

# 開発ツールと DOM API

2015 年度 情報可視化論

陰山 聡

2015.06.02

## 事務連絡

来週（6/9）もこの部屋（第1演習室）で行います。

以前、「6月9日だけは第2演習室で行う」と予告しましたが変更です。

# WebGL のコーディングスタイル

## (この講義での) 命名規則

- `canvas.getContext()` で取り出した `WebGLRenderingContext` は常に `gl` という変数に保存。
- `gl` はグローバル変数とする
- JavaScript もシェーダも camelCase
- シェーダの変数では、属性 (attribute) には a のプリフィックス。  
e.g., `aVertexPosition`
- シェーダの変数では、varying 変数には v のプリフィックス。 e.g.,  
`vColor`
- シェーダの変数では、uniform には u のプリフィックス。 e.g.,  
`uMVMMatrix`

# WebGL コードのデバッグ

## JavaScript コンソール

- それぞれのブラウザで「コンソール表示」
  - Safari の場合：開発 → エラーコンソールを表示
- エラーメッセージが表示される
- JavaScript の `console.log("文字列");` → 文字列をプリント
- “printf デバッグ” に使える

## 演習

Safari のエラーコンソールを開き、

- 自作 HTML ファイル中の JavaScript プログラムから `console.log()` 関数で文字列を書き出せ。

## WebGL 用のデバッグツール

- Chrome: Chrome デベロッパーツール
- Firefox: Firebug (Firefox のエクステンション)
- 各種ブラウザ: Web Inspector



# Safari Web Inspector

- Safari 用 Web 開発ツール
- OS X & iOS

# Safari Web Inspector

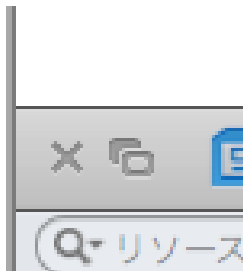
## 使い方

- WebGL アプリを含むページを表示
- メニュー：開発 → Web インスペクタを表示

ツールバーに置くと便利：

- メニュー：表示 → ツールバーをカスタマイズ... → Web インспекタのアイコンをツールバーにドラッグ

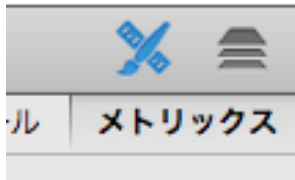
## Web Inspector パネルの detach



# Inspecting DOM

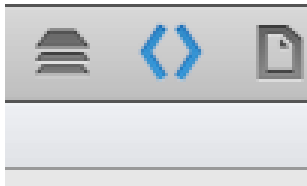
- DOM tree 表示
- DOM のノードにカーソルを置くと、対応するエレメントが色で表示
- ソースか DOM か切替可能

## サイズ表示



- たとえば、HTML の canvas にマウス → 配置サイズが表示

## DOM ノード情報



- 選択した HTML の要素 (DOM のノード) の情報

# Timelines

- ダウンロード時間の解析
- JavaScript のプロファイラ

## Timelines

時計のアイコン (タイムライン) をクリック

- ダウンロード時間の解析 & JavaScript のプロファイラ
- ネットワーク要求
- レイアウトとレンダリング
- JavaScript とイベント

時間計測法：

- プロファイル (左下) → 記録ボタン



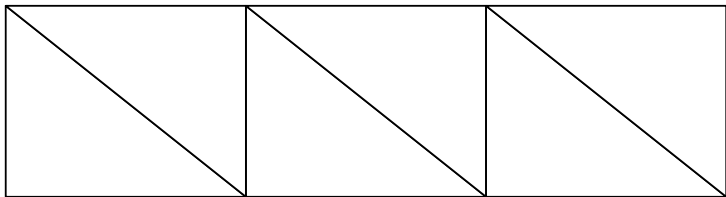
## デバッグ

- ブレークポイント (左下)
- コールスタック (左上)

# 演習

# 演習

- 先週の演習課題の続き。下の図（あるいは他でも OK）を drawElements と TRIANLGE\_STRIP を使って描け。
- ただし、今日は WebGL Inspector 等のツールも使ってみよう。



# DOM

# DOM とは

## Document Object Model (DOM)

- HTML や XML 文書にアクセスするための API
  - プログラム (スクリプト言語) から (構造をもつ) 文書にアクセスする
  - プログラム (スクリプト言語) から文書の構造、スタイル、内容を変更する

URL: <http://www.w3.org/DOM/DOMTR>

## DOM の例

サンプル：document\_object.html

Web Inspector（またはエディタ）でソースコードを見てみよう。

実行結果と見比べよう。

DOM ツリーを見てみよう。

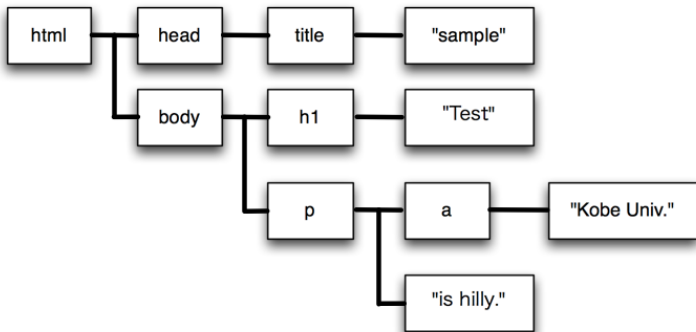
# DOM node tree

階層構造を持った文書 = ノードの木構造とみる

```
<html>
  <head>
    <title>sample</title>
  </head>
  <body>
    <h1>Test</h1>
    <p><a href="http://www.kobe-u.ac.jp/">Kobe Univ.</a>
      is hilly.
    </p>
  </body>
</html>
```

# DOM node tree

階層構造を持った文書 = ノードの木構造とみる



DOM は各ノードへのアクセス手段を提供する。



## ノードへの指定方法

- ツリーを root からたどる。例：  
`document.firstChild.childNodes[1].childNodes[1].childNodes[1].nodeValue`  
で “is hilly.” を取得。
- 要素名（タグ名）で指定する `getElementsByTagName()` メソッド
- 要素の id で指定する `getElementById()` メソッド

# DOM を使うための準備

特になし

ブラウザに組み込まれている

## 例：

サンプルコード `dom_sample_00.html`

Web Inspector のソース表示（またはエディタ）でソースを見てみよう。

Web Inspector で DOM ツリーを表示してみよう。

# DOM を使ったサンプルプログラム： dom\_sample\_00.html

---

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>DOM Sample 00</title>
<meta charset="utf-8">
<script type="text/javascript">
window.onload = function() {
    var paragraphs = document.getElementsByTagName("p");
    var p00 = paragraphs[0].textContent; // 1st paragraph
    var p01 = paragraphs[1].textContent; // 2nd paragraph
    alert(p01 + p00);
    document.getElementById("id000").style.color = "red";
    document.getElementById("id000").textContent = "Here it is!"
    " ;
}

```

```
</head>
```

```
<body>
```

```
<h2> Sample HTML </h2>
```

```
<p> Hello. This is the 1st paragraph. </p>
```

```
<p> Hi. I'm 2nd paragraph. </p>
```

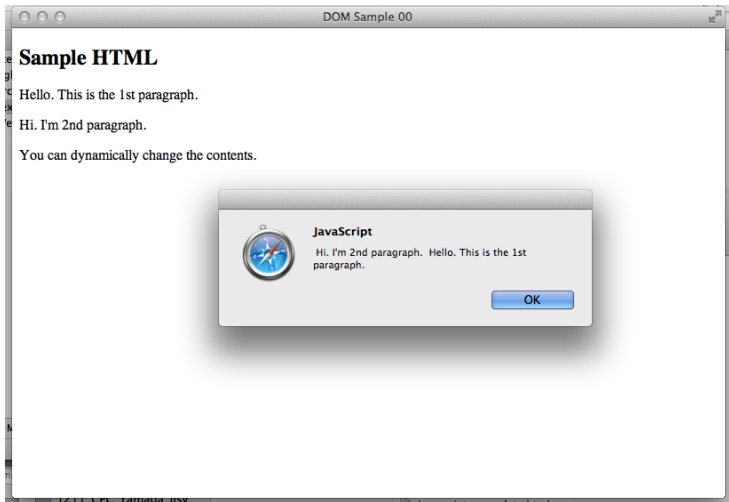
```
<div id="id000"> You can dynamically change the contents. </div>
```

```
</body>
```

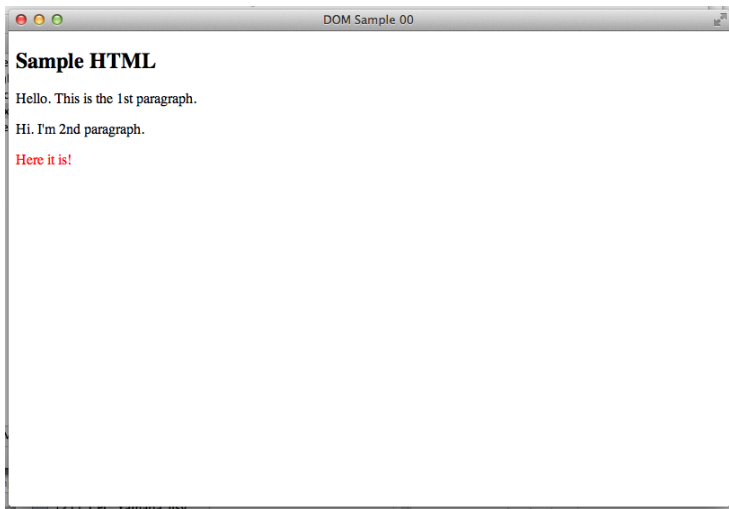
```
</html>
```

---

# 読み込み結果



# 読み込み結果



## DOM を使ってシェーダソースコードをロードする

- これまでのサンプルプログラムでは、シェーダソースコードは JavaScript の文字列変数として直接書いていた。
- DOM API を使えばもっと読みやすくなる
- HTML の script タグとして書く：
  - 頂点シェーダ：

```
<script id="shader-vs" type="x-shader/x-vertex">
```
  - フラグメントシェーダ：

```
<script id="shader-fs" type="x-shader/x-fragment">
```
- この二つの script type をブラウザは知らないので無視する。
- [注意] HTML のヘッダに書かずにファイルから読み出す方法もある (後述)。



## webgl\_sample\_triangle\_02\_shader\_from\_DOM.html

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

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

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

  void main() {
    gl_FragColor = vec4(0.2, 0.4, 0.6, 1.0);
  }
</script>
```

## あとはDOMを使って読み込めばいい

```
function loadShaderFromDOM(id) {  
    var shaderScript = document.getElementById(id);  
  
    if (!shaderScript) {  
        return null;  
    }  
  
    var shaderSource = "";  
    var currentChild = shaderScript.firstChild;  
    while (currentChild) {  
        if (currentChild.nodeType === 3) { // 3 <= TEXT_NODE  
            shaderSource += currentChild.textContent;  
        }  
        currentChild = currentChild.nextSibling;  
    }  
}
```

```
var shader;
if (shaderScript.type === "x-shader/x-fragment") {
    shader = gl.createShader(gl.FRAGMENT_SHADER);
} else if (shaderScript.type === "x-shader/x-vertex") {
    shader = gl.createShader(gl.VERTEX_SHADER);
} else {
    return null;
}

gl.shaderSource(shader, shaderSource);
gl.compileShader(shader);

if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    alert(gl.getShaderInfoLog(shader));
}
```

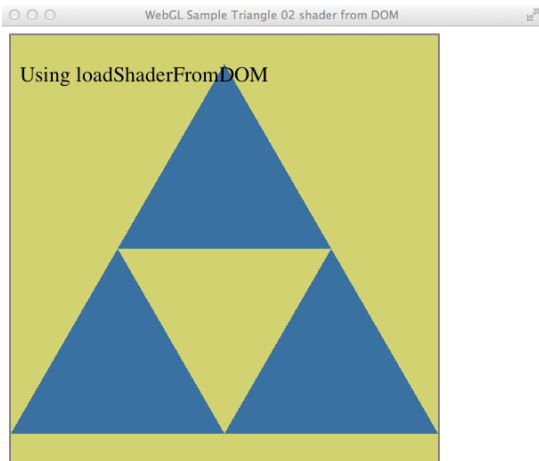
```
    return null;  
  }  
  return shader;  
}
```

```
function setupShaders() {
```

```
  var vertexShader = loadShaderFromDOM("shader-vs");  
  var fragmentShader = loadShaderFromDOM("shader-fs");
```

---

# 実行結果



# レポート課題

- WebGL で好きな図を描け。
- ただしシェーダプログラムは DOM API を使ってロードすること。
- **アーカイブにはしない。**
- 提出はメールで。添付ファイルは2つ
  1. レポートの PDF ファイル： **ファイル名：report\_02.pdf**
  2. 作成した HTML ファイル： **ファイル名：report\_02.html**
    - ファイル名中のアンダースコア ( \_ ) は半角
    - ファイル拡張子は html とし、htm としない
- 提出先：kageyama.lecture@gmail.com
- メールのタイトル： **情報可視化論 レポート 2**
- レポートの PDF ファイルは1ページのみ。(表紙をつけない。)
- レポートの PDF ファイルには以下の4点を記述する。
  - 1 学籍番号と氏名
  - 2 ウェブ公開時の名前 (本名またはイニシャル)
  - 3 どのような図形を描いたか
  - 4 描いた図形のキャプチャ図
- 締め切り：6/9 (火) 07:00