WebGL によるデータ可視化入門*

射影変換とモデルビュー変換

陰山 聡

神戸大学 システム情報学研究科 計算科学専攻

2013.06.11

*情報可視化論 X021（2013 年前期） LR301 →演習室

射影変換

クリップ座標

これまではクリップ座標の立方体の中だけで描画していた。つまりビューボリュームは正規化ビューボリューム

$$[-1,+1]\times[-1,+1]\times[-1,+1]$$

で固定であった。これは不便。

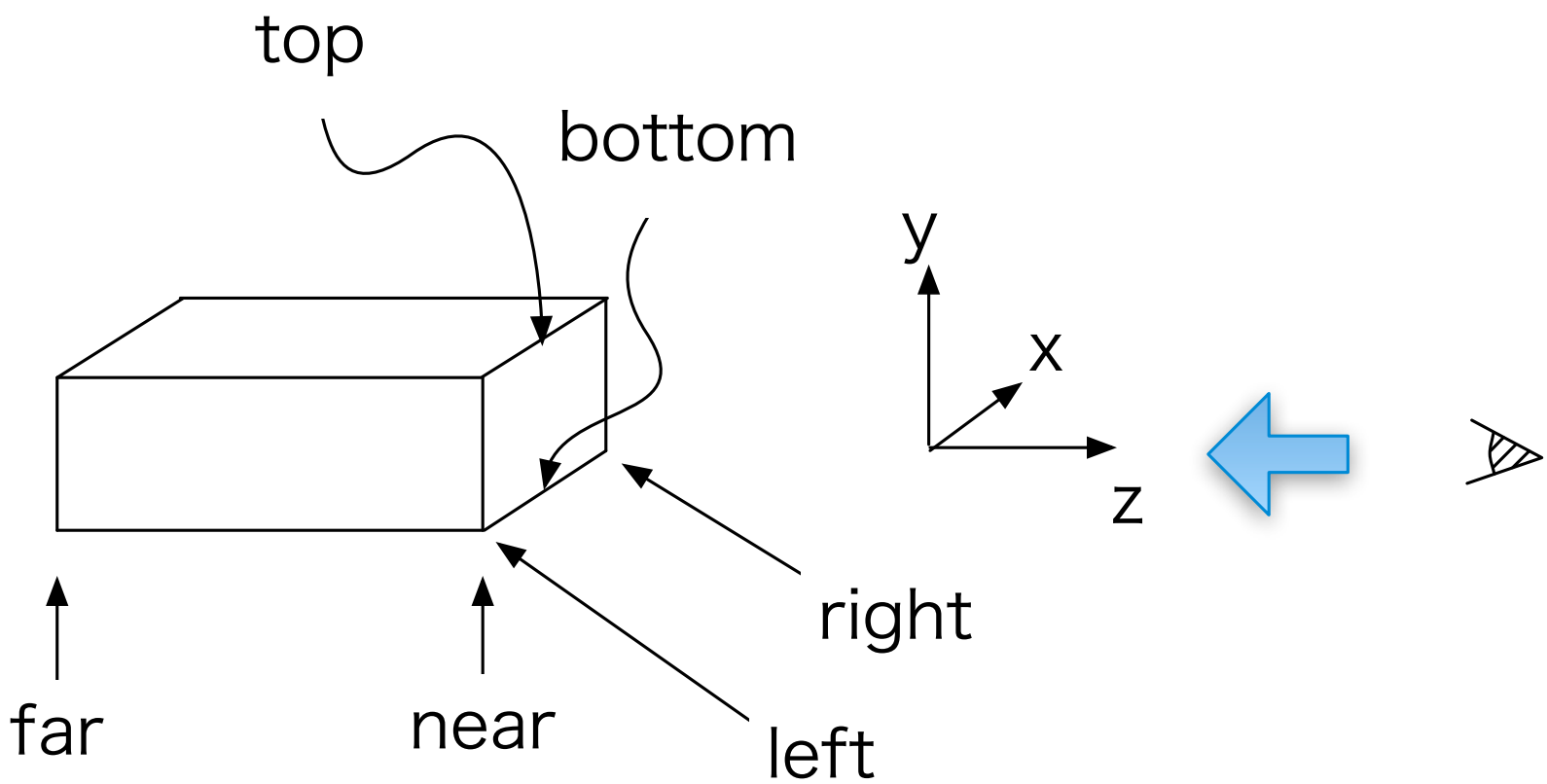
長さの単位（大きさ）を気にしないで、つまりワールド座標で物体を自由に定義したい。

射影変換

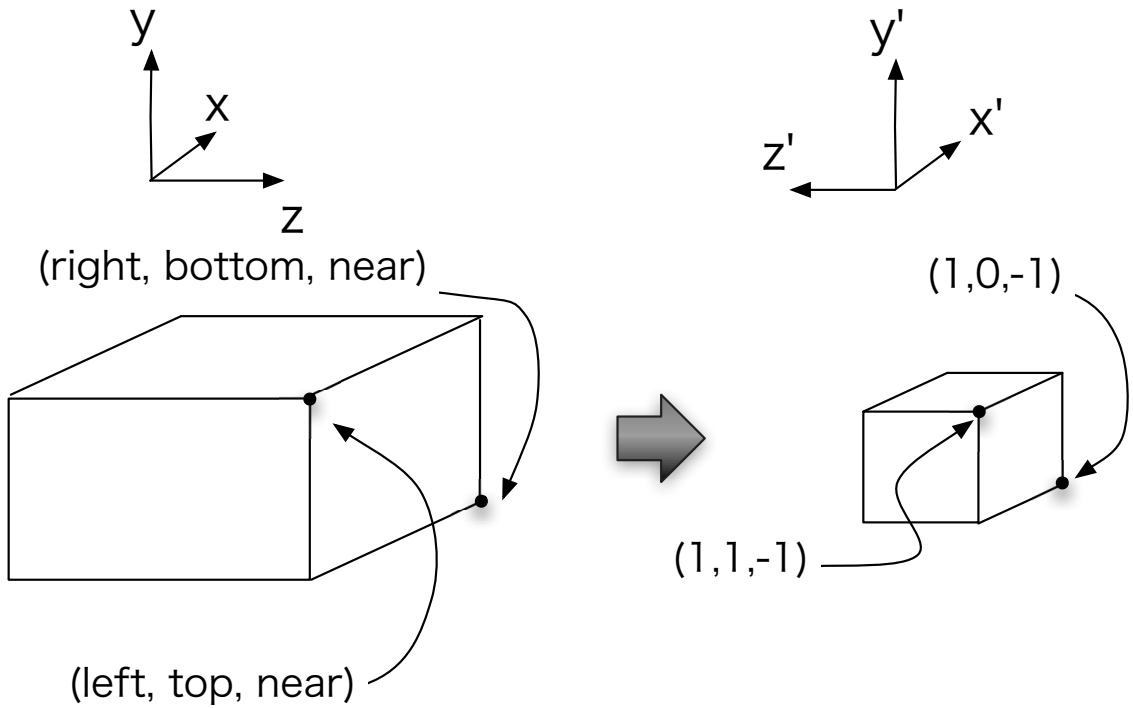
- 正射影
- 透視射影

正射影

- カメラは $z = \infty$ に位置し、$-z$ 方向を向いている。
- 正規化ビューボリュームの座標系は左手系。



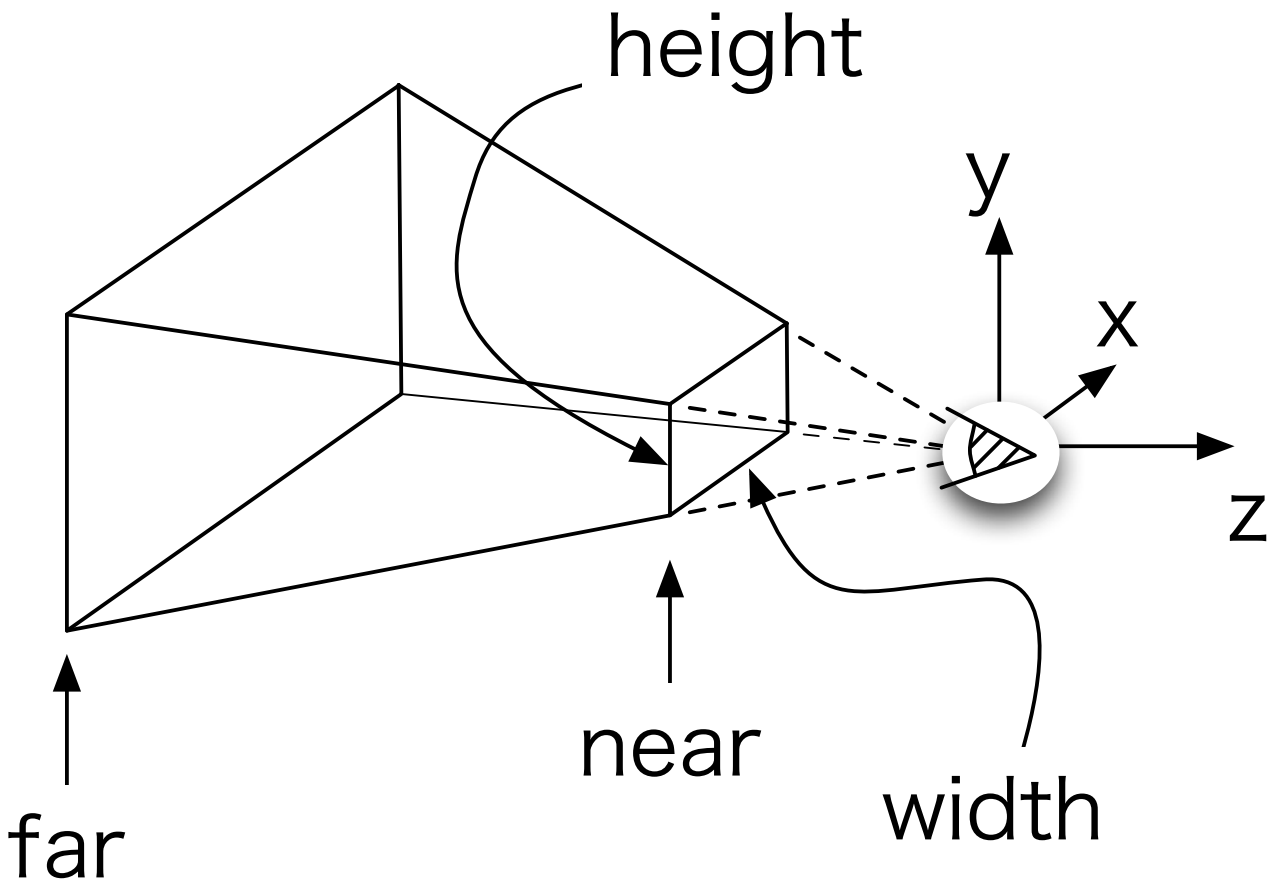
正射影



- 直方体から立方体（正規化ビューボリューム）への単純なスケール変換行列。
- 自分で行列を設定するのも簡単だが、次の関数が用意されている
- mat4.ortho(left, right, bottom, top, near, far, projectionMatrix);

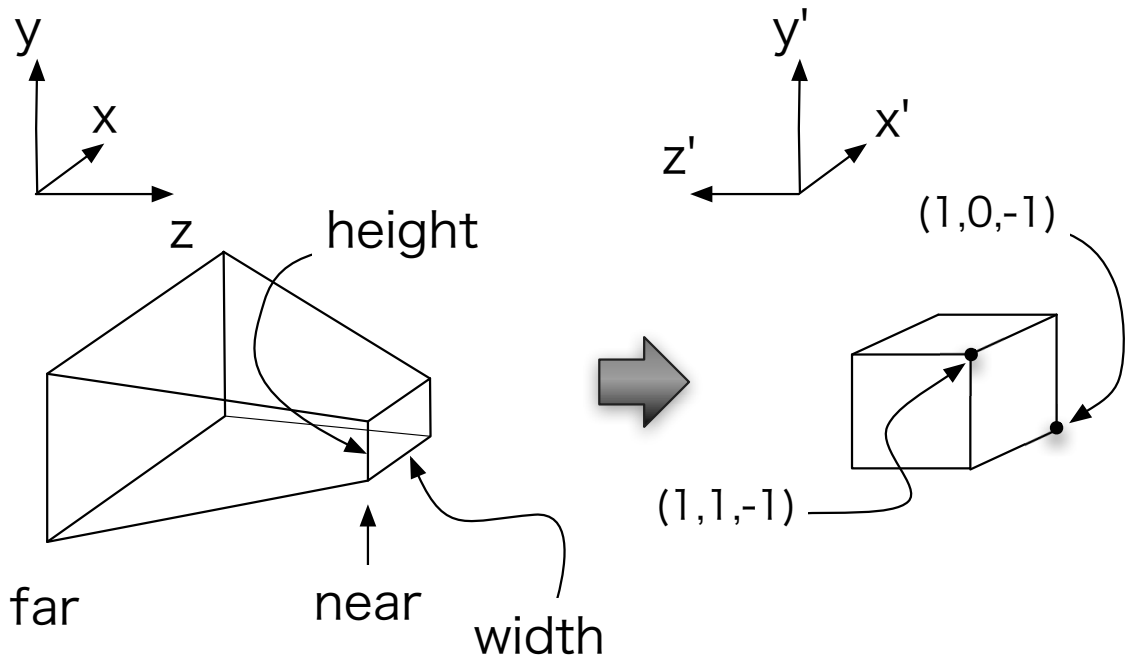
透視射影

- カメラは原点に位置し、$-z$ 方向を向いている†。



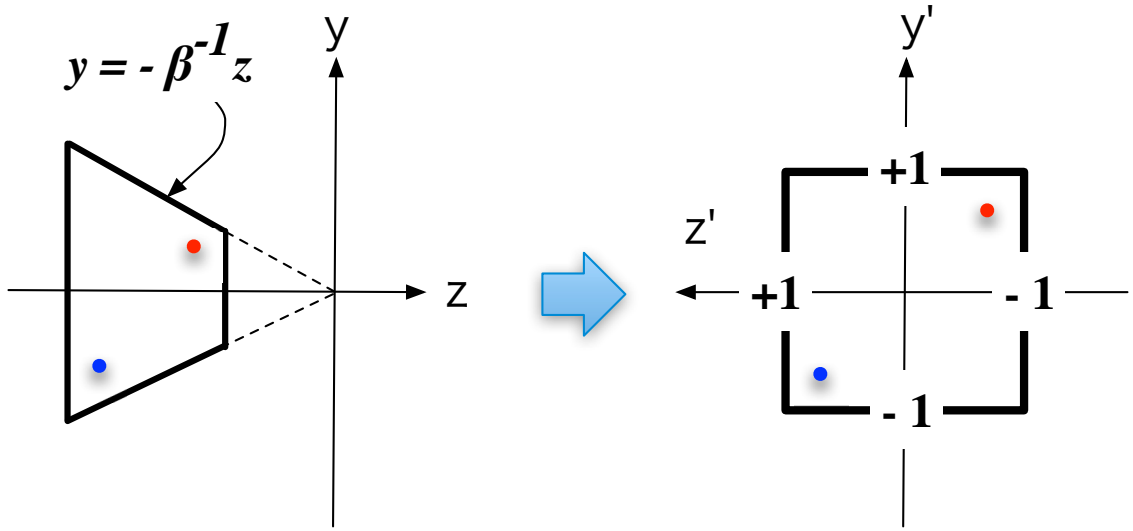
†物体を回転・移動させればいつでも $-z$ 方向に見えるようにできる。

透視射影



- 視錐台形から立方体への変換。

透視射影



$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\beta\, x/z \\ -\beta\, y/z \\ c_1\, z + c_2 \end{pmatrix}$$

$\beta > 0, c_1, c_2$ は定数。x と y に関しては $-\beta/z$ 倍のスケール変換‡。

‡このスケール係数は、たとえば、視錐台の上面 $y = -\beta^{-1}\, z$ が $y' = +1$ に変換されることで容易に確認できる。

(復習)

平行移動変換

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \end{pmatrix}$$

は同次座標を使うと行列演算で書けた。

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

では、透視射影

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\beta\, x/z \\ -\beta\, y/z \\ c_1\, z + c_2 \end{pmatrix}$$

も工夫すれば行列で書けるか？

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

では、透視射影

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\beta\, x/z \\ -\beta\, y/z \\ c_1\, z + c_2 \end{pmatrix}$$

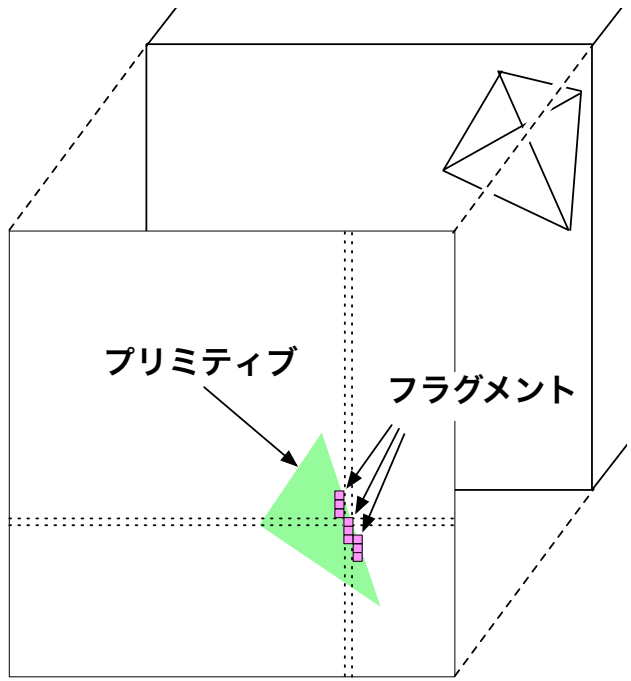
も工夫すれば行列で書けるか？

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

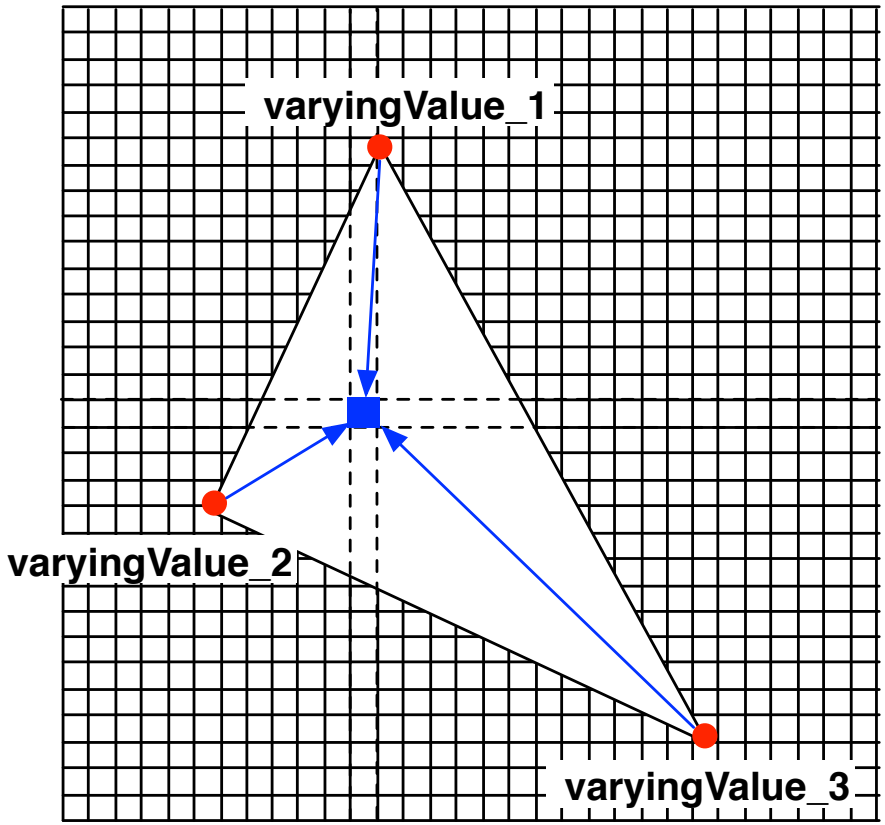
••• できない。

z 補間の問題

- 透視射影の場合、もう一つ注意しなければいけない問題がある。
- 復習：フラグメントシェーダに渡される時の varying 値の自動線形補間

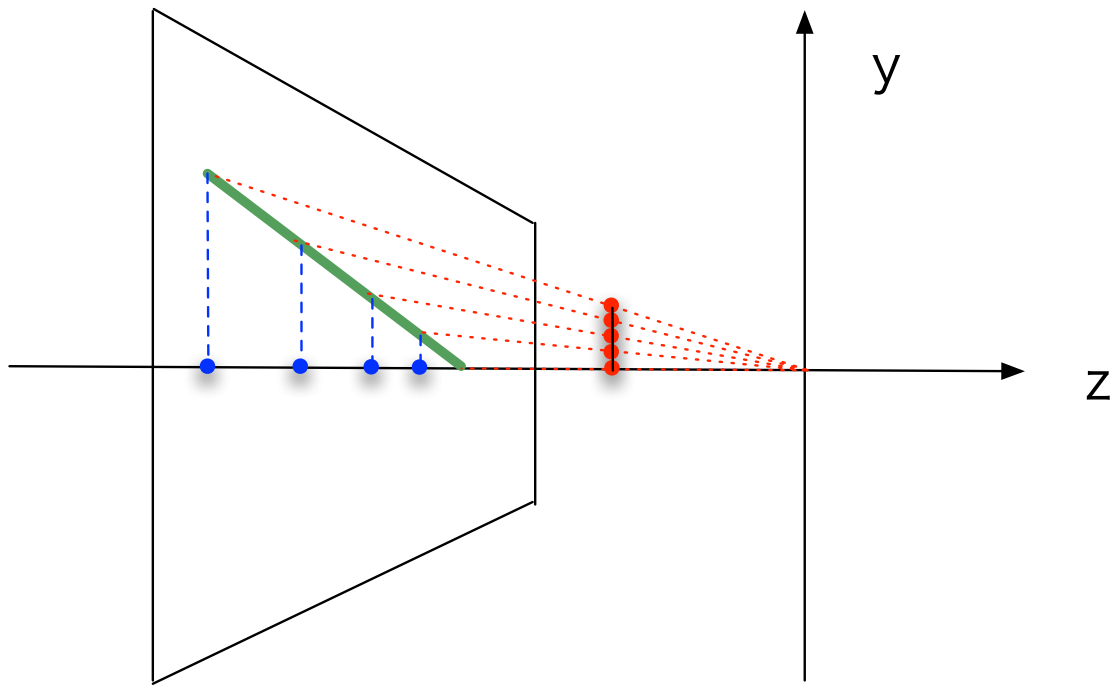


z 補間の問題

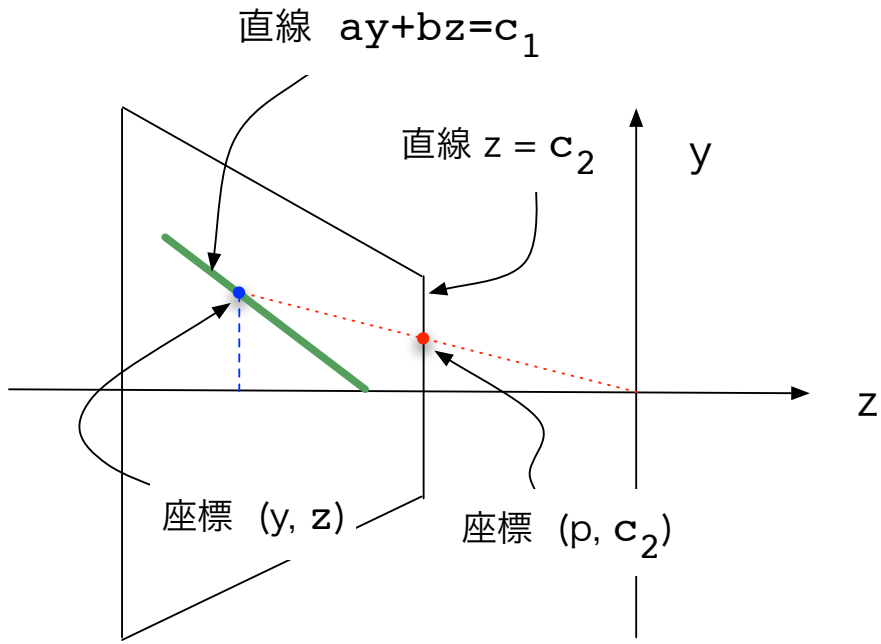


z 補間の問題

- ピクセル（赤）が等間隔に並んでいても
- プリミティブ（緑）上の補間点（青）の z 値は非等間隔。
- これは都合が悪い。



等間隔ピクセル p は等間隔の $1/z$ に対応



$$(ap/c_2 + b)z = c_1$$

$$\frac{1}{z} = \frac{a}{c_2\, c_1} p + \frac{b}{c_1}$$

そこで z の代わりに

$$z \Rightarrow z' = -\alpha - \gamma/z$$

という座標変換[§]して z' をフラグメントシェーダ（プリミティブ合成ステージ）に渡せば、z 方向にも自然に等間隔の線形補間になる[¶]。

—

そういえば ··· 透視射影の x, y 成分も z の除算が入っていた：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -\beta\, x/z \\ -\beta\, y/z \end{pmatrix}$$

[§]負の z に対して、$\gamma > 0$ なら z' は z の単調増加関数。
[¶]$-1 \le z' \le +1$ となるように α と γ を調整する。

そこで

ある点 $(x, y, z)^t$ に対して、

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\beta\, x/z \\ -\beta\, y/z \\ -\alpha - \gamma/z \end{pmatrix}$$

をプリミティブ組み立てステージに渡せばよい[‖]。これを OpenGL では、次の 2 段階に分けて実行する。途中で目標の $(x', y', z')^t$ の z 倍の座標を経由する。

$$\begin{pmatrix} x^\dagger \\ y^\dagger \\ z^\dagger \end{pmatrix} := \begin{pmatrix} zx' \\ zy' \\ zz' \end{pmatrix} = \begin{pmatrix} -\beta\, x \\ -\beta\, y \\ -\alpha\, z - \gamma \end{pmatrix}$$

[‖] 3 成分全てに z の除算が入っていることに注意。

透視射影変換の二つのステップ

(1) 普通の同次座標の行列演算**

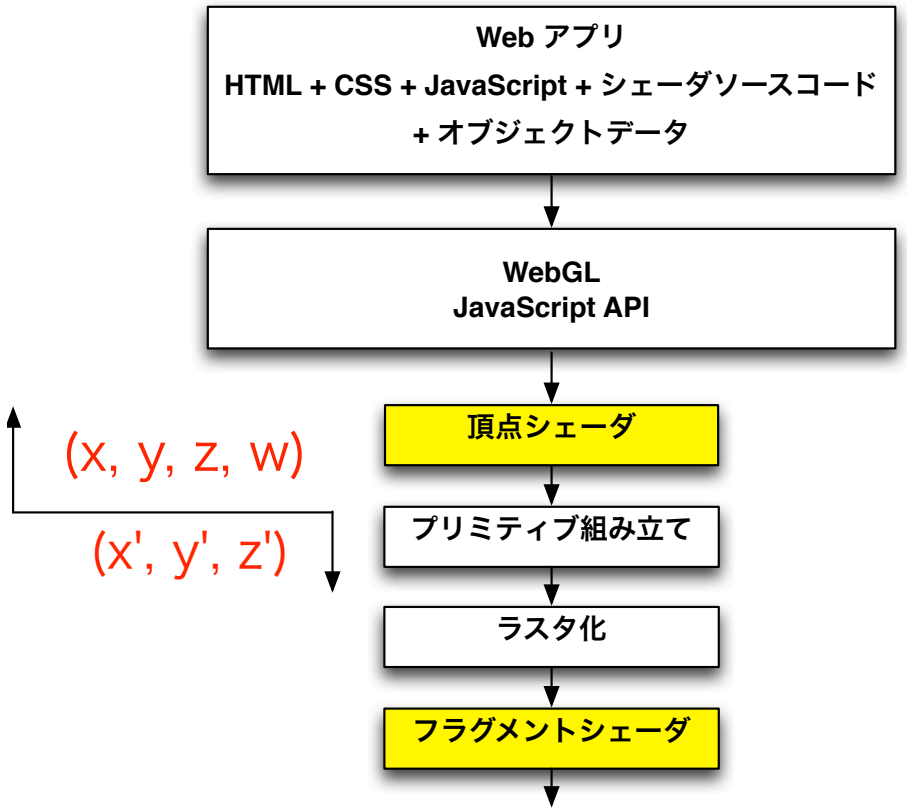
$$\begin{pmatrix} x^\dagger \\ y^\dagger \\ z^\dagger \\ w^\dagger \end{pmatrix} = \begin{pmatrix} \beta & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \alpha & \gamma \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(2) w 座標での割り算。これを透視除算という。

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x^\dagger/w^\dagger \\ y^\dagger/w^\dagger \\ z^\dagger/w^\dagger \end{pmatrix}$$

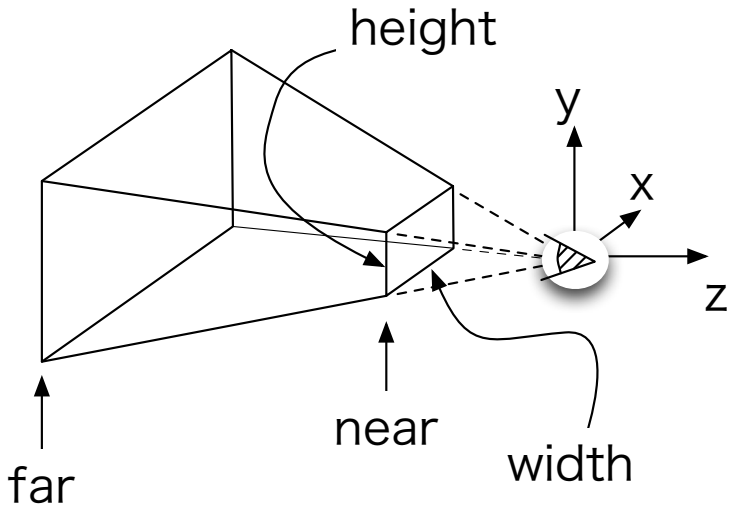
** この行列は一意には決まらない。ここで挙げるのは一つの例。

パイプラインでの座標の変化



- プリミティブ組み立て時
- $(x', y', z') = (x/w, y/w, z/w)$
- GPU が自動的に実行

透視射影



mat4.perspective(fovy, aspect, near, far, projectionMatrix);

fovy （field of view）は視野の高さ（y）方向の角度。aspect は縦横比。

上下と左右に非対象な frustum（視錐台）を作る場合[††]は、
mat4.frustum(left, right, bottom, top, near, far, projectionMatrix);

[††]どういう時？ ⇒ CAVE!

演習 01

演習 01

- 透視射影行列 mat を mat4.perspective で作り、その成分をみよ。
- mat4 に任意の位置座標 $(x, y, z, 1)$ をかけてどう変換されるかみよ。

座標変換

座標変換の実際

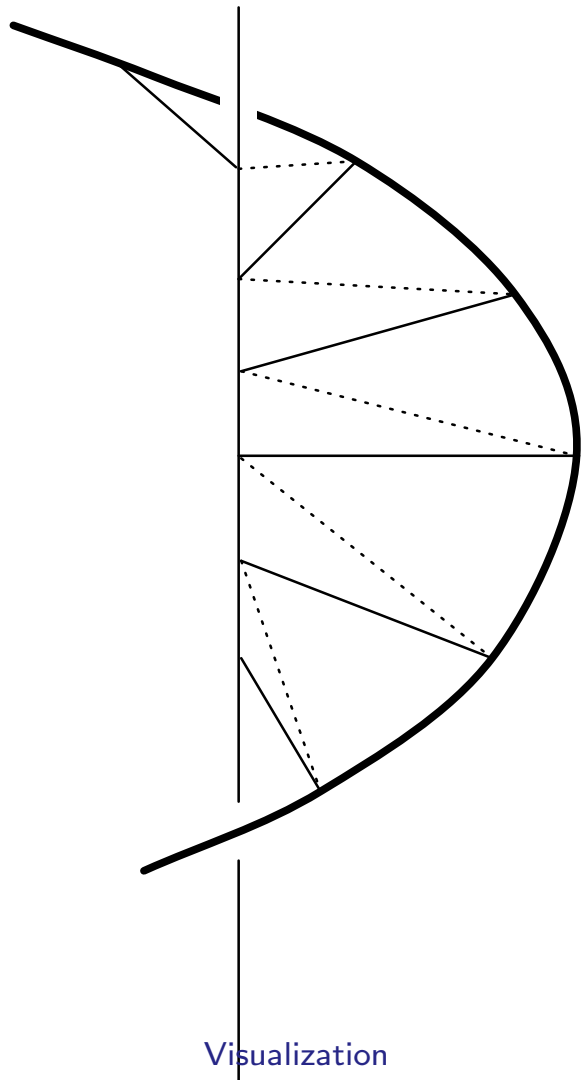
- モデルビュー変換
- 射影変換

について、あるオブジェクトに少しずつ座標変換をかけながらその効果を見る。

サンプル3D物体

らせん状の物体を考える

中心軸とそれにまきつく螺旋の間を面でつなぐ



webgl_sample_spiral_00.html

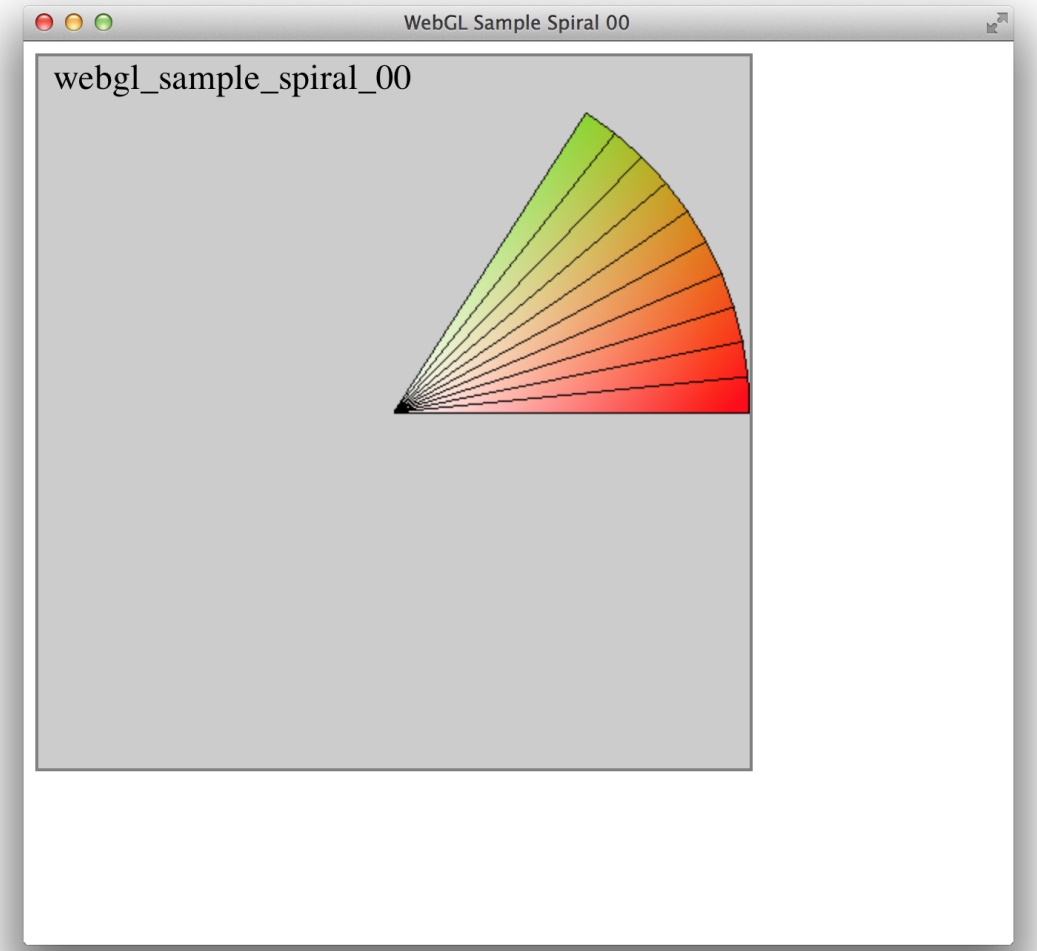
オブジェクトの構成部分

```
var dz = 0.1;
var pitch = 1.0;
var pof = 0;        // positionOffsetInFloats
var cof = 12;       // colorOffsetInBytes

for (var k=0; k<Nz; k++) {
  var z = k*dz;
  positionView[  pof] = 0.0;   // x
  positionView[1+pof] = 0.0;   // y
  positionView[2+pof] = z;     // z
  colorView[  cof] = 255;      // R
  colorView[1+cof] = 255;      // G
  colorView[2+cof] = 255;      // B
  colorView[3+cof] = 255;      // A
  pof +=vertexSizeInFloats;
```

```
    cof +=vertexSizeInBytes;

    phase = pitch*z;
    positionView[  pof] = Math.cos(phase);        // x
    positionView[1+pof] = Math.sin(phase);        // y
    positionView[2+pof] = z;                      // z
    colorView[  cof] = 255*Math.cos(phase);       // R
    colorView[1+cof] = 255*Math.sin(phase);       // G
    colorView[2+cof] = 255*Math.sin(phase*0.2);   // B
    colorView[3+cof] = 255;                       // A
    pof +=vertexSizeInFloats;
    cof +=vertexSizeInBytes;
  }
```

結果

何も変換していないので、見にくい。正規化ビューボリュームから外れている部分が見えない。



モデル変換
webgl_sample_spiral_01.html

モデルを移動・縮小・回転する行列 modelViewMatrix をつくり、頂点シェーダに送る。これも頂点属性（attribute）。

```
mat4.identity(modelViewMatrix);
mat4.translate(modelViewMatrix, [0.8,-0.5, -0.8]);
mat4.scale(modelViewMatrix,[0.3,0.3,0.3]);
mat4.rotateX(modelViewMatrix,-Math.PI/3);
mat4.rotateY(modelViewMatrix,-Math.PI/3);
gl.uniformMatrix4fv(uniLocation,
                    false,
                    modelViewMatrix);
//--</new>--


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

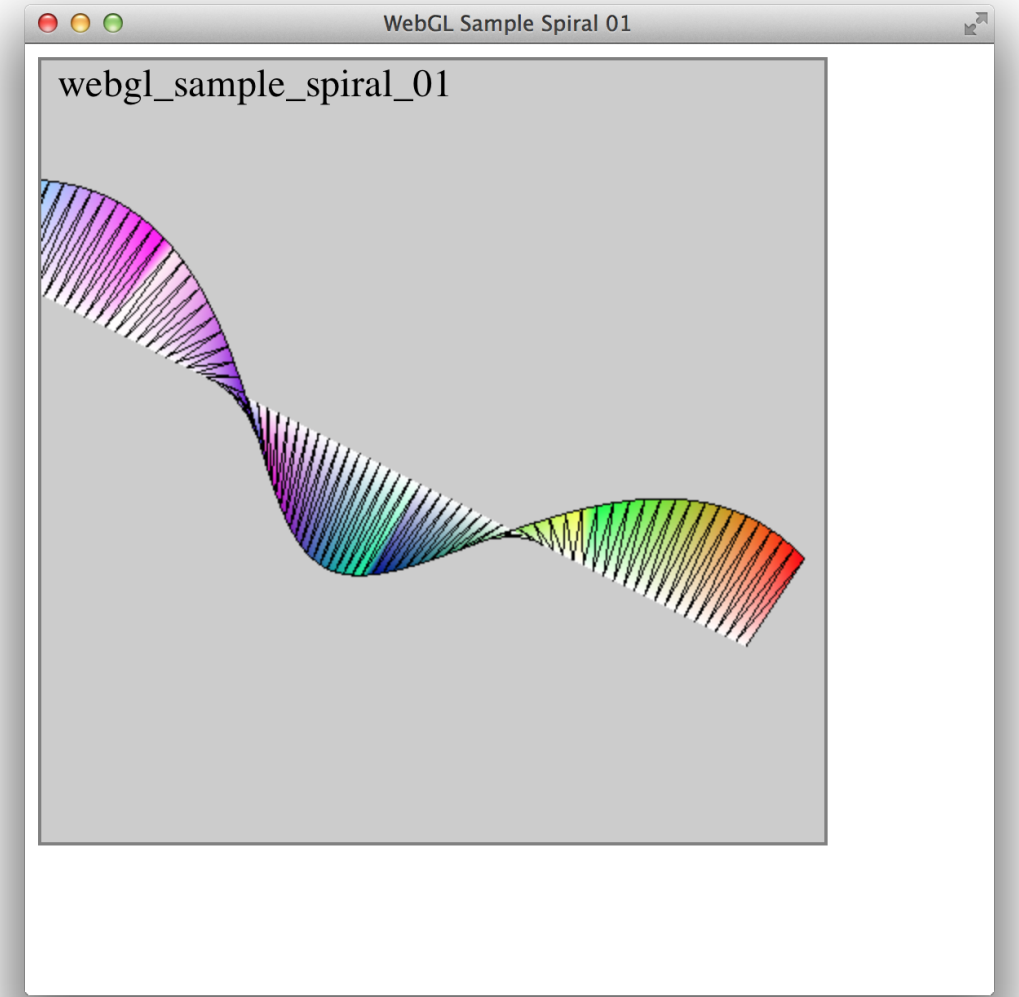
頂点シェーダ

modelViewMatrix も頂点の attribute

復習：全ての頂点に共通した attribute 値は uniform

```
<!-- new -->

<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;
  attribute vec4 aVertexColor;
  uniform   mat4 uMVMatrix;      //<--new
  varying   vec4 vColor;

  void main() {
    vColor = aVertexColor;
    gl_Position = uMVMatrix * vec4(aVertexPosition, 1.0); //
        <--new
  }
```

スナップショット
webgl_sample_spiral_01.html



射影変換
webgl_sample_spiral_02.html

常に正規化ビューボリュームのサイズを意識してモデルを移動・縮小・回転を設定するよりも、ワールド座標（CG 世界）の中にカメラを設定する、と考えた方が自然。

⇒ 射影変換。

mat4.perspective で projectionMatrix を作る

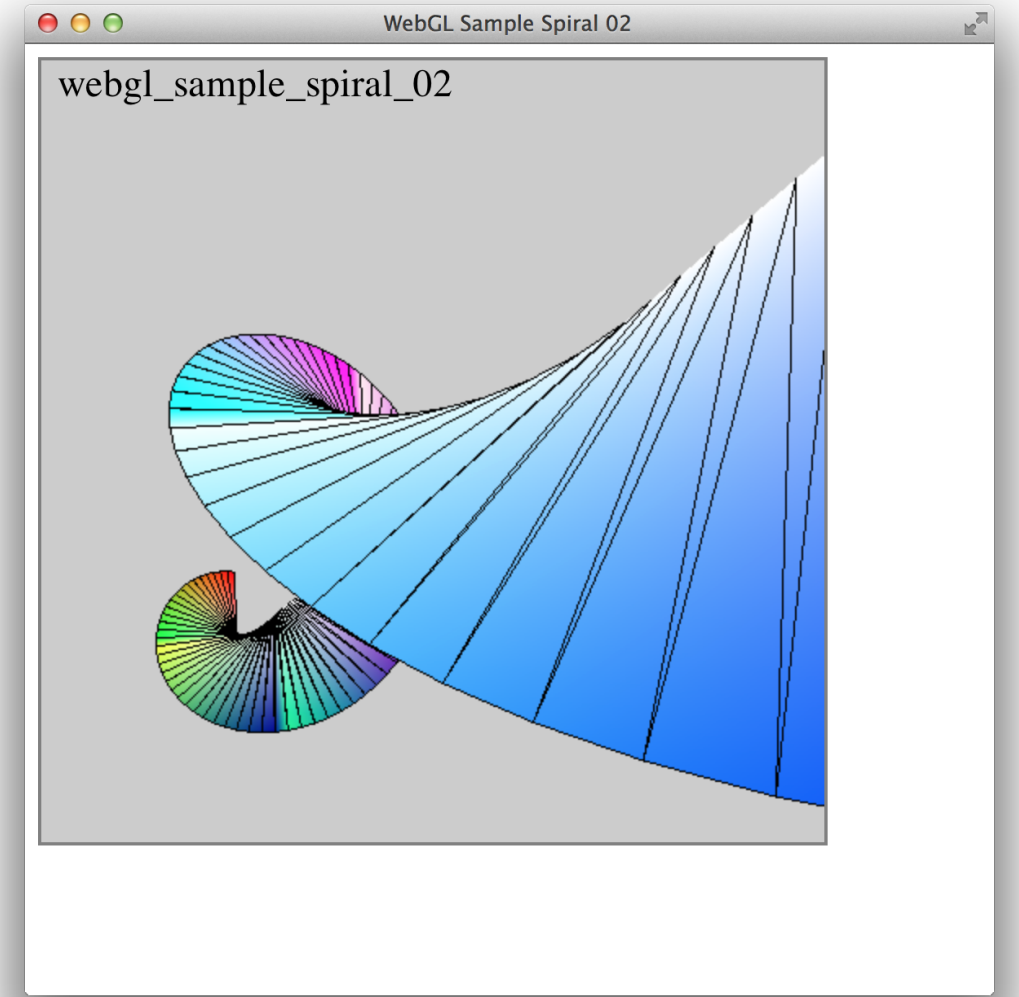
projectionMatrix を uniform として頂点シェーダに送る

シェーダではこの二つの行列をかける

gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);

このサンプルプログラムのこれ以外のポイントとして、デプステストとポリゴンオフセットがあるが説明は省略する。

スナップショット
webgl_sample_spiral_02.html



複数の変換行列の管理

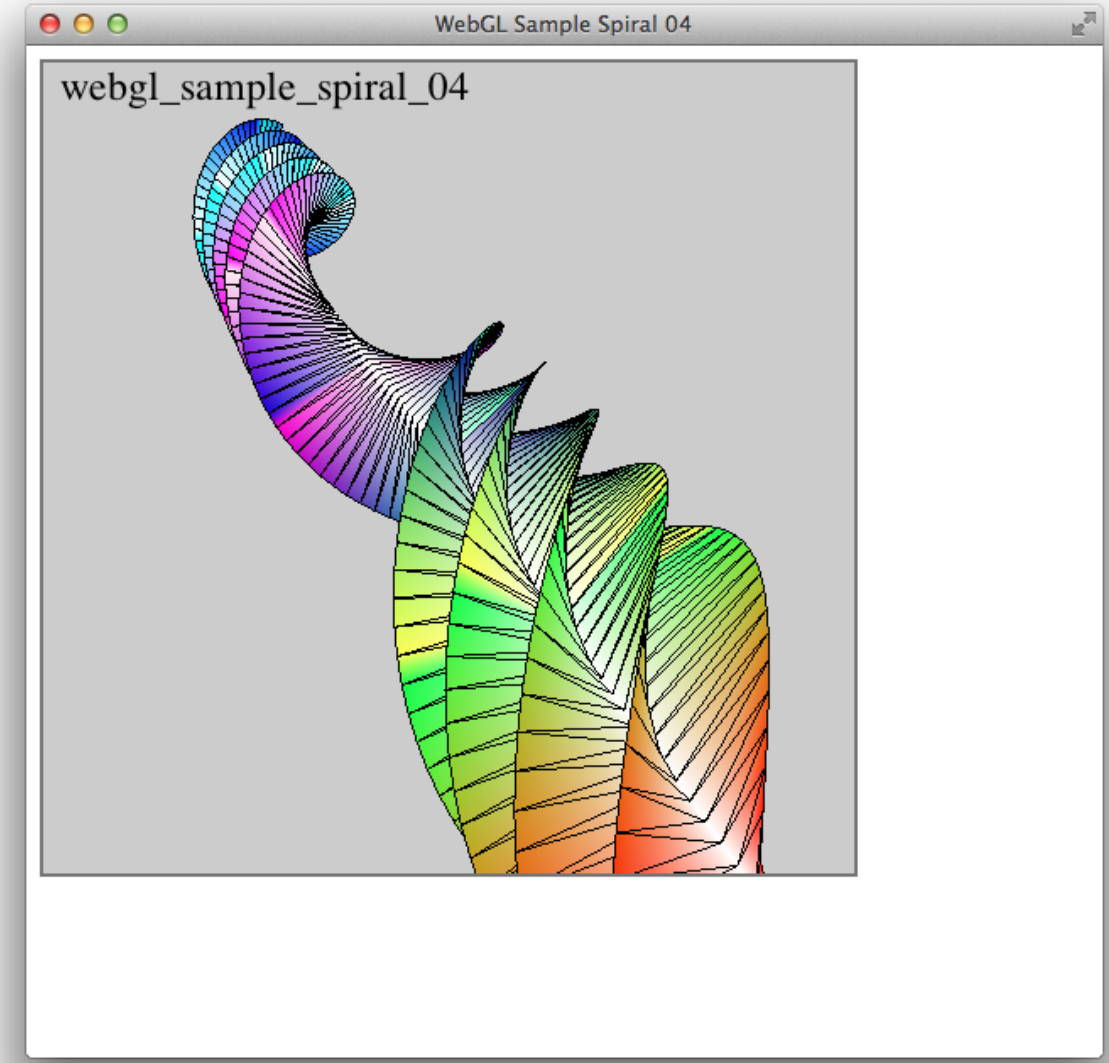
描画する物体が複数の要素から構成されているときなど、カメラ（射影行列）は固定していて、モデルビュー行列は頻繁に更新してロードする場合には、モデルビュー行列のスタックを使うのが便利。

以前の OpenGL （OpenGL 1.x）には glPushMatrix(), glPopMatrix() があったが、今の OpenGL にはない。

JavaScript には配列に push と pop のメソッドが備わっているのでこれを使えばいい。

練習

螺旋型の面を位相をずらして 5 枚描く



変換行列の push と pop
webgl_sample_spiral_04.html

```
    }
    modelViewMatrix = modelViewMatrixStack.pop();
}

//--new function
function uploadModelViewMatrixToShader() {
    gl.uniformMatrix4fv(uniLocation[1],
                        false,
                        modelViewMatrix);
}
```

変換行列のシェーダへのロード
webgl_sample_spiral_04.html

```
function uploadProjectionMatrixToShader() {
  gl.uniformMatrix4fv(uniLocation[0],
                      false,
                      projectionMatrix);
}

//--new function--
function draw_a_spiral() {
  // Draw triangles
```

描画
webgl_sample_spiral_04.html

```
}

function startup() {
  canvas = document.getElementById("myGLCanvas");
  gl = createGLContext(canvas);

  shader_program = create_shader_program();
```

レポート課題

- 正射影または透視射影を使い、任意の物体を描け。
- 提出はメールで。添付ファイルは２つ‡‡。
  1. レポート PDF ファイル
  2. 作成した HTML ファイル
     - ファイル拡張子は html
     - このファイル名で、そのままウェブ公開するかもしれないので注意
- gmail アドレス：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
  - どのような物体を描いたか
  - 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： 6/17（月）15:00

‡‡アーカイブはしないでください。

索引

References