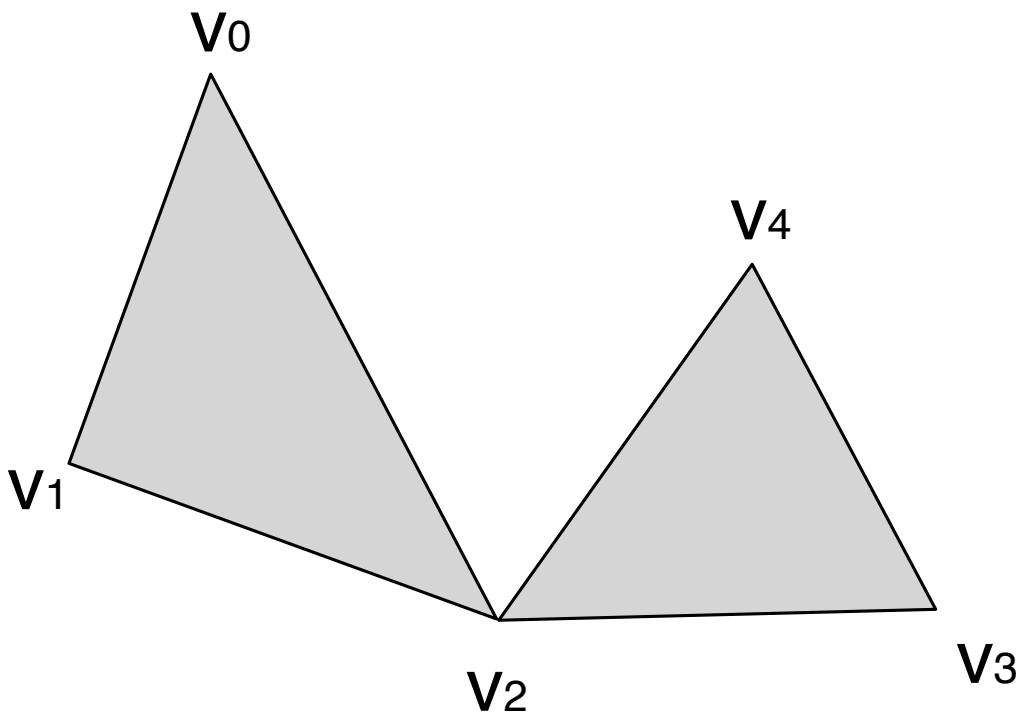
頂点 v_2 と v_3 は重複。無駄なメモリと通信。



drawElements()

配列バッファ | v_0 | v_1 | v_2 | v_3 | v_4 | と、

要素配列バッファ | 0 | 1 | 2 | 2 | 3 | 4 | を用意して gl.TRIANGLES で描く。(WebGLBuffer オブジェクトを二つ使う。)



gl.drawElements()

`void drawElements(GLenum mode, GLsizei count, GLenum type, GLintptr offset)`

mode は `gl.X: X={POINTS, LINES, LINE_LOOP, LINE_STRIP, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN}`

以下、ab を `gl.ELEMENT_ARRAY_BUFFER` にバインドされた要素配列バッファとすると、

count は ab に何個のインデックスがあるか

type は ab の要素インデックスの型。`gl.UNSIGNED_BYTE` または `gl.UNSIGNED_SHORT` のどちらか

offset は ab の中で実際に使うインデックスの開始位置（オフセット）

サンプルコード webgl_sample_triangle_02.html

```
function setupBuffers() {
  vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  var triangleVertices = [
        0.000000,  0.866025, 0.0,
       -0.500000,  0.000000, 0.0,
       -1.000000, -0.866025, 0.0,
        0.000000, -0.866025, 0.0,
        1.000000, -0.866025, 0.0,
        0.500000,  0.000000, 0.0
  ];
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(
      triangleVertices),
                gl.STATIC_DRAW);
  vertexBuffer.itemSize = 3;
```

サンプルコード webgl_sample_triangle_02.html

```
  indexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
  var indexNumbers = [
        0, 1, 5,
        1, 2, 3,
        3, 4, 5
  ];
  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(
      indexNumbers),
                gl.STATIC_DRAW);
  indexBuffer.size = 9;
}

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

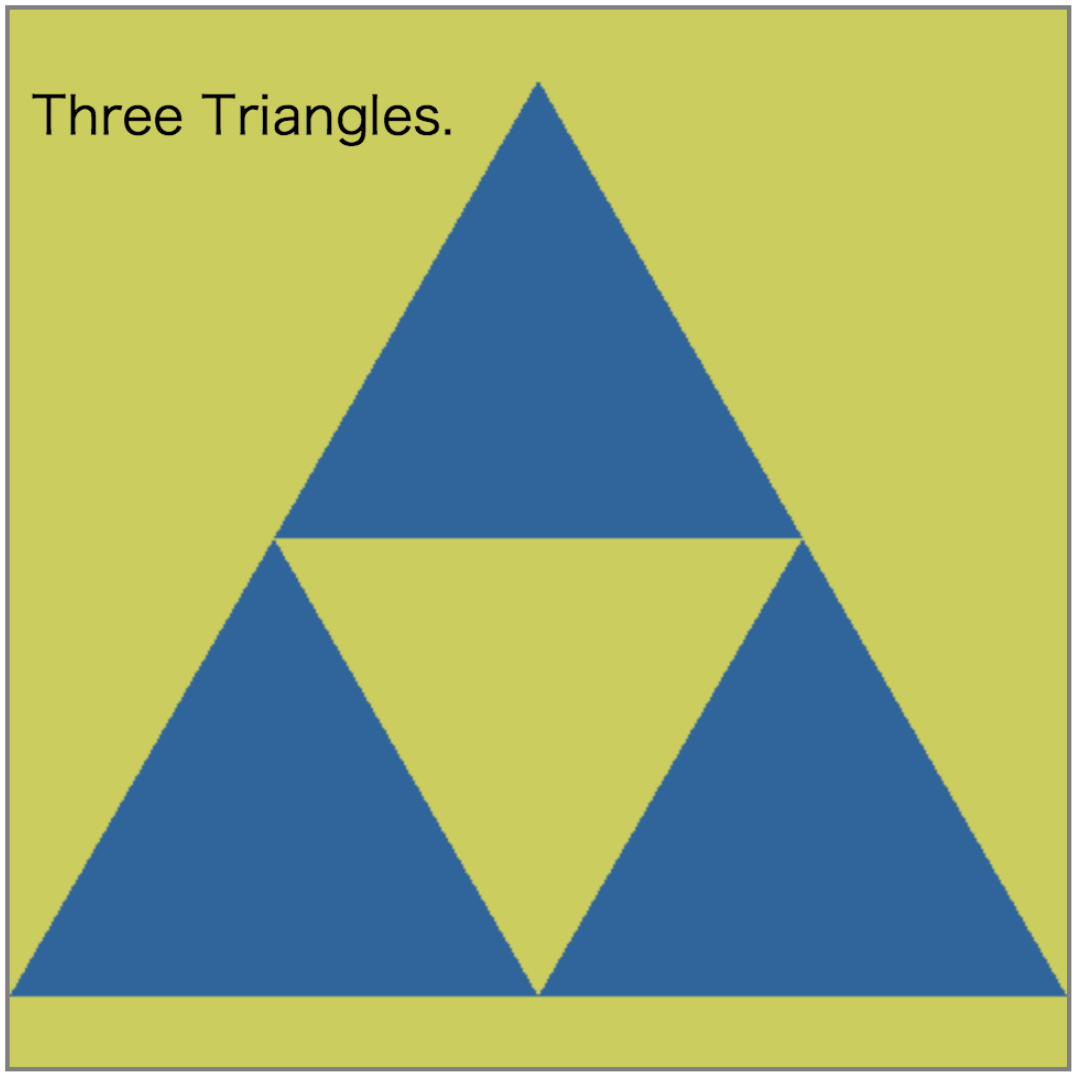
サンプルコード webgl_sample_triangle_02.html

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

    gl.drawElements(gl.TRIANGLES, indexBuffer.size,
                        gl.UNSIGNED_SHORT, 0);
}

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

webgl_sample_triangle_02.html の実行結果



演習

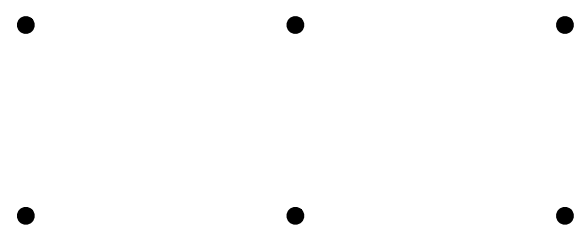
演習 00

webgl_sample_triangle_02.htmlを読んで

- 「三角形を描け」という指示がどこでなされているか見つけよう。
- 要素配列の index（頂点番号）の順番等を変えて、その効果を見よう。
- 三角形ではなく線分で図形を描いてみよう（ヒント：LINE_STRIP）。

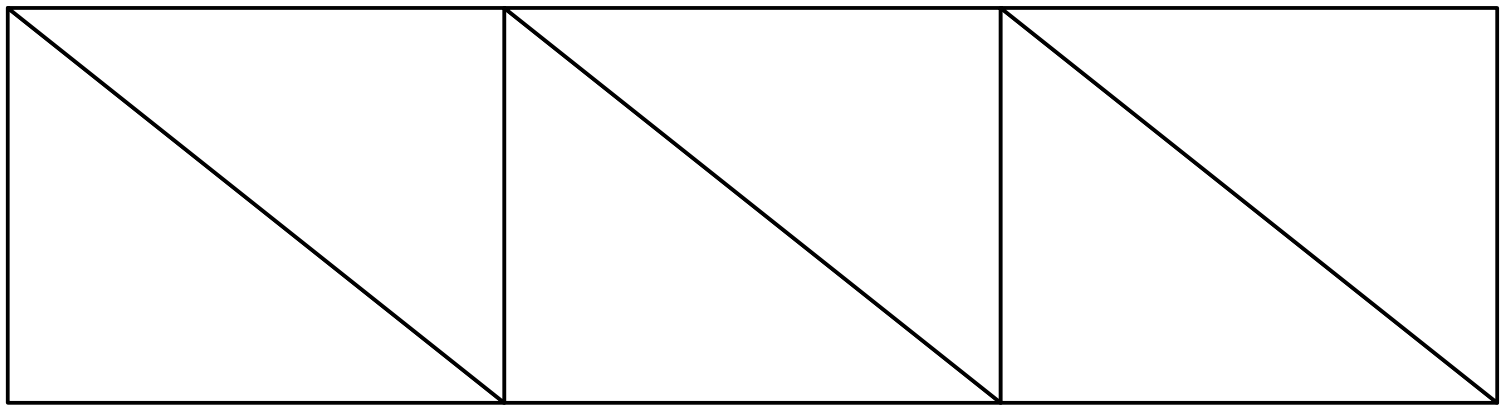
演習 01

drawArrays で次のような点列を描こう。



演習 02

drawArrays を使い、次のような図形を TRIANLGE_STRIP で描こう。



演習 03

今度は drawElements を使って同じ図を描こう。