福山平成大学経営学部紀要
第 15 号（2019），157－164 頁

WebGL による 2 次元幾何アニメーション表示の高速化

尾崎 誠

福山平成大学経営学部経営学科

要旨：筆者は HTML5 および Javascript による 2 次元幾何アニメーション表示を行うプログラムを作成している。その描画には汎用性を重視し Canvas を用いている。しかし、3 次元幾何アニメーション表示への拡張や、シミュレーション表示プログラムの作成を考慮すると、より高速に描画が可能な WebGL への描画方法の変更が必要となった。そのため、プログラムの描画部分を Canvas から WebGL への変更を行ったため、それについて報告する。

キーワード：2 次元アニメーション、Canvas、WebGL

1. はじめに

筆者は教育分野での利用を目的に、ブラウザ上で動作する2次元幾何アニメーション表示を行うプログラムをHTML5およびJavascriptを用いて開発してきた[1]。その際、図形の描画には汎用性を重視してCanvasを用いていた。これは、旧世代のiOSや一部のデスクトップ、ノートパソコンが、Canvasよりも高速描画が可能なWebGLに対応していなかったためである。しかし、汎用性を重視したCanvasでの描画速度はWebGLでの描画速度よりもかなり遅く、2次元幾何アニメーション表示を3次元に拡張すると、同時に表示する図形の数が多くなった時に描画が追い付かず、コマ落ちが起きる可能性が高かった。さらに、シミュレーション表示を行う場合には、計算処理にも膨大なCPU処理が必要となり、描画速度が追い付かない可能性は非常に高いことは明らかである。

そこで今回は、既にiOSでもWebGLに対応しており、またAndroidや多くのデスクトップパソコンやノートパソコンもWebGLに対応していることから、図形の描画をCanvasからWebGLに変更することにした。

そのため、図形を描画する関数の中身を大幅に変更する必要があり、今回は基本的な図形を描画する関数のみWebGLに対応したプログラムに変更した。ここでは、それについて報告する

2. Canvas による図形の描画方法について

Canvas では、基本的な図形を描画する命令が準備されており、その命令を利用して図形の描画を行う。例としていくつかの基本的な図形を描画する命令群を以下に紹介する。

2.1 直線の描画

Canvas では、直線の描画は以下のような命令群により行う。

context.strokeStyle = ‘#000000’; 直線の色の指定
context,linewidth = 1; 直線の幅の指定
context.lineCap = ‘square’; 直線の終端処理
context.beginPath(); 直線の描画開始
context.moveTo(-1, 0); 直線の描画開始座標の移動
context.lineTo(1, 0); 直線の描画終了座標の指定
context.stroke(); 直線の描画開始座標から描画終了座標まで描画

以上の命令を実行すると、図 2.1 のように色が黒で線の幅 1 ピクセル、終端の形が四角となる直線を、(-1, 0)から(1, 0)まで描画する。Canvas での命令群の変数を外部から与える connect 関数を作成し、直線を描画する際には connect 関数に必要な変数を与えて描画している。

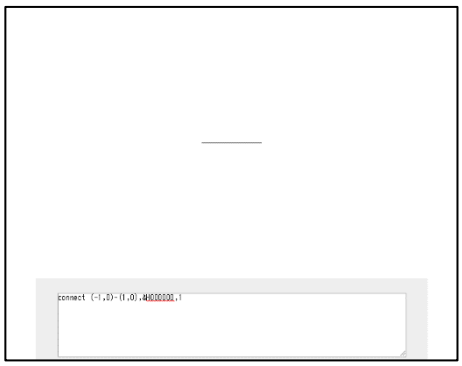

図 2.1 直線の描画

2.2 円の描画

Canvas では、円の描画は以下のような命令群で行う。
context.fillStyle = ‘#000000’; 塗りつぶしの色の指定
context.strokeStyle = ‘#000000’; 線の色の指定
context.lineWidth = 1; 幅の指定
context,arc(0, 0, 2, (Math.PI/180)*0, (Math.PI/180)*360, false); (中心の x 座標, 中心の y 座標, 半径, 円の描画開始角度[ラジアン], 円の描画終了角度[ラジアン], 円の描画方向[true 反時計回り, false 時計回り])
context.stroke(); 円の描画

以上の命令を実行すると、図 2.2 のように塗りつぶしの色が黒、線の色が黒で幅が 1 ピクセル、中心座標が(0,0)、円の半径が 2 の円が描画される。それぞれの変数を外部から与える circle 関数を作成し、円を描く際には circle 関数に必要な変数を与えて描画している。

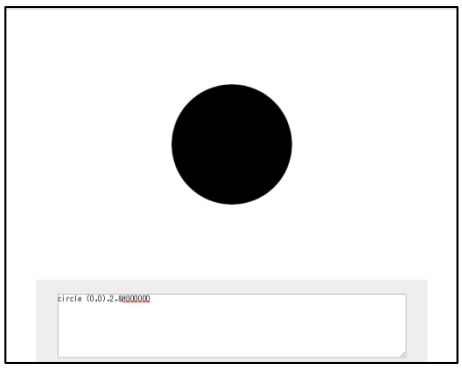

図 2.2 円の描画

2.3 矩形の描画

Canvas では、矩形の描画は以下のような命令群で行う。

context.fillStyle = '#000000'; 塗りつぶしの色の指定

context.strokeStyle = '#000000'; 線の色の指定

context.lineWidth = 1; 線の幅

contex.fillRect(0, 0, 2, 1); (矩形の中心の x 座標, 矩形の中心の y 座標, 矩形の幅, 矩形の高さ)

以上の命令を実行すると、図 2.3 のように塗りつぶしの色が黒、線の色が黒、線の幅が 1 ピクセル、中心座標が(0, 0)、幅 2 ピクセル、高さ 1 ピクセルの矩形が描画される。それぞれの変数を外部から与える box 関数を作成し、矩形を描く際には box 関数に必要な変数を与えて描画している。

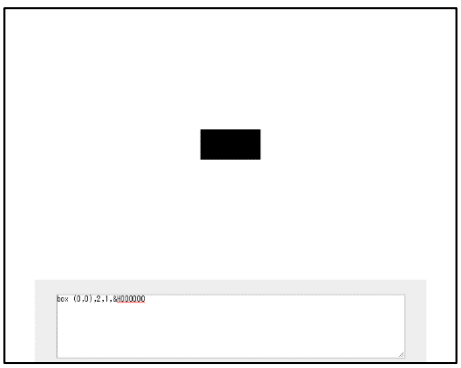

図 2.3 矩形の描画

3. WebGL による図形の描画方法について

WebGL では、Canvas に比べて非常に高速で図形の描画を行うことができる。しかし、Canvas とは違い基本的な図形を描画する関数は用意されていない。WebGL で用意されている命令は、点、線、三角形しかない。そのため、基本的な図形を描画するには、三角形を組み合わせて描画する必要がある。また、Canvas とは使用できる命令も違うため、基本的な図形を描画する関数の中身も大幅に変更する必要がある。

さらに Canvas と WebGL では、座標の表現法が異なる。Canvas では、描画領域の左上の座標が(0, 0)となり、描画領域の幅、高さは指定したピクセルとなる。つまり、描画領域の幅が 640、高さが 480 であれば、描画領域の右下の座標は(639, 479)となる。しかし WebGL では、常に描画領域の中心座標が(0 ,0)となり、描画領域の左上の座標は(-1, -1)、描画領域の右下の座標は(1, 1)に固定される。そのため、実際に描画領域に図形の描画を行う実座標から、WebGL での座標に変換する必要もある。

3.1 直線の描画

WebGL では、直線の描画は以下のような命令群により行う。

```
// 画面を消去する色などの初期化
gl.clearColor(1.0, 1.0, 1.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);
// 空バッファオブジェクトの生成
var buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
// 頂点シェーダー
var vSource = [
      "precision mediump float;",
      "attribute vec2 vertex;",
      "void main(void) {",
            "gl_Position = vec4(vertex, 0.0, 1.0);",
      "}"
].join("¥n");

var vShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vShader, vSource);
gl.compileShader(vShader);
gl.getShaderParameter(vShader, gl.COMPILE_STATUS);
// フラグメントシェーダー
var rgba = [0.0, 0.0, 0.0, 1.0]; // Red, Green, Blue, Alpha
var fSource = [
```

```
    "precision mediump float;",
    "void main(void) {",
        "gl_FragColor = vec4("+ rgba.join(",") +");",
    "}"
].join("¥n");

var fShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fShader, fSource);
gl.compileShader(fShader);
gl.getShaderParameter(fShader, gl.COMPILE_STATUS);
// プログラムオブジェクトの生成
var program = gl.createProgram();
gl.attachShader(program, vShader);
gl.attachShader(program, fShader);
gl.linkProgram(program);
gl.getProgramParameter(program, gl.LINK_STATUS);
gl.useProgram(program);
// シェーダー側の変数を Javascript 側での受け取り
var vertex = gl.getAttribLocation(program, "vertex");
gl.enableVertexAttribArray(vertex);
gl.vertexAttribPointer(vertex, 2, gl.FLOAT, false, 0, 0);
// 座標のセット
var vertices = [
    -0.125, 0,
    0.125, 0
];
// 直線の描画
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.DYNAMIC_DRAW);
gl.drawArrays(gl.LINE_STRIP, 0, vertices.length/2);
```

以上の命令を実行すると Canvas での描画と同じく、図 3.1 のように色が黒で線の幅 1 ピクセル、終端の形が四角となる直線を、(-1, 0)から(1, 0)まで描画される。このような、直線を描画するのに必要な命令群を、Canvas で描画する際と同様に connect 関数とし、外部から必要な変数を与えて直線が描画できるようにしている。

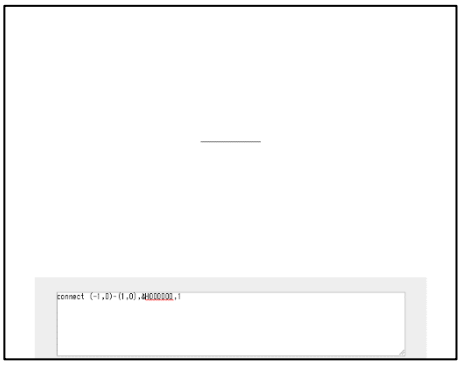

図 3.1 直線の描画

3.2 三角形の描画

WebGL では、図 3.2 のように頂点座標を順番に指定し、その間を直線で繋ぐことにより三角形の描画を行う。矩形や円、多角形のような図形は、全て三角形を組み合わせたポリゴン表現により描画が行われる。

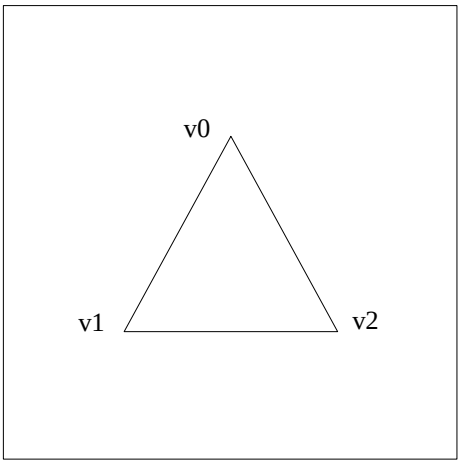

図 3.2 三角形の描画

WebGL で三角形を描画する命令群は、直線を描画する命令群での「画面を消去する色などの初期化」から「シェーダー側の変数を Javascript 側での受け取り」の部分までは、ほぼ共通のため割愛する。「座標のセット」から「三角形の描画」までの命令群は、以下の通りである。

```
// 座標のセット
var vertices = [
    0, -0.5,
    -0.5, 0.5
    0.5, 0.5
];
// 三角形の描画
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.DYNAMIC_DRAW);
gl.drawArrays(gl.TRIANGLES, 0, vertices.length/3);
```

WebGL では、矩形などの多角形は三角形を組み合わせたポリゴン表現により描画を行うこととなる。そのため、多角形を表示する関数では、三角形を表示する関数を繰り返し呼び出して処理を行うことで表示している。例えば、矩形を表示する際には図 3.3 のように三角形を 2 枚組み合わせることで表示している。図 3.3 では、分かりやすいように三角形をそのまま表示しているが、実際には矩形として表示される。

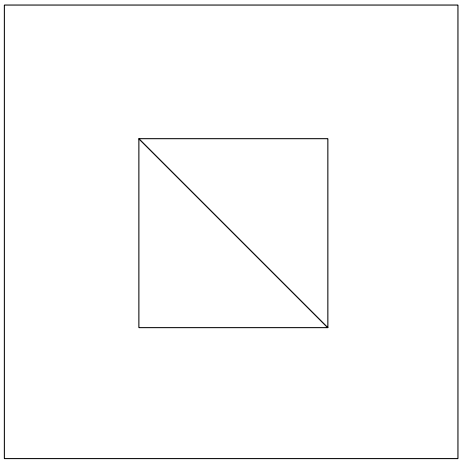

図 3.3 矩形の表示

4. おわりに

今回、2 次元幾何アニメーション表示プログラムの描画方法を、Canvas から WebGL に変更することにより描画速度を向上させた。しかし、まだ基本的な図形を描画する関数しかプログラム変更が終わっておらず、引き続き書き換え作業を行っていく必要がある。また、その作業が終わり次第、3 次元幾何アニメーション表示への拡張を行っていく予定である。

WebGL は、今後改定され WebGL2 となる予定であり、将来的には再度のプログラム変更が必要となる可能性がある。そのため、今後の作業負担を考慮し、Three.js などの WebGL 描画用のライブラリの利用も検討する必要がある。特に、3 次元幾何アニメーション表示に拡張する際には熟慮する必要がある。

参考文献

［1］ 尾崎誠、「HTML5 および Javascript による幾何アニメーション表示」、福山平成大学経営研究 （2017）