

本科 5 年情報系 主専攻／副専攻

コンピュータグラフィックス

第3週

- ・グラフィック処理の基礎（2）

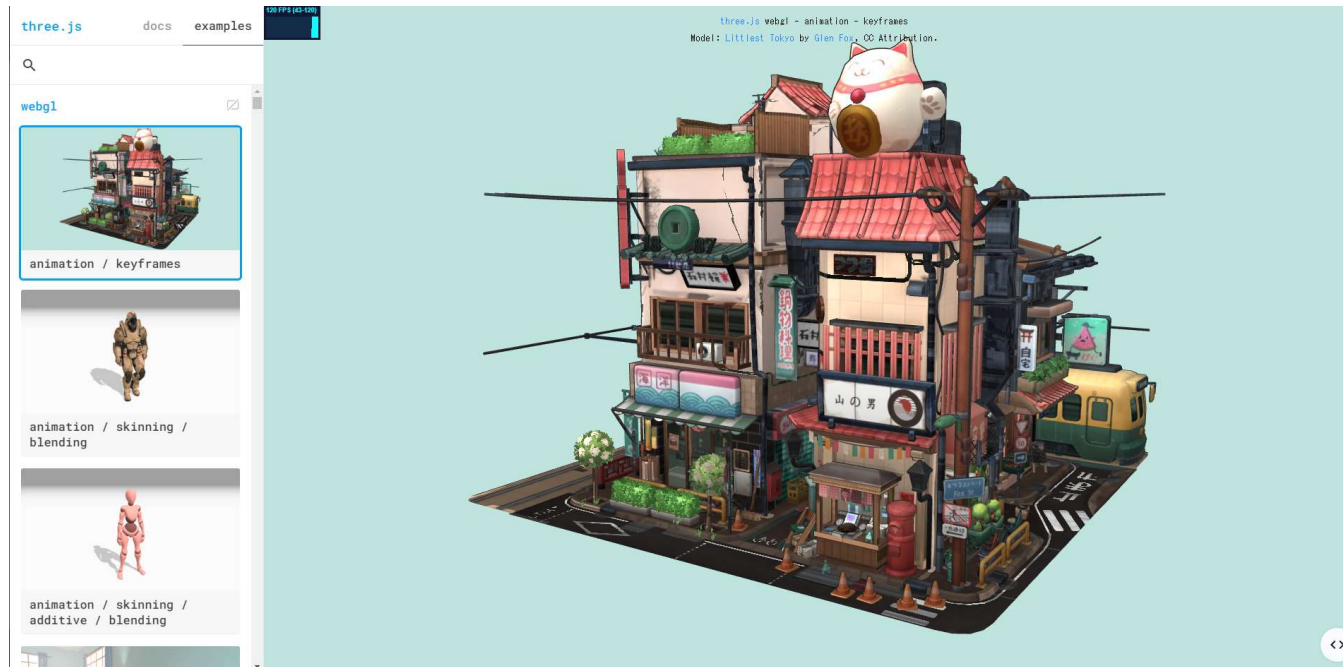
本日の授業

- ・ 復習) 基本図形の描画
- ・ 2次元座標系と図形の基本変換
- ・ WebGLにおける図形の変換

復習) WebGL

- OpenGLをWebブラウザで表示できるようにしたもの
- **JavaScript**に対応し, ブラウザによりOSに依存せず利用可能
- WebGLのAPIを簡略化する各種ライブラリあり
例) lightgl.js, Three.js など

Three.js : Webブラウザ上で3DCGを簡単に表示できるJavaScriptライブラリ



Three.js公式サイト
3DCGのサンプルを
閲覧(操作)できる

<https://threejs.org/examples/>

復習) WebGLを使ってみる

- WebGLの簡略化ライブラリとして **「lightgl.js」** を使用
- 主に初期設定等を簡単に記述することが可能
- 基本的な描画命令 (API) は**OpenGL**と同じ

```
template.html > ...
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <script src="lightgl.js"></script>
5     <script type="text/javascript">
6
7   function draw() {
8     const gl = GL.create();
9
10    gl.ondraw = function() {
11      // 特記しない限り、ここにOpenGLの命令 (機能) を記述
12      // OpenGLの命令を示す際には冒頭の『gl.』を省略する場合あり
13
14    };
15
16    gl.fullscreen();
17  }
18
19  </script>
20 </head>
21 <body onload="draw();">
22   <canvas id="glCanvas"></canvas>
23 </body>
24
25
26 </html>
```

↓

ここを書換え

演習用にテンプレートを用意
「template.html」

注：同じフォルダに
「lightgl.js」 が必要

復習) WebGLの基本命令 (1)

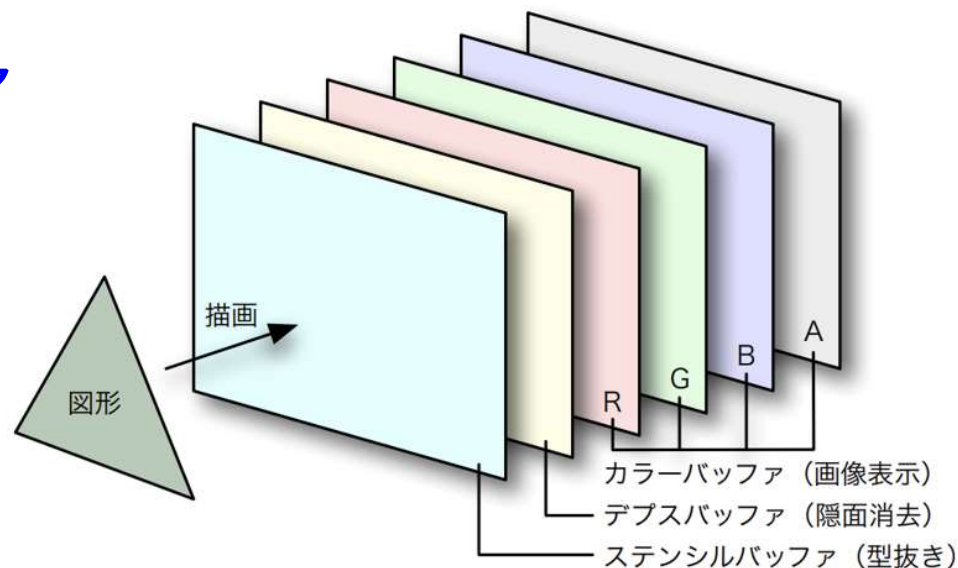
`gl.clearColor(r, g, b, a)`

- ・画面を消去する際の背景色（クリアカラー）を設定
- ・パラメータは r(赤), g(緑), b(青), a(アルファ値, 透明度)
(値は 0.0~1.0 の範囲)

`gl.clear(mask)`

- ・画面の表示内容（フレームバッファ）を消去
- ・maskには `gl.COLOR_BUFFER_BIT` や `gl.DEPTH_BUFFER_BIT` を指定して、色情報や深度情報をクリアする

フレームバッファ のイメージ



復習) WebGLの基本命令 (1) サンプルコード

例) 背景色を変えてみる

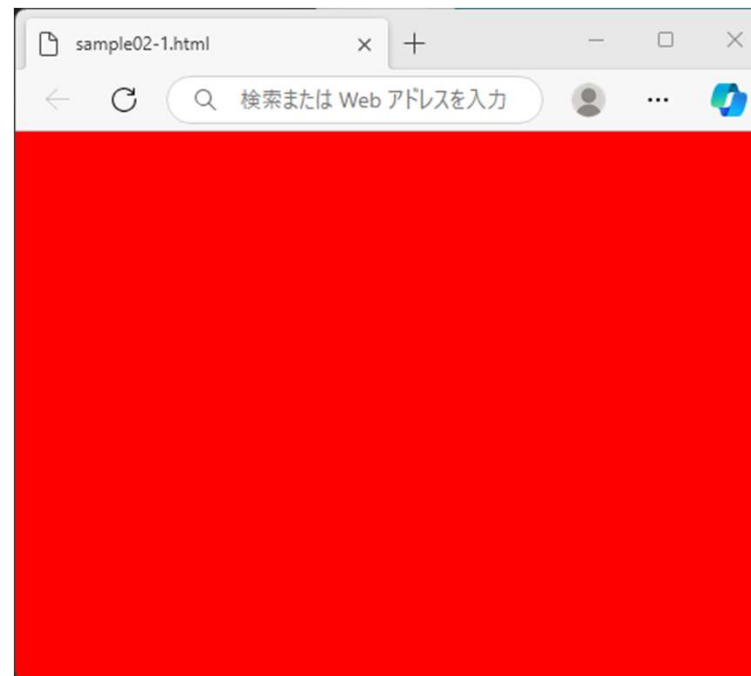
sample02-1.html

```
gl.clearColor(1.0, 0.0, 0.0, 1.0);  
gl.clear(gl.COLOR_BUFFER_BIT);
```

画面消去時の色を指定
(RGBA形式)

画面を消去
(色情報をリセット)

注: テンプレートの改変部のみ記述



背景色が赤になる
(1.0, 0.0, 0.0)

復習) WebGLの基本命令 (2) lightgl.jsの図形描画

`gl.begin(mode)`

- ・ 描画を開始する
- ・ modeには描画したい図形を指定する


`gl.vertex(x, y, z)`

- ・ 頂点の位置座標 (x,y,z) を指定する
- ・ この命令で設定した頂点を基に, 指定した図形が描かれる

`gl.end()`

- ・ 描画を終了する

図形描画の流れ

- ① `gl.begin(図形の指定);`
- ②  `gl.vertex(x1, y1, z1);`
`gl.vertex(x2, y2, z2);`
`. . .`
`gl.vertex(xn, yn, zn);`
- ③ `gl.end();`

復習) WebGLの基本命令 (2) サンプルコード

例) 三角形を描いてみる

sample02-2.html

```
gl.clearColor(1.0, 1.0, 1.0, 1.0);  
gl.clear(gl.COLOR_BUFFER_BIT);  
  
gl.translate(0.0, 0.0, -3.0);  
  
gl.color(1.0, 0.0, 0.0, 0.0);  
  
gl.begin( gl.TRIANGLES );  
    gl.vertex( 0.0, 1.0, 0.0 );  
    gl.vertex(-1.0, -1.0, 0.0 );  
    gl.vertex( 1.0, -1.0, 0.0 );  
gl.end();
```

画面のリセット (白)

図形を奥 (-z) に動かす

以降の図形の色を赤にする

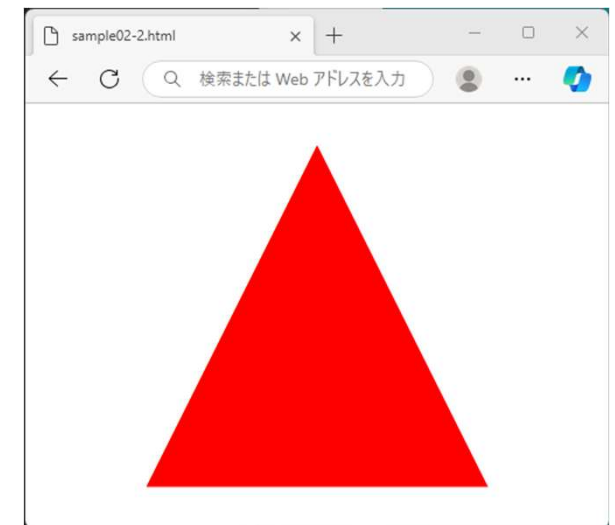
「三角形」を指定して描画開始

三角形の頂点座標 (x,y,z)

描画終了

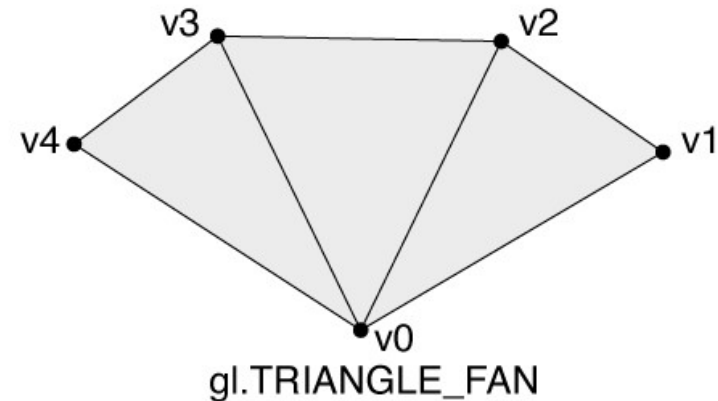
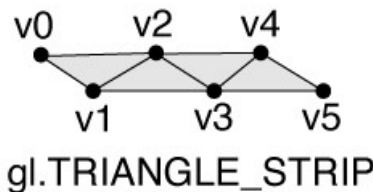
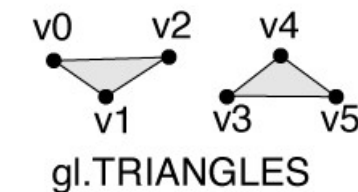
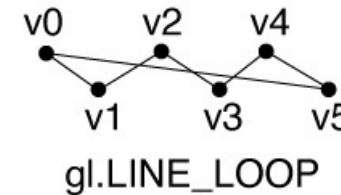
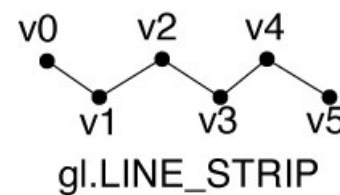
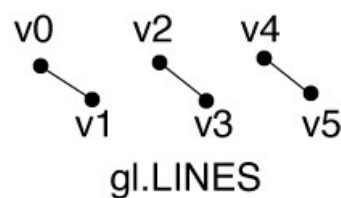
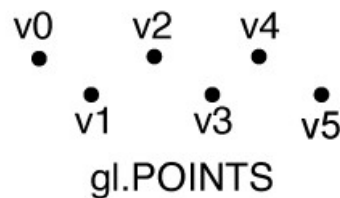
注: テンプレートの改変部のみ記述

赤色の三角形が描画される



復習) 使用できる基本図形 (プリミティブ)

描画モード	説明
gl.POINTS	各頂点を独立した点として描く
gl.LINES	2つの頂点を一組とし, それぞれ独立した線として描く
gl.LINE_STRIP	頂点を順に結んで連続した線を描く (始点と終点は非接続)
gl.LINE_LOOP	頂点を順に結んで連続した線を描く (始点と終点を接続)
gl.TRIANGLES	3つの頂点を一組とし, それぞれ独立した三角形として描く
gl.TRIANGLE_STRIP	初めの3頂点で三角形を描き, それ以降の頂点で連続した三角形を描く
gl.TRIANGLE_FAN	最初の頂点を中心に, 扇形に連続した三角形を描く



復習) WebGLの基本命令 (3) lightgl.jsの図形描画

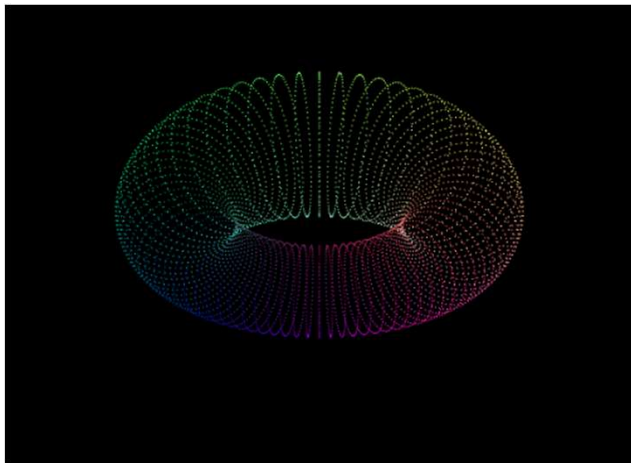
gl.color(r, g, b, a)

- ・ 描画時の色を指定する
- ・ パラメータは r(赤), g(緑), b(青), a(アルファ値, 透明度)
(値は 0.0~1.0 の範囲)

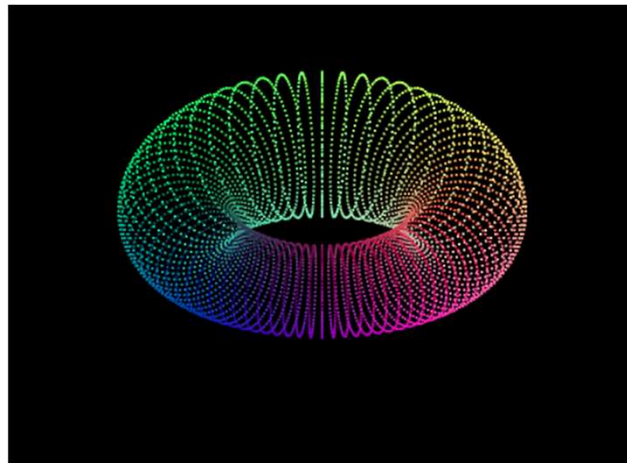
gl.pointSize(size)

gl.POINTSで描画する際の, 点の大きさを設定する (単位はピクセル)

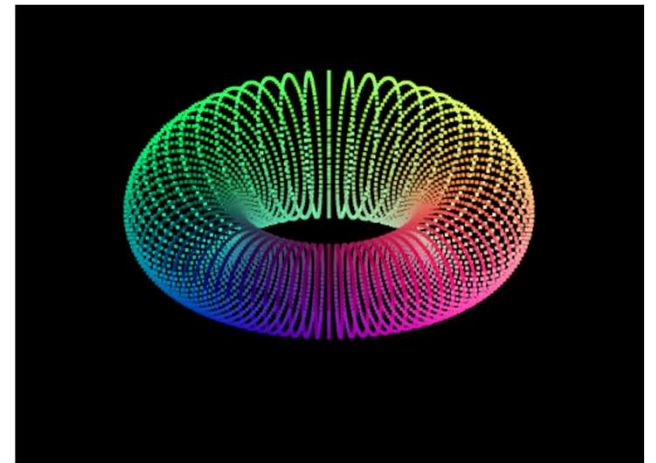
例) 点群で描いたトーラス



size = 1.0



size = 2.0



size = 3.0

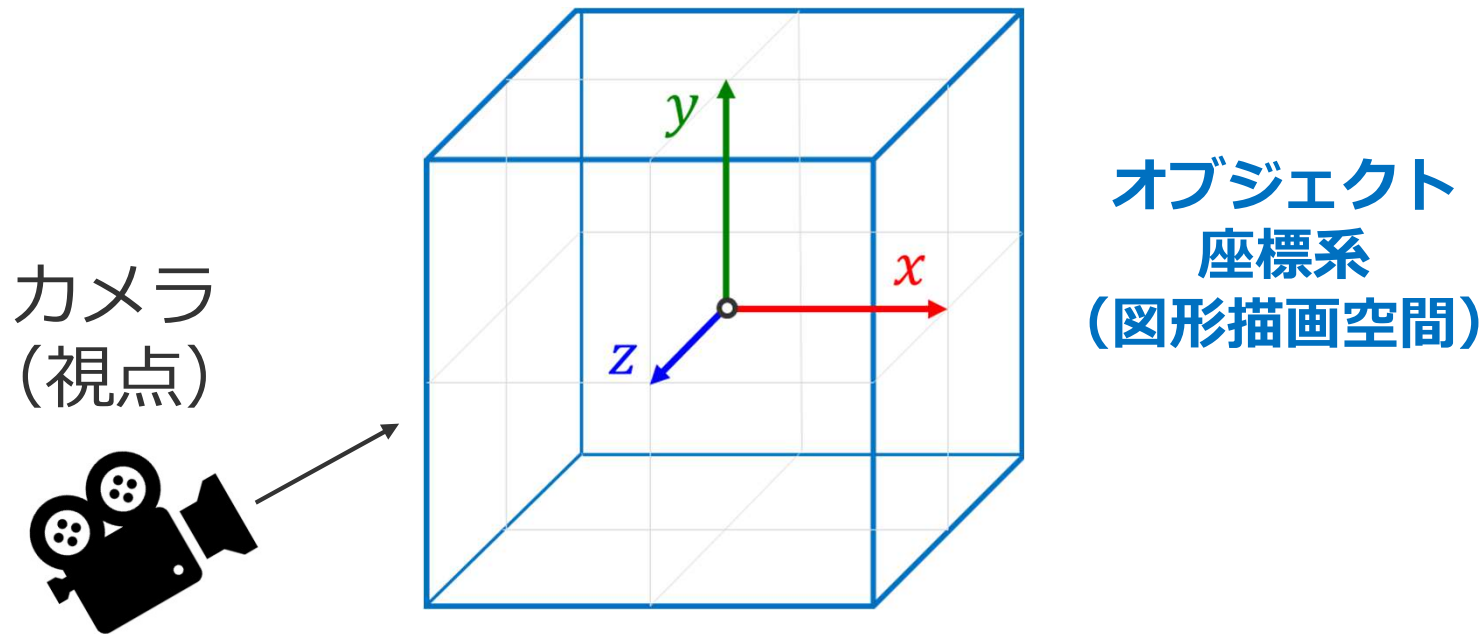
復習) WebGLの基本命令 (4) 座標変換

`gl.translate(x, y, z)`

オブジェクトの位置を, 指定した距離 (x,y,z) だけ移動する

補足)

WebGLでは**オブジェクト座標系**と呼ばれる座標系内に図形を描画
→ カメラ (視点) から見た映像を画面に出力



補足) JavaScriptの基礎

- 基本的な文法は C言語／C++ とほぼ同じ
例) 代入文, セミコロン, if 文, for 文, コメント文など

条件分岐

if, else if, else

```
javascript

let score = 75;
if (score > 80) {
    console.log("Good");
} else if (score > 60) {
    console.log("Okay");
} else {
    console.log("Try again");
}
```

くり返し

for, while

```
javascript

for (let i = 0; i < 5; i++) {
    console.log(i);          // 0, 1, 2, 3, 4
}
```

```
javascript

let i = 1; // 初期値
while (i <= 5) { // 条件
    console.log(i); // 現在の値を出力
    i++; // カウンタを増加
}
```

補足) JavaScriptの基礎

- 値の変化を伴わない定数は **const**, 変化を伴う変数は **let** で変数宣言
- 型指定は不要 (重複等の心配がなければ, const や let による変数宣言も不要)

javascript

```
let x = 10;      // 変更可能な変数
const y = 20;    // 定数 (変更不可)
var z = 30;      // 旧スタイルの変数宣言 (あまり使わない)
```

javascript

```
let number = 5;      // 数値
let text = "Hello";  // 文字列
let isTrue = true;   // ブール値
let nothing = null;   // null 値
let notDefined;      // 未定義 (undefined)
```

補足) JavaScriptの基礎

- 関数定義は次の構成で記述する

```
function 関数名 ( 引数, ... ) { ...; return 戻り値; }
```

javascript

```
function greet(name) {  
  return "Hello, " + name;  
}  
console.log(greet("Alice")); // "Hello, Alice"
```

- ライブラリ関数は「**ライブラリ名.関数名**」の形で指定する
例) 数学ライブラリ: Math.sin, Math.cos, Math.PI など

javascript

```
// ラジアンでの三角関数  
console.log(Math.sin(Math.PI / 2)); // 1  
console.log(Math.cos(0)); // 1  
console.log(Math.tan(Math.PI / 4)); // 1
```

補足) console.log()は、
Webブラウザ上では使用不可

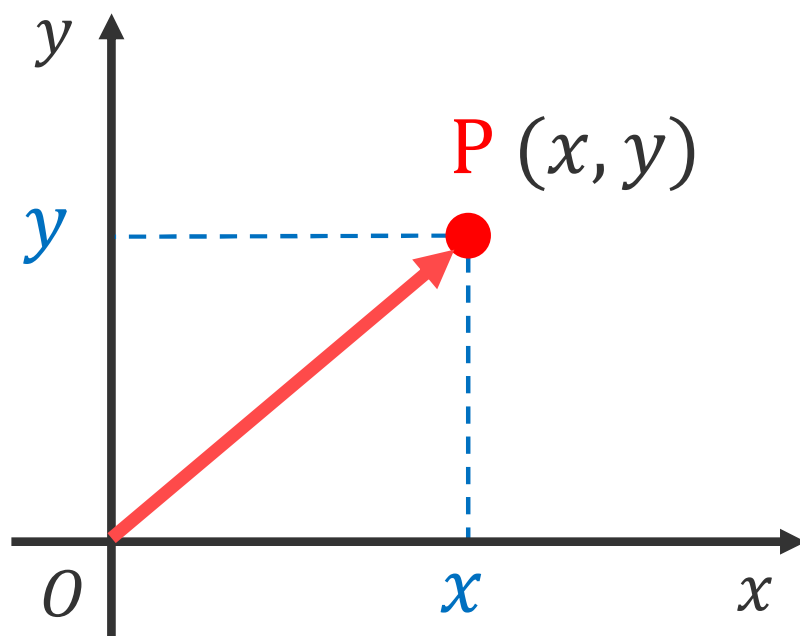
本日の授業

- ・ 復習) 基本図形の描画
- ・ 2次元座標系と図形の基本変換
- ・ WebGLにおける図形の変換

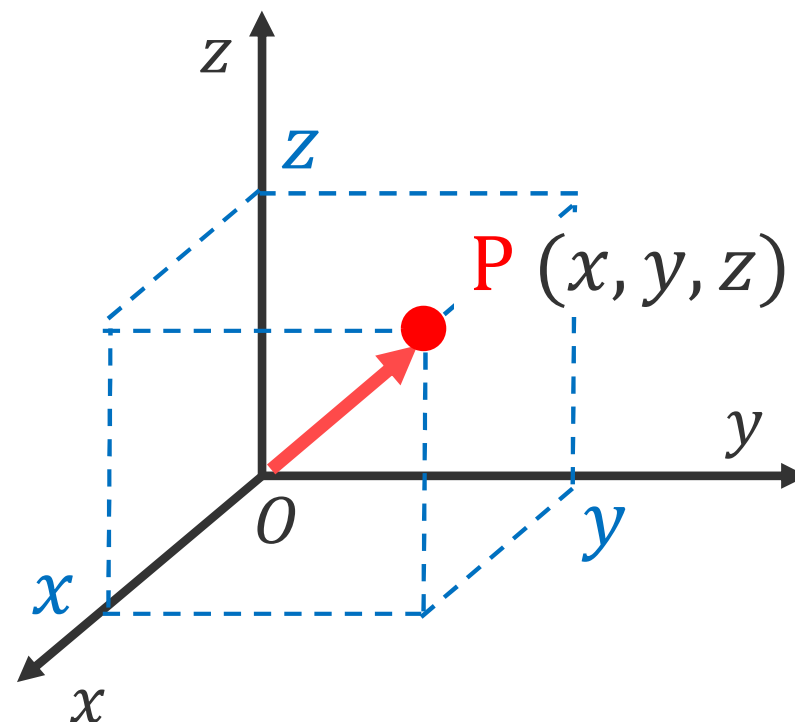
座標系

座標系 (Coordinate System)

- **原点**や**座標軸**など、空間の1点を定める基準を与えるもの
- CGにおける図形は、座標系を基準に定義される。
(一般に、**直交座標系**が用いられる)



2次元直交座標系



3次元直交座標系

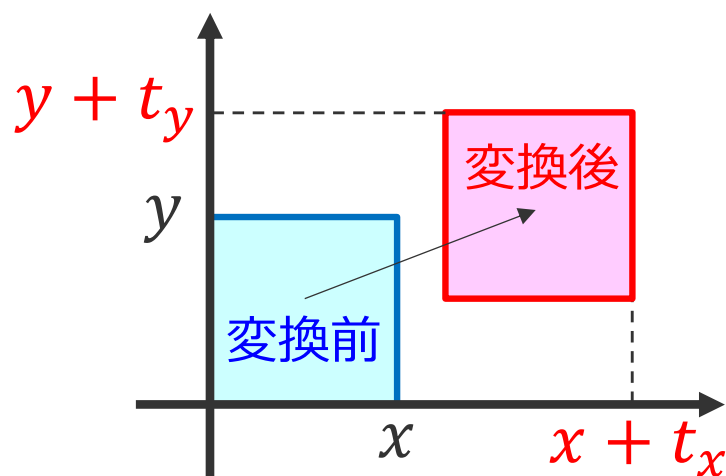
2次元図形の基本変換

図形は、頂点などの点群により構成される

→ 図形を構成する点に**幾何学的変換**を行うことで図形を操作

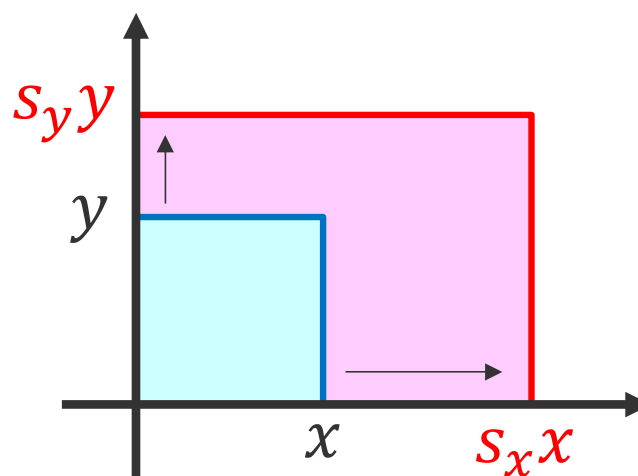
例) 平行移動, 拡大・縮小, 回転, せん断 (スキュー)

① 平行移動 (Translation)



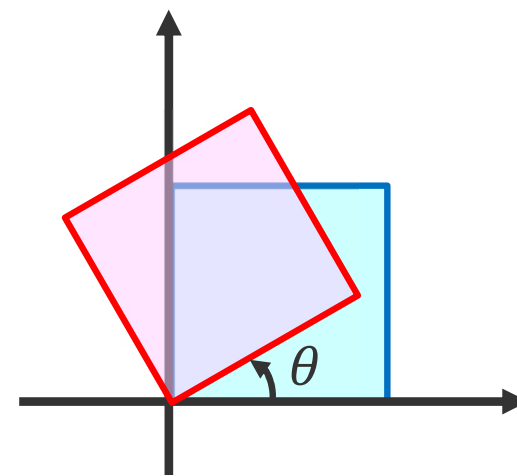
$$\begin{bmatrix} x + t_x \\ y + t_y \end{bmatrix}$$

② 拡大縮小 (Scaling)



$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

③ 回転 (Rotation)

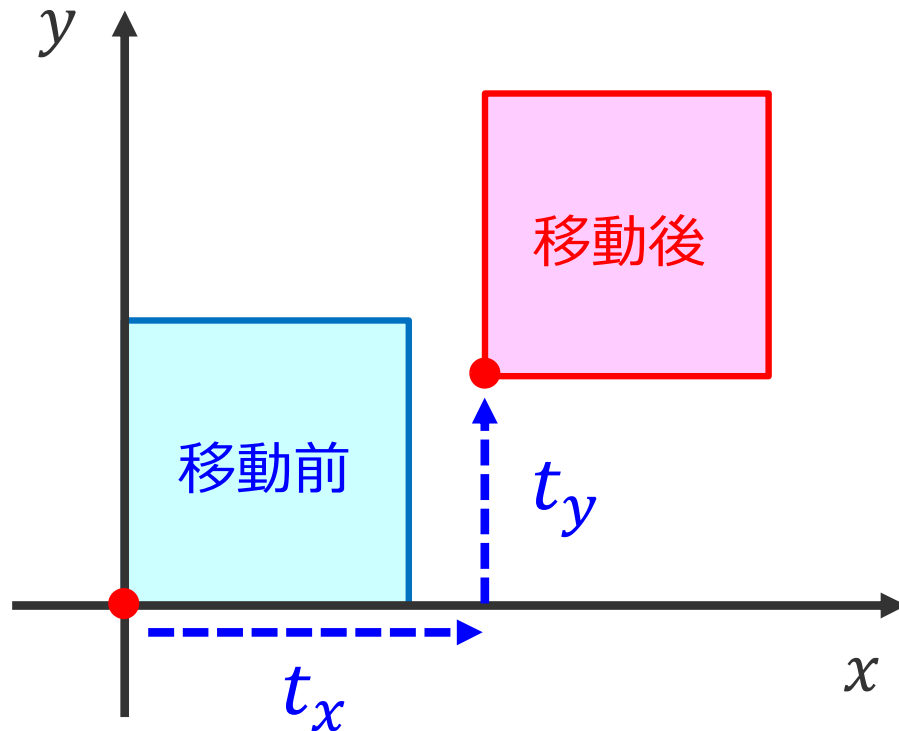


$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

基本変換 ① 平行移動

① 平行移動 (Translation)

- 図形全体をある方向に一定の距離だけ動かす操作
- 平行移動は、各点の座標に**定数を加える**ことで実現



$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

(x, y) : 移動前の座標

(x', y') : 移動後の座標

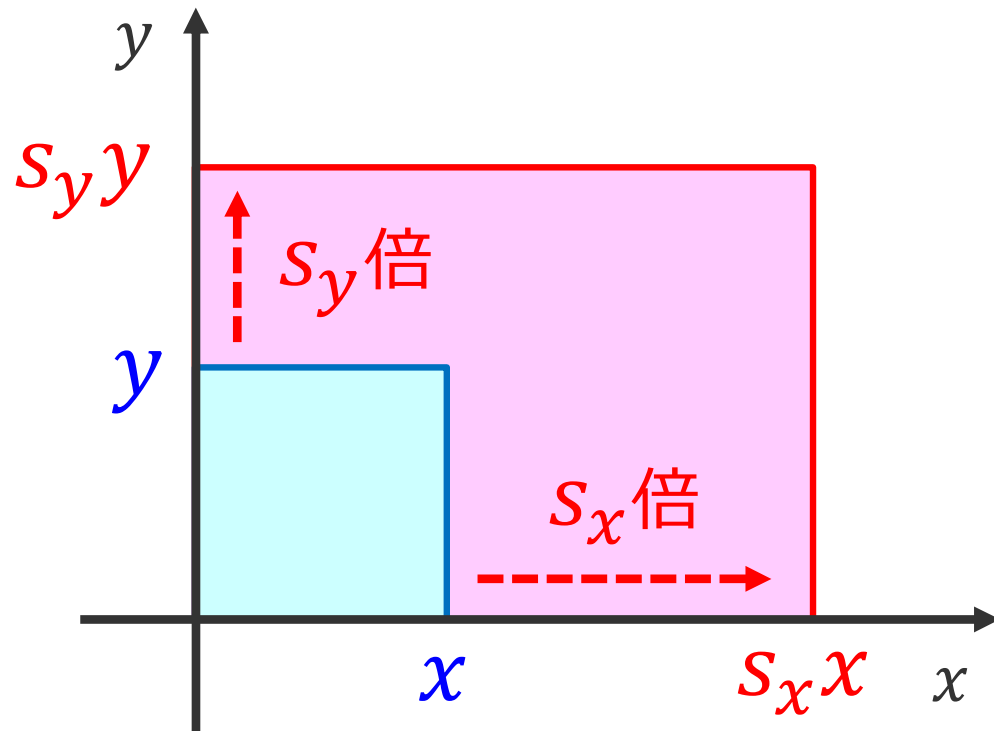
t_x : x 軸方向の移動量

t_y : y 軸方向の移動量

基本変換 ② 拡大・縮小

② 拡大・縮小 (Scaling)

- 図形を特定の倍率で拡大または縮小する操作
- 各点の座標に**倍率をかける**ことで実現



$$\begin{cases} x' = S_x x \\ y' = S_y y \end{cases}$$

(x, y) : 変換前の座標

(x', y') : 変換後の座標

S_x : x 軸方向の倍率

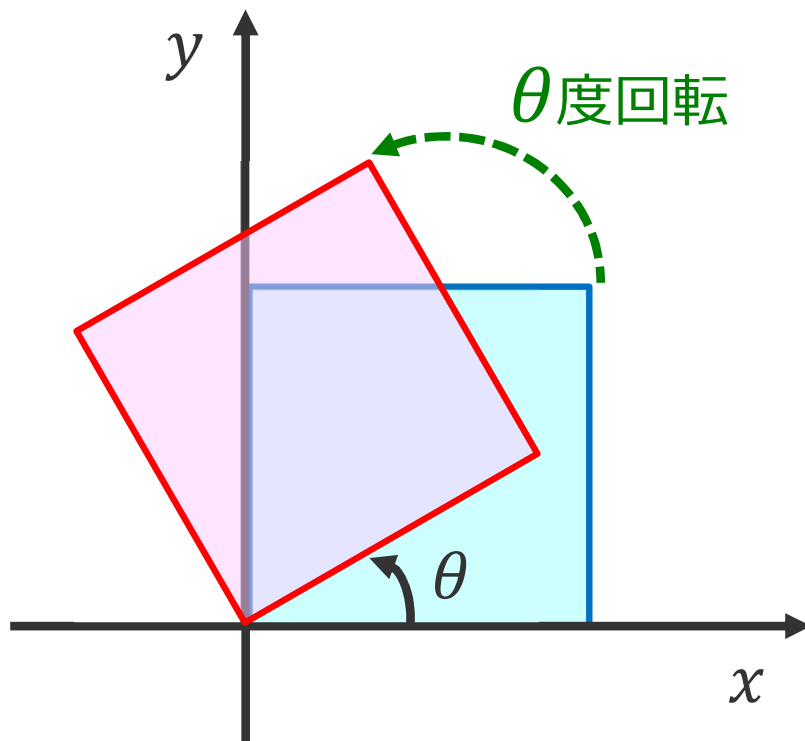
S_y : y 軸方向の倍率

($S_x, S_y > 1$ のとき拡大, $S_x, S_y < 1$ のとき縮小)

基本変換 ③ 回転

③ 回転 (Rotation)

- ある基準点（通常は原点）を中心にして**図形を回転**させる操作
- 回転角度によって、各点の新しい座標が計算される



$$\begin{cases} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \end{cases}$$

行列を使った表現

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

(x, y) : 変換前の座標

(x', y') : 変換後の座標

θ : (原点回りの) 回転角度 (正 : 左回り)

補足) 回転行列の導出

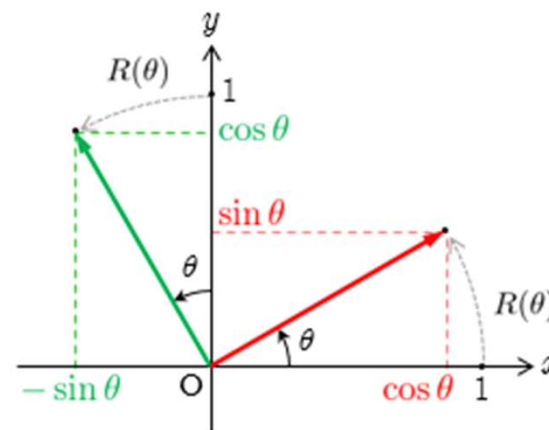
2次元直交座標平面において、原点を中心に θ だけ回転する変換

を表す回転行列を $R(\theta)$ とすると、基本ベクトル $\mathbf{i} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$,

$\mathbf{j} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ は $R(\theta)$ によって,

$$R(\theta)\mathbf{i} = R(\theta) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix}$$

$$R(\theta)\mathbf{j} = R(\theta) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -\sin \theta \\ \cos \theta \end{pmatrix}$$



のように変換される. このことから、一般のベクトル $\mathbf{r} = \begin{pmatrix} x \\ y \end{pmatrix} = x\mathbf{i} + y\mathbf{j}$ を $R(\theta)$ によって変換すると

$$\begin{aligned} R(\theta)\mathbf{r} &= R(\theta)(x\mathbf{i} + y\mathbf{j}) = xR(\theta)\mathbf{i} + yR(\theta)\mathbf{j} = x \begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix} + y \begin{pmatrix} -\sin \theta \\ \cos \theta \end{pmatrix} \\ &= \begin{pmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \end{aligned}$$

となるので,

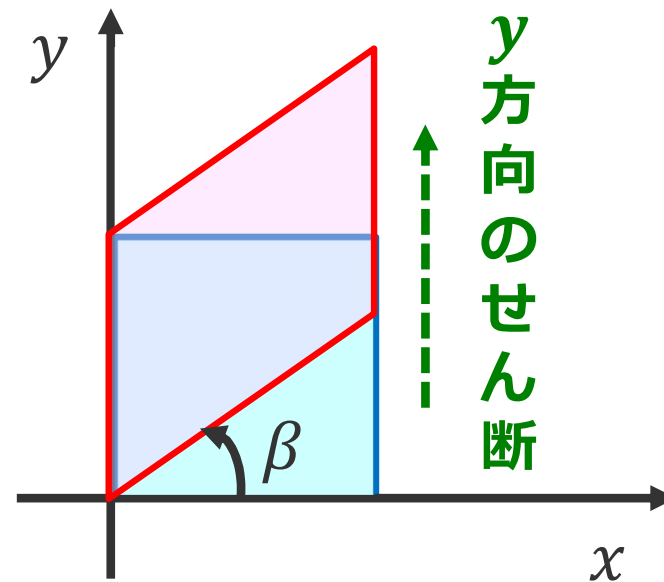
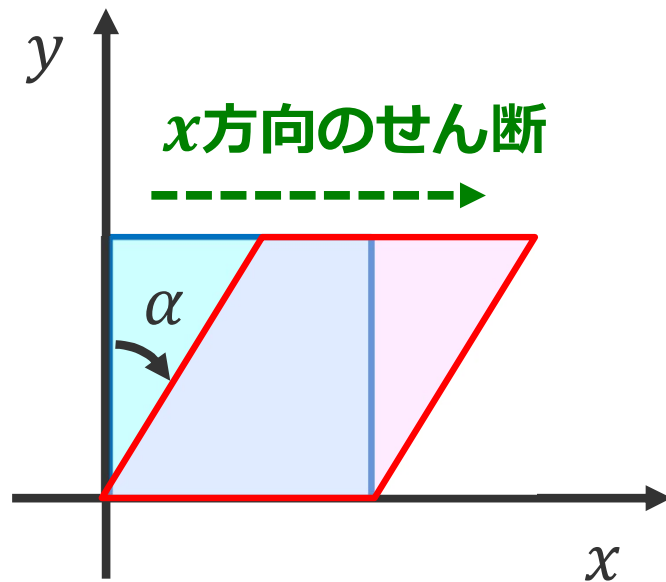
$$R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

である.

基本変換 ④ せん断 (スキュー)

④ せん断 (Shear), スキュー (Skew)

- 図形を特定の方に「傾ける」操作
- x 方向と y 方向のせん断があり, 方向によって異なる形となる



(x, y) : 変換前の座標
 (x', y') : 変換後の座標
 α, β : 傾ける角度

$$\begin{cases} x' = x + y \tan \alpha \\ y' = y \end{cases}$$

x方向のずれ

$$\begin{cases} x' = x \\ y' = y + x \tan \beta \end{cases}$$

y方向のずれ

本日の授業

- ・ 復習) 基本図形の描画
- ・ 2次元座標系と図形の基本変換
- ・ WebGLにおける図形の変換

図形の基本変換（２） サンプルコード

例）三角形を平行移動・回転してみる

Sample03-1.html

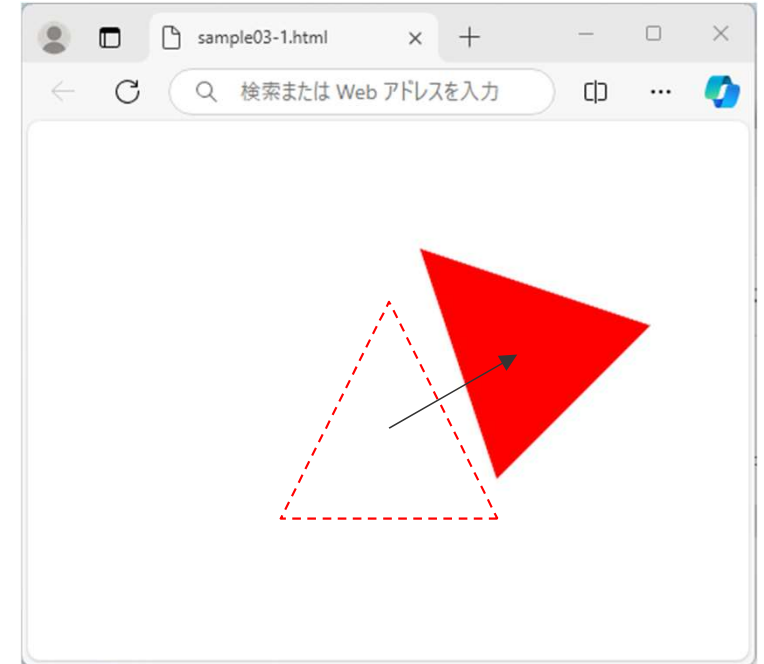
```
gl.pushMatrix();

gl.translate(0.5, 0.3, 0.0);

gl.rotate(45.0, 0.0, 0.0, 1.0);

gl.color(1.0, 0.0, 0.0, 0.0);
gl.begin(gl.TRIANGLES);
    gl.vertex(-0.5, -0.5, 0);
    gl.vertex(0.5, -0.5, 0);
    gl.vertex(0, 0.5, 0);
gl.end();

gl.popMatrix();
```



コードの前半部は省略

図形の基本変換（２） サンプルコード

例）三角形を平行移動・回転してみる

Sample03-1.html

```
gl.pushMatrix(); ← 変換行列をスタックに保存

gl.translate(0.5, 0.3, 0.0); ← 平行移動 (x方向0.5、y方向0.3)

gl.rotate(45.0, 0.0, 0.0, 1.0); ← z軸周りに45度回転

gl.color(1.0, 0.0, 0.0, 0.0);
gl.begin(gl.TRIANGLES);
    gl.vertex(-0.5, -0.5, 0);
    gl.vertex(0.5, -0.5, 0);
    gl.vertex(0, 0.5, 0);
gl.end(); ← 三角形の描画

gl.popMatrix(); ← 変換行列をスタックから取り出し
```

コードの前半部は省略

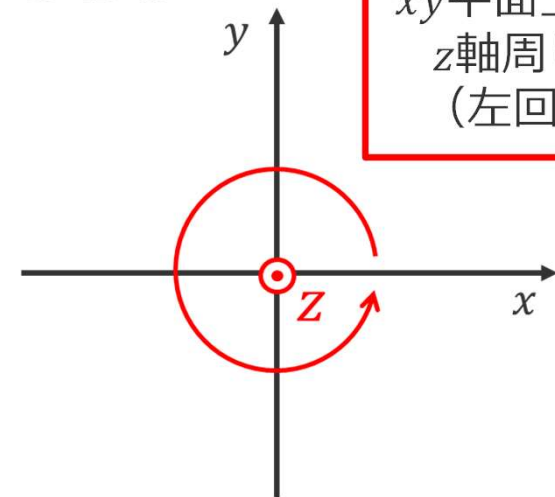
WebGLの基本変換命令（1） 平行移動，回転

`gl.translate(x, y, z)`

物体を指定したベクトル方向（ x, y, z ）に平行移動させる

`gl.rotate(angle, x, y, z)`

- ・ 指定した回転軸（ x, y, z ）に沿って物体を回転させる
- ・ `angle` は，回転する角度を [deg] で指定する



xy平面上の回転は
z軸周りの回転
(左回転が正)

`gl.scale(x, y, z)`

- ・ 物体の大きさを拡大または縮小する
- ・ 各軸方向（ x, y, z ）に設定した倍率で，図形が拡大・縮小する

WebGLの基本変換命令（2） 変換行列の保存と復元

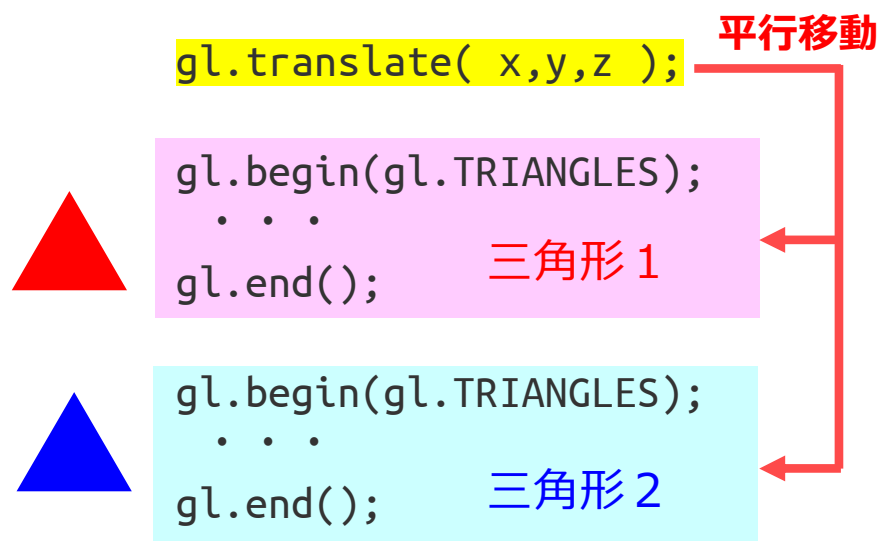
gl.pushMatrix()

- ・図形を描画する前に、現在の**座標変換（平行移動や回転）を保存**する
- ・図形ごとに変換を適用でき、**他の図形にその影響を与えない**ようにできる

gl.popMatrix()

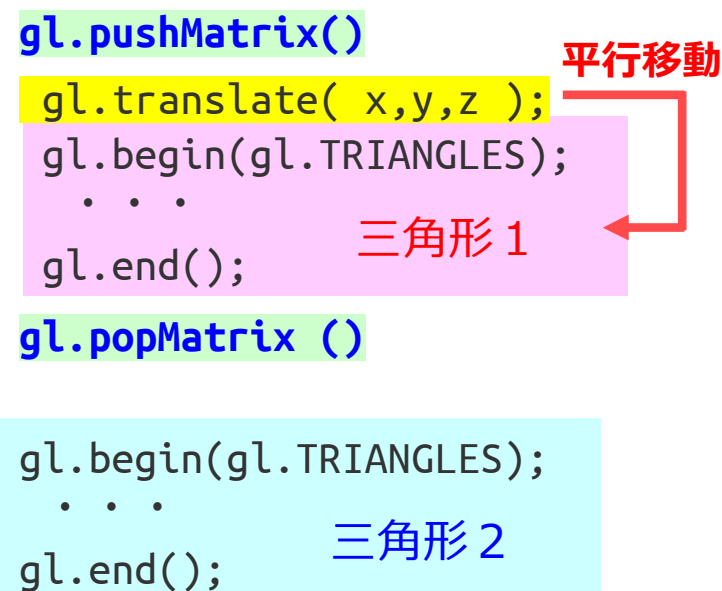
保存していた**座標変換の状態を復元**する。

そのままだと



変換が両方の図形に作用する

pushMatrix, popMatrixを使うと



変換の作用する範囲を決定できる

本日の演習

課題 3-1 図形の変換（1）

次に示す WebGL のサンプルコード **sample03-1** を実行し、図形の変換関数(**rotate**, **translate**)の役割を確認しなさい。（ひな型ファイル **Template.html** 内に以下のコードを入力する形で実行すること）

sample03-1

```
gl.clearColor(1.0, 1.0, 1.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);
gl.translate(0.0, 0.0, -3.0);

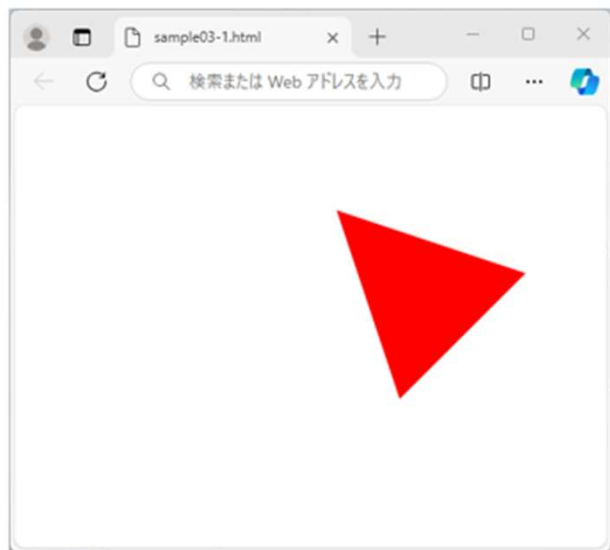
gl.pushMatrix(); //変換行列をスタックに保存

    gl.rotate(45.0, 0.0, 0.0, 1.0); // ①回転

    gl.translate(0.5, 0.3, 0.0); // ②平行移動

// 三角形の描画
gl.color(1.0, 0.0, 0.0);
gl.begin(gl.TRIANGLES);
    gl.vertex(-0.5, -0.5, 0);
    gl.vertex(0.5, -0.5, 0);
    gl.vertex(0, 0.5, 0);
gl.end();

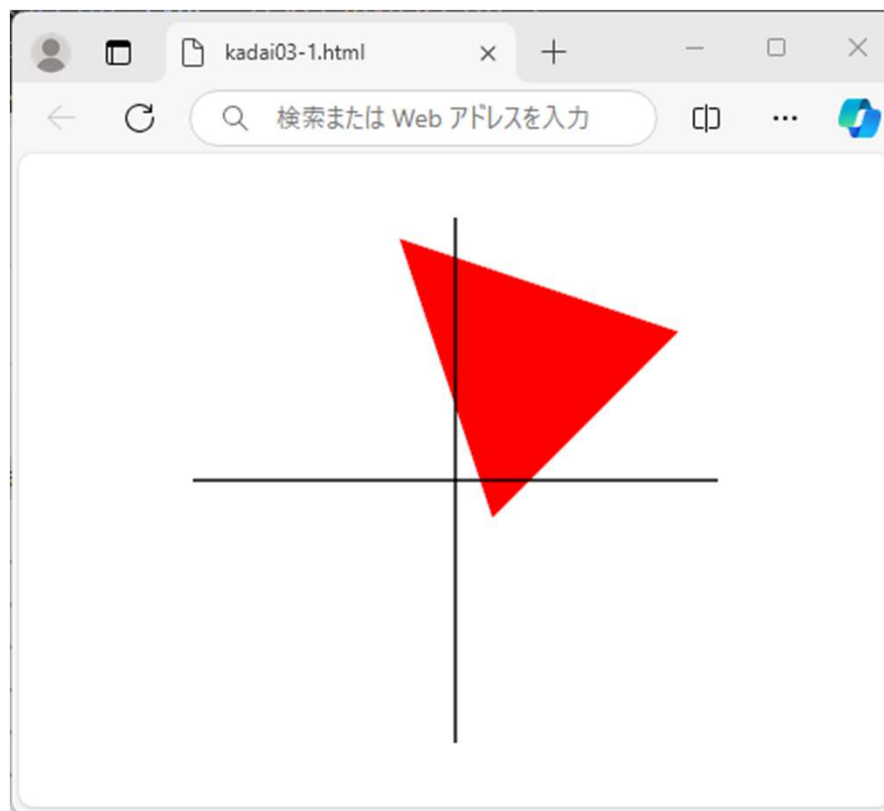
gl.popMatrix(); //変換行列をスタックから取り出す
```



本日の演習

課題 3-2 座標軸の描画

サンプルコード **sample03-1** を書き換えて、次の図のような**座標軸**を描画できるようにしなさい。なお、座標軸は、図形に各種変換を与えてもその影響を受けないものとする。



本日の演習

課題 3-4 図形の変換

次に示す WebGL のサンプルコード **sample03-4** は、次のような正方形を描くものである。次の 1～3 の条件に沿った形状となるように、サンプルコードの空欄を適切に埋め、その動作を確認しなさい。

- 1) 図 1 のように、 x, y 方向ともに 2 倍の大きさの正方形を描きなさい。
- 2) 図 2 のように、 y 軸に対して線対称な図形を描きなさい。
- 3) 図 3 のように、原点に対して点対称でかつ大きさが 2 倍の図形を描きなさい。

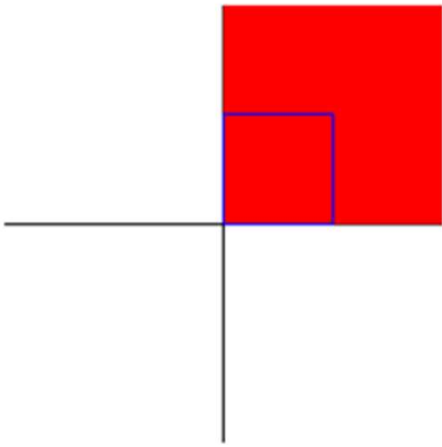


図 1

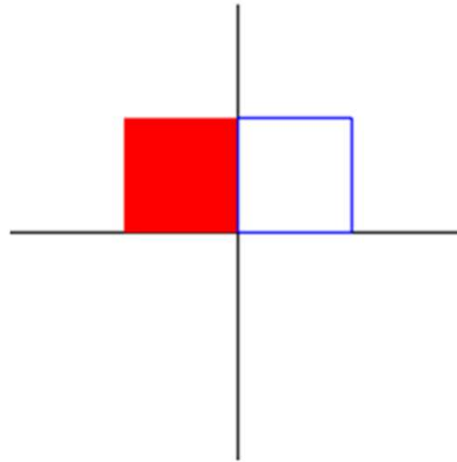


図 2

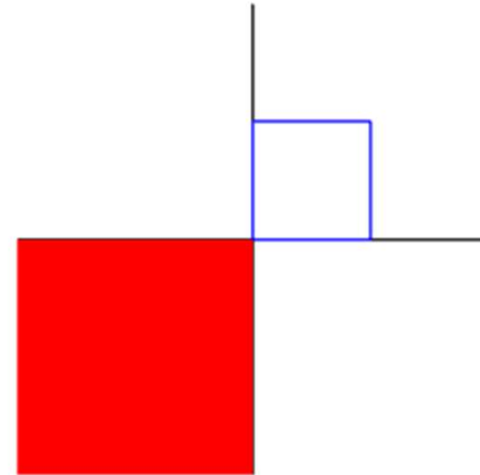


図 3

本日の演習

課題 3-5 合成変換（1）

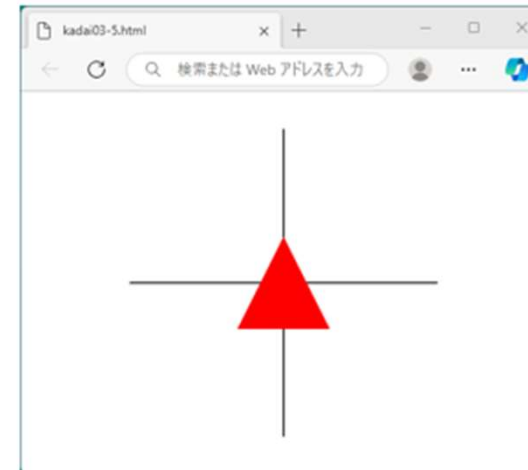
次に示す WebGL のサンプルコード **sample03-5** は、右下の三角形を描くものである。この三角形に対して、次の図形変換を順番に与え、変換後の図形を表示しなさい。

- 1) 三角形をx軸に沿って左に 0.5, y軸に沿って上に 0.2 平行移動しなさい。
- 2) 三角形を原点周りに 90° 回転しなさい。
- 3) 三角形をx軸方向に 2 倍拡大しなさい。

sample03-5

```
// 三角形の描画
gl.color(1.0, 0.0, 0.0, 1.0);
gl.begin(gl.TRIANGLES);
    gl.vertex(-0.3, -0.3, 0.0);
    gl.vertex(0.3, -0.3, 0.0);
    gl.vertex(0.0, 0.3, 0.0);
gl.end();
```

ただし、背景色等の設計、座標軸の描画については記載省略



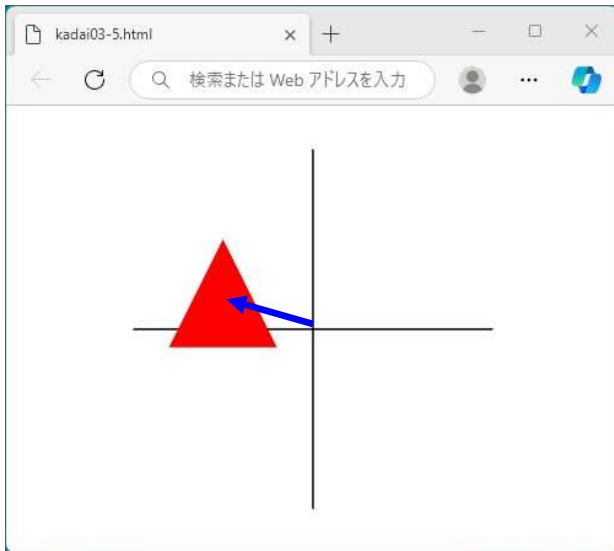
本日の演習

課題 3-5 合成変換（1）

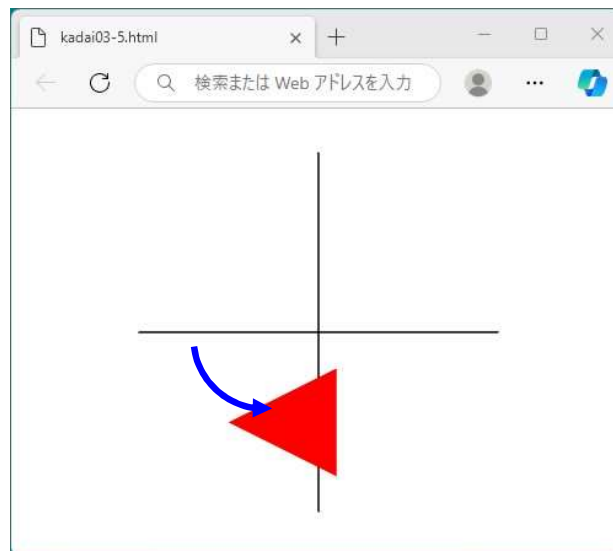
次に示す WebGL のサンプルコード **sample03-5** は、右下の三角形を描くものである。この三角形に対して、次の図形変換を順番に与え、変換後の図形を表示しなさい。

- 1) 三角形を x 軸に沿って左に 0.5, y 軸に沿って上に 0.2 平行移動しなさい。
- 2) 三角形を原点周りに 90° 回転しなさい。
- 3) 三角形を x 軸方向に 2 倍拡大しなさい。

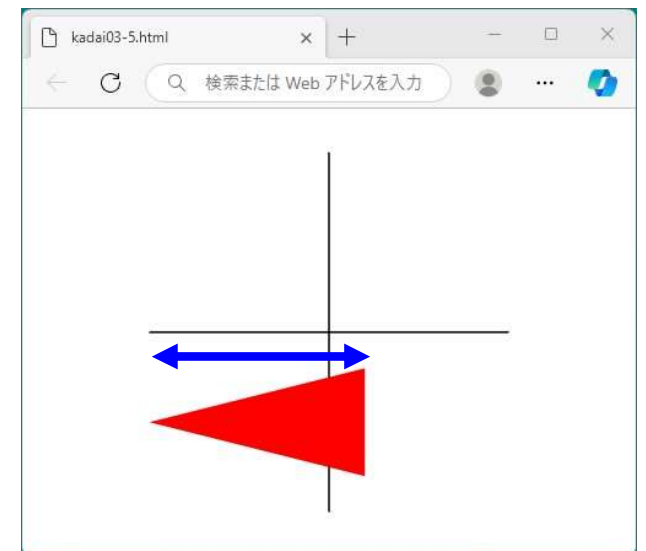
① 平行移動



② 回転



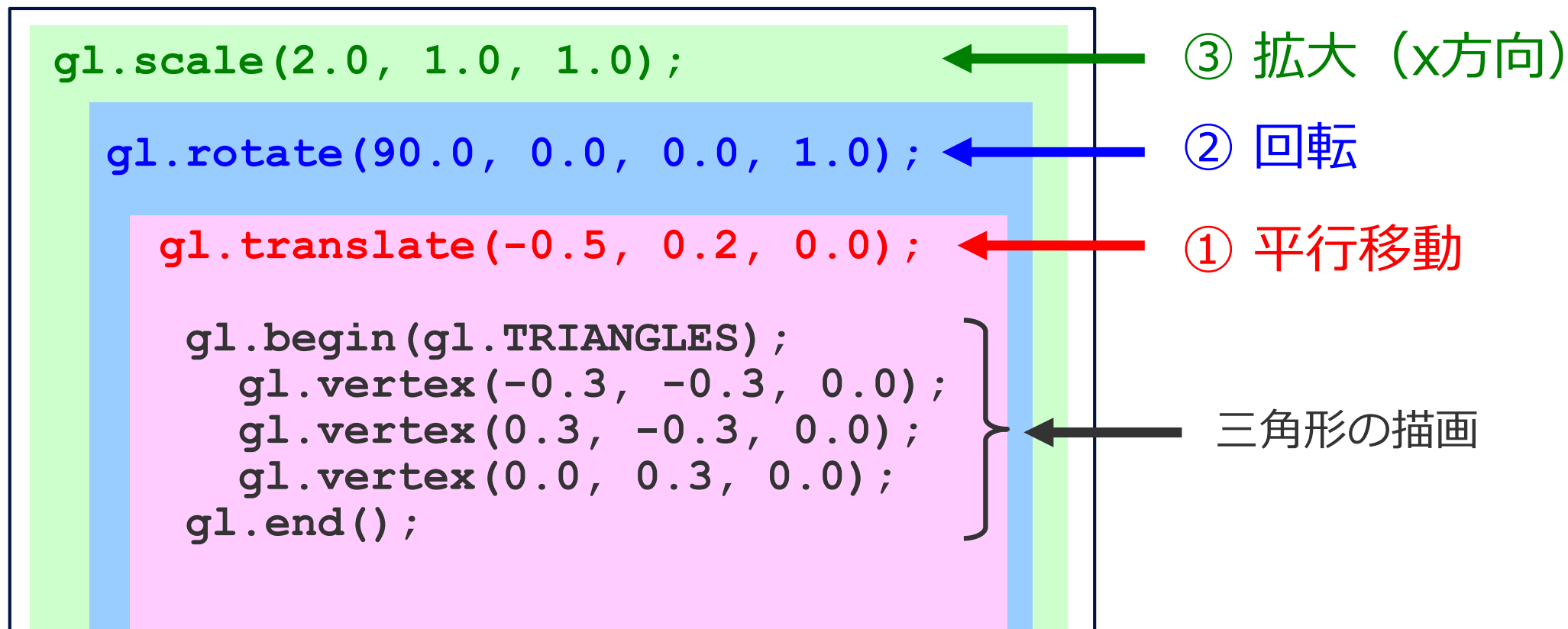
③ 拡大（x方向）



変換の順番

図形変換命令は、以降の図形に影響を与える

→ **変換の順番に対して変換関数は逆順となる**



行列の計算のイメージ

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \text{③ 拡大} \end{bmatrix} \begin{bmatrix} \text{② 回転} \end{bmatrix} \begin{bmatrix} \text{① 平行移動} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{\text{変換前}}$$

変換後

順番が違うと結果が異なる

本日の演習

課題 3-7 合成変換 (3)

サンプルコード **sample03-5** の三角形にせん断を適用し, x 方向に $\alpha = 45[\text{deg}]$ 図形を傾けなさい。

Hint: lightgl.js には, せん断の変換命令はないので, 自分でせん断後の頂点を計算する。その計算結果を頂点 `gl.vertex` に入力し図形を表示する。

