

# WebGL によるデータ可視化入門\*

## 描画高速化のテクニック

陰山 聡

神戸大学 システム情報学研究科 計算科学専攻

2013.05.28

レポート講評

縮退三角形

型付き配列

演習 01

頂点属性のインターリーブ

演習 02

定数頂点データ

演習 03

索引

References

## レポート講評

- 6つの作品をウェブに掲載<sup>†</sup>
- 掲載は独創性と見た目を重視<sup>‡</sup>

---

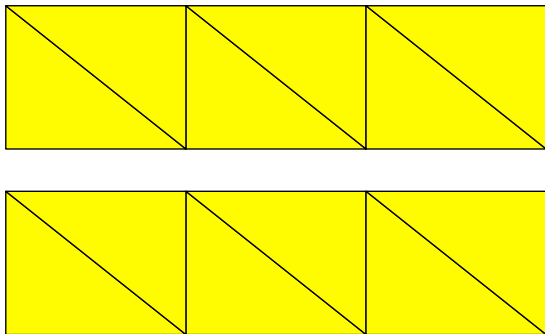
<sup>†</sup>一部改変したところがあります。

<sup>‡</sup>掲載されないレポートは点数が低いというわけではありません。

# 縮退三角形

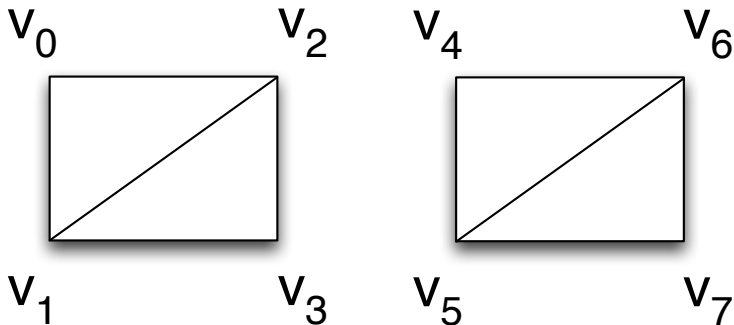
## 前回の演習課題の一つ

- 二つの長方形。



## 飛びのある TRIANGLE\_STRIP

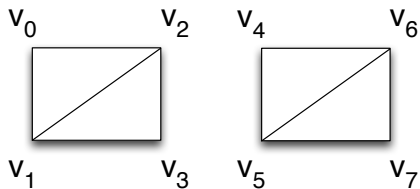
- `gl.drawArrays()` や `gl.drawElements()` を呼び出す回数は少ない方がいい
- 飛びのある TRIANGLE\_STRIP



## 縮退三角形

- `gl.TRIANGLE_STRIP` で連続していないストリップを結合する
- ジャンプする部分に「縮退三角形」をおく
- ダミーの頂点をおく
- 縮退三角形 = 面積ゼロの三角形      GPU が検出、自動的に破棄

## 赤がダミーの頂点



0	1	2	3	3	4	4	5	6	7
---	---	---	---	---	---	---	---	---	---

← (0,1,2) →

← (2,1,3) →

← (2,3,3) →

← (3,3,4) →

← (3,4,4) →

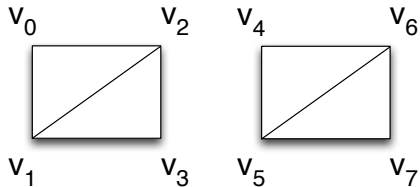
← (4,4,5) →

← (4,5,6) →

← (6,5,7) →



## 縮退三角形



← (0,1,2) →

← (2,1,3) →

← (2,3,3) →

← (3,3,4) →

← (3,4,4) →

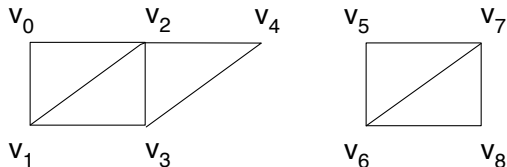
← (4,4,5) →

← (4,5,6) →

← (6,5,7) →

縮退三角形

## 最初のストリップに三角形が奇数個ある場合



0	1	2	3	4	4	5	5	6	7	8
---	---	---	---	---	---	---	---	---	---	---

要素配列バッファ

← (0,1,2) →

← (2,1,3) →

← (2,3,4) →

← (4,3,4) →

← (4,4,4) →

← (4,4,5) →

← (4,5,5) →

← (5,5,6) →

← (5,6,7) →

← (7,6,8) →

縮退三角形

# 型付き配列

# 型付き配列

- JavaScript はバイナリーデータの処理が不得意であった
- 型付き配列 ( typed array ) の導入

## 型付き配列

```
var buffer = new ArrayBuffer(8);
```

- 8バイトのバッファ
- バッファ内のデータを直接操作できない
- ビューの導入

```
var viewFloat32 = new Float32Array(buffer);
```

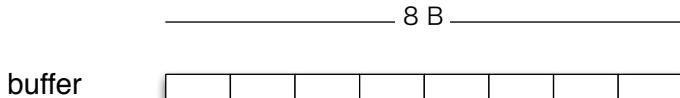
同じバッファに複数のビューを割り当てることができる。

```
var viewUint16 = new Uint16Array(buffer);
```

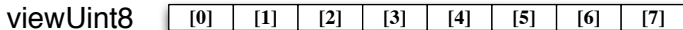
```
var viewUint8 = new Uint8Array(buffer);
```

## 型付き配列

バッファ



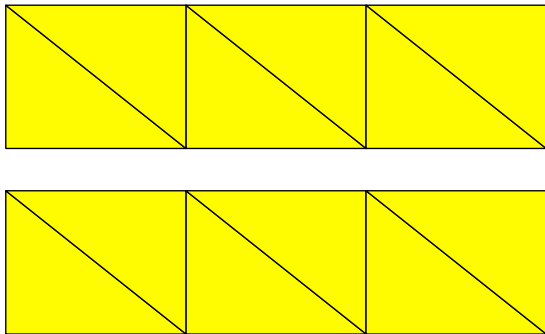
ビュー



# 演習 01

## 演習 01

- 二つの長方形を TRIANGLE\_STRIP 一回の呼び出しで描け。
- 面の色は任意。三角形の枠線は描かなくてもよい。





# 頂点属性のインターリーブ

## 頂点属性 ( attribute )

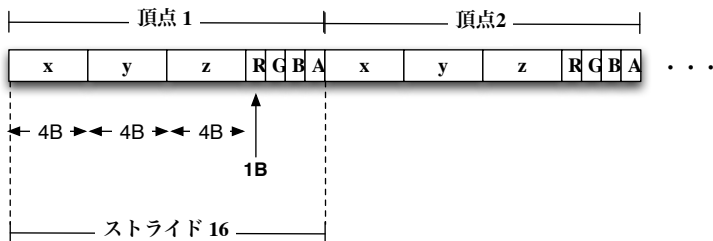
- 座標
- 色
- 法線ベクトル
- テクスチャ座標

必要な属性を頂点毎にまとめて一つの大きなバッファに納める<sup>§</sup>。

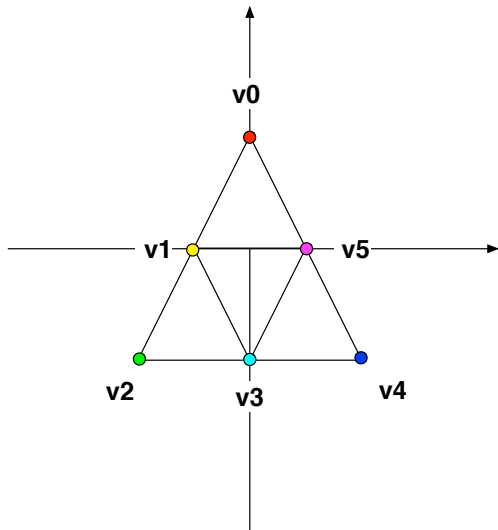
---

<sup>§</sup>それぞれの属性を別々のバッファに納める方法もあるが、その方法は遅い。

## データ (位置と色属性) のインターリーブ



## 例題



## webgl\_sample\_triangle\_03.html

## 頂点シェーダ

---

```
<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;
  attribute vec4 aVertexColor;
  varying vec4 vColor;

  void main() {
    vColor = aVertexColor;
    gl_Position = vec4(aVertexPosition, 1.0);
  }
</script>
```

## webgl\_sample\_triangle\_03.html

フラグメントシェーダ

---

```
<script id="shader-fs" type="x-shader/x-fragment">
  precision mediump float;

  varying vec4 vColor;
  void main() {
    gl_FragColor = vColor;
  }
</script>
```

## webgl\_sample\_triangle\_03.html

## シェーダのセットアップ

---

```
function setupShaders() {  
  
    var vertexShader = loadShaderFromDOM("shader-vs");  
    var fragmentShader = loadShaderFromDOM("shader-fs");  
  
    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");  
  
shaderProgram.vertexColorAttribute =  
    gl.getAttribLocation(shaderProgram, "aVertexColor");  
  
gl.enableVertexAttribArray(shaderProgram.  
    vertexPositionAttribute);  
gl.enableVertexAttribArray(shaderProgram.  
    vertexColorAttribute);  
}
```

---



## webgl\_sample\_triangle\_03.html

## JavaScript 配列でデータを用意

---

```
var triangleVertices = [  
  // ( x y z )( r g b a )  
  0.000000, 0.866025, 0.0, 255, 0, 0, 255, // red  
  -0.500000, 0.000000, 0.0, 255, 255, 0, 255, // yellow  
  -1.000000, -0.866025, 0.0, 0, 255, 0, 255, // green  
  0.000000, -0.866025, 0.0, 0, 255, 255, 255, // cyan  
  1.000000, -0.866025, 0.0, 0, 0, 255, 255, // blue  
  0.500000, 0.000000, 0.0, 255, 0, 255, 255 // magenda  
];
```

# webgl\_sample\_triangle\_03.html

あとは言葉で説明

## バッファのセットアップ

1. バッファのアロケート ( new ArrayBuffer )
2. バッファに Float32Array ビューをマップ ( 位置座標用 )
3. 同じバッファに Uint8Array ビューをマップ ( 色用 )
4. JavaScript 配列の値をバッファにロード

## webgl\_sample\_triangle\_03.html

( 続き )

## 要素バッファのセットアップ

1. 要素バッファのセットアップ
2. 要素バッファを ELEMENT\_ARRAY\_BUFFER にバインド
3. 要素バッファに要素のデータをロード

## webgl\_sample\_triangle\_03.html

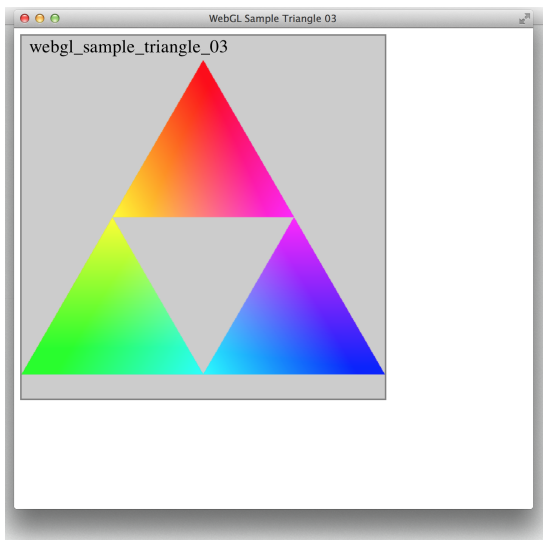
( 続き )

## 描画

1. バッファを ARRAY\_BUFFER にバインド ( `bindBuffer` )
2. バッファ中の位置データの配置を記述 ( `vertexAttribPointer` )
3. バッファ中の色データの配置を記述 ( `vertexAttribPointer` )
4. 描画 ( `drawElements` )

以上 ( 詳しくはソースコード参照 )

# webgl\_sample\_triangle\_03.html



# 演習 02

## 演習 02

- 頂点データのインターリーブの練習をしよう。
- 対象は任意。

# 定数頂点データ



## 定数頂点データ

複数の頂点で属性が共通な場合（例えば複数の頂点で色が同じなど）

定数頂点データ（constant vertex data）を指定する

方法は簡単

1. その属性の汎用属性インデックスを無効化する（`disableVertexAttribArray`）
2. 定数属性データを指定する（`vertexAttrib4f` 等）
  - `gl.vertexAttrib4f(index, v0, v1, v2, v3)`
  - `gl.vertexAttrib3f(index, v0, v1, v2)`
  - `gl.vertexAttrib2f(index, v0, v1)`
  - `gl.vertexAttrib1f(index, v0)`

## 頂点配列と定数頂点データが混在するとき

`gl.enableVertexAttribArray` がコールされている

頂点配列から読む

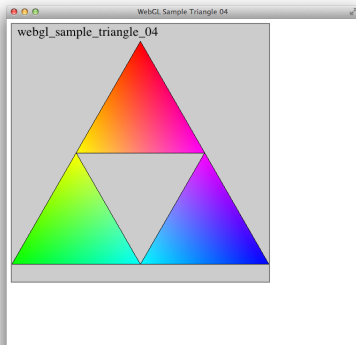
`gl.disableVertexAttribArray` がコールされている  
定された値を使う

`gl.vertexAttrib()` で設

## 例題 : webgl\_sample\_triangle\_04.html

面の色 : 頂点配列

線の色 : 定数頂点データ (黒)



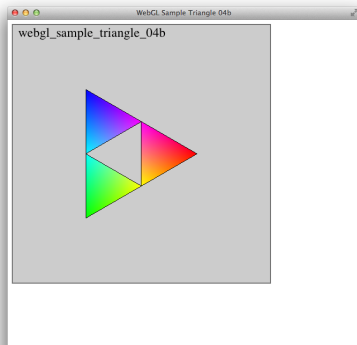
## 次回以降に備えてちょっと復習

頂点シェーダ内部での計算

webgl\_sample\_triangle\_04b.html

 $(x, y, z, 1) \implies (y/2, x/2, z, 1)$  (縮小して裏返し)

裏面を表示 (表の面をカリング)



# 演習 03

## 演習 03

- 定数頂点データの練習をしよう
- 対象は任意

# 索引

インターリーブ, 19

型付き配列, 12

定数頂点データ, 33

縮退三角形, 7

# References