

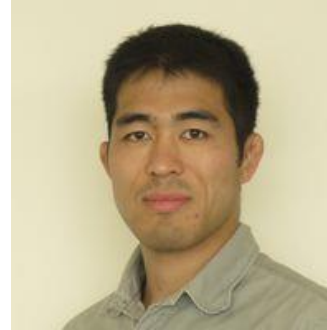
# コンピュータグラフィックス論

2017年4月6日

高山 健志

# 教員紹介

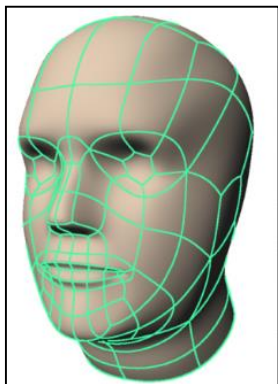
- 高山 健志 (国立情報学研究所 助教)
  - <http://research.nii.ac.jp/~takayama/>
  - [takayama@nii.ac.jp](mailto:takayama@nii.ac.jp)
- 蜂須賀 恵也 (創造情報学専攻 講師)
  - <http://www.ci.i.u-tokyo.ac.jp/~hachisuka/>
  - [thachisuka@siggraph.org](mailto:thachisuka@siggraph.org)
- 安東 遼一 (国立情報学研究所 助教)
  - <https://scholar.google.com/citations?user=Ag3RwxUAAAAJ&hl=en>



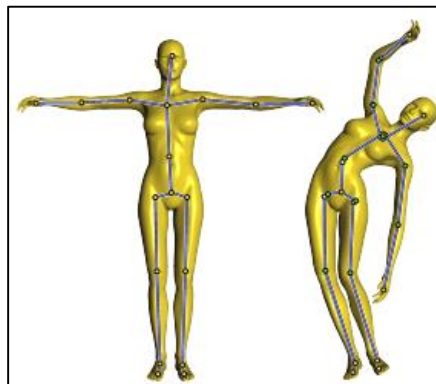
TA : 盧 承鐸 (五十嵐研)  
<https://sites.google.com/site/seungtaknoh/home>  
[seungtak.noh@gmail.com](mailto:seungtak.noh@gmail.com)

# 講義の概要

モデリング



アニメーション



レンダリング



画像処理



- 各トピックについて2~3回、計12回 (休講2回)
- レンダリングの回は蜂須賀先生が担当
- 流体アニメーションの回は安東先生が担当

# 成績評価の方法

- プログラミング課題のみ
  - 試験はしない、出席も取らない
- 「基本課題」と「発展課題」の二種類
  - 基本課題：各トピックごとに一つ (計4個)、とても簡単
  - 発展課題：やる気のある人向け
- 締切：授業期間中 (7月末)
- 評価基準
  - 1個の課題を提出 → 単位を出す最低ライン
  - 4個の課題を提出 → 良以上
  - 工夫の有無や全体のバランスを加味して優・優上の分布を決める
- 提出方法等の詳細は後で説明

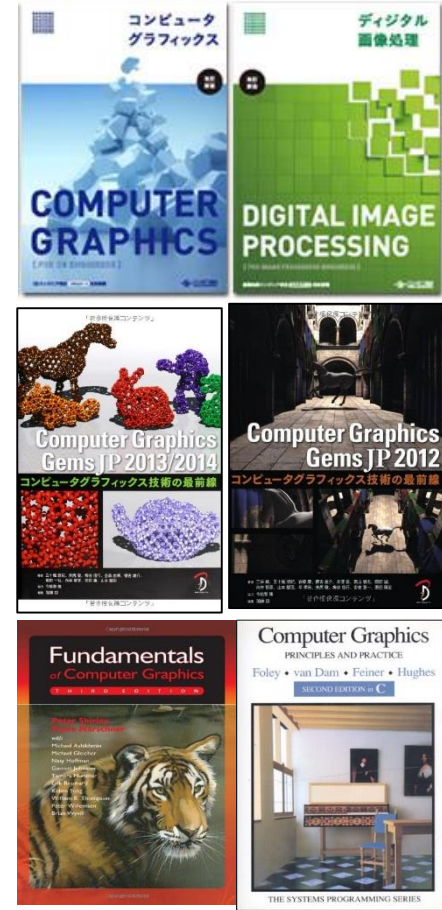
# 講義情報

- 講義ページ

- <http://research.nii.ac.jp/~takayama/teaching/utokyo-iscg-2017/>

- 参考書

- コンピュータグラフィックス 改訂新版 (9784903474496)
- デジタル画像処理 改訂新版 (9784903474502)
- CG Gems JP 2012 (9784862461858)
- CG Gems JP 2013/2014 (9784862462190)
- Fundamentals of Computer Graphics (9781568814698)
- Computer Graphics: Principles and Practice in C (9780201848403)



# 研究紹介 (安東先生)

# 東大でCGを専門とする新進気鋭の先生



蜂須賀先生@創造情報学  
[thachisuka@siggraph.org](mailto:thachisuka@siggraph.org)  
(レンダリングの回を担当)



楽 (がく) 先生@複雑理工  
[yonghao@acm.org](mailto:yonghao@acm.org)

- CGに興味のある学生を絶賛募集中！

# 座標変換



# 線形変換

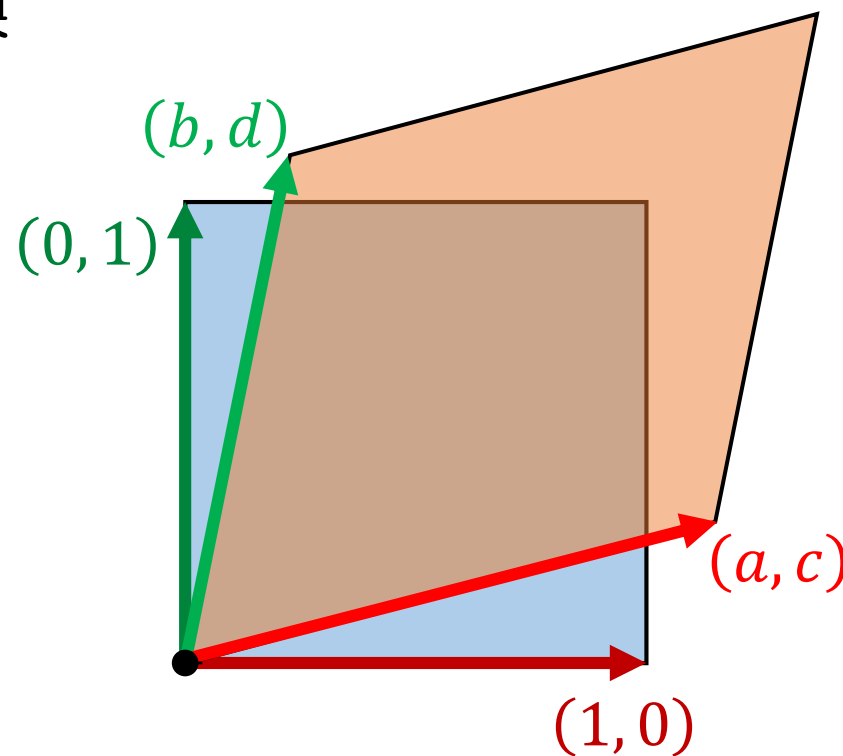
2Dの場合：  $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$       3Dの場合：  $\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

- イメージ：座標軸を移すような変換

$$\begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

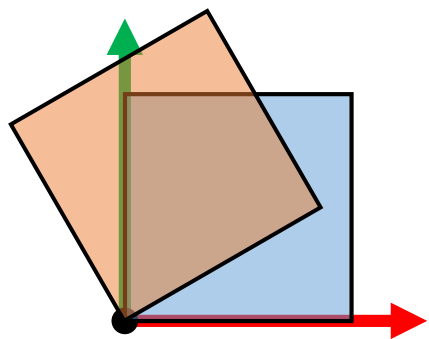
$$\begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- 原点は動かない



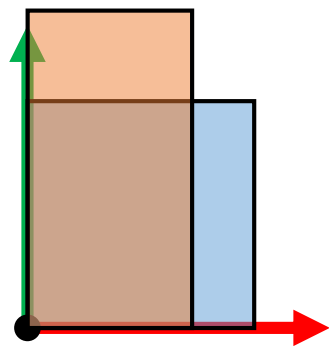
# いろいろな線形変換

回転



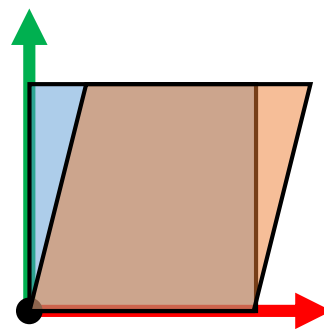
$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

スケーリング



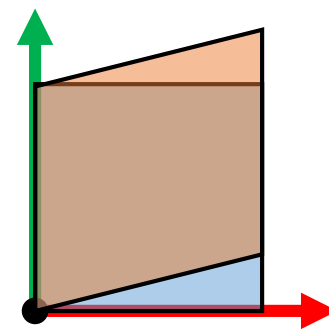
$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

せん断 (X方向)



$$\begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$$

せん断 (Y方向)



$$\begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix}$$

# 線形変換＋平行移動＝アフィン変換

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad \Leftrightarrow \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

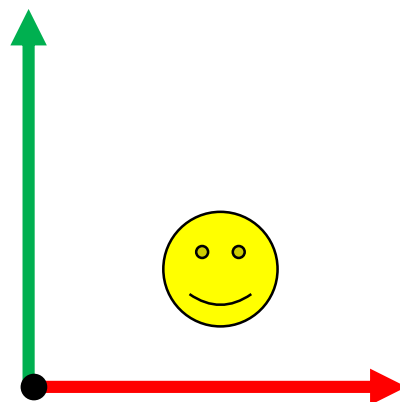
- **同次座標**：2D (3D) 座標を表すのに、便宜的に3D (4D) ベクトルを使う
- 線形変換と平行移動を、行列の積として同じように表せる！
  - 実装上都合が良い

# アフィン変換の合成

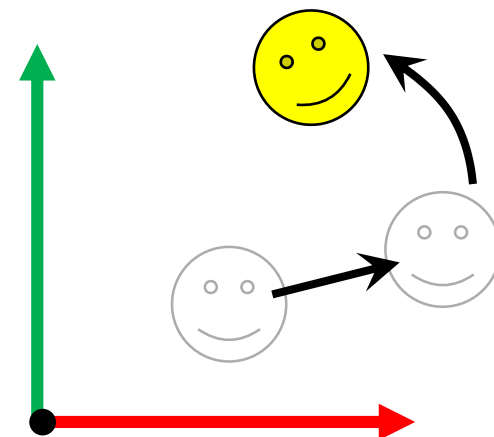
- 変換行列を掛けるだけ
- 掛ける順番に注意！

$$R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

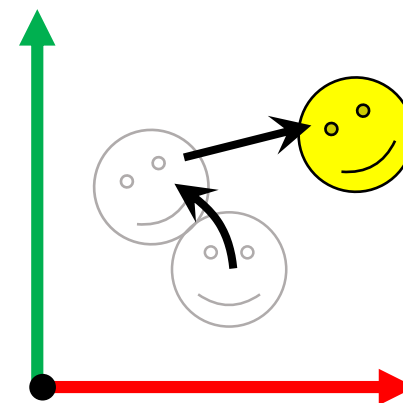
$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{x}' = R T \mathbf{x}$$



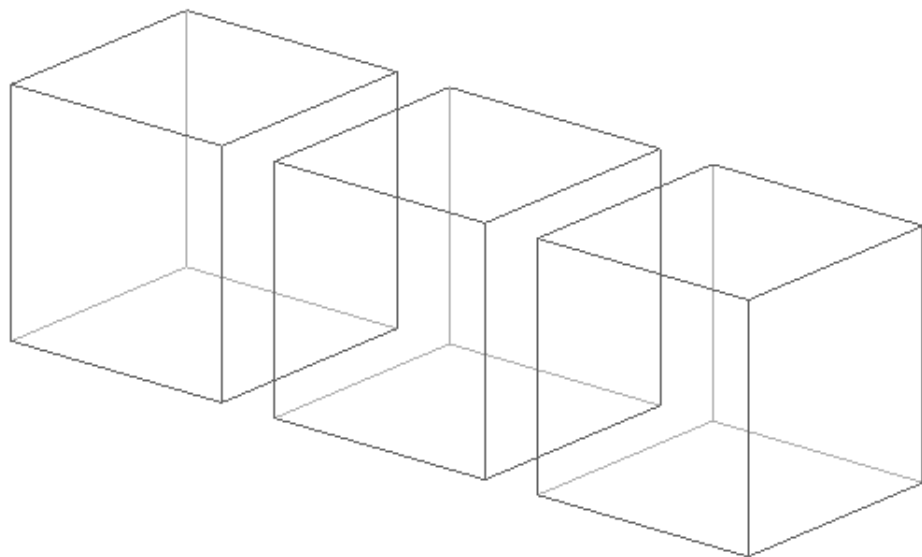
$$\mathbf{x}' = T R \mathbf{x}$$



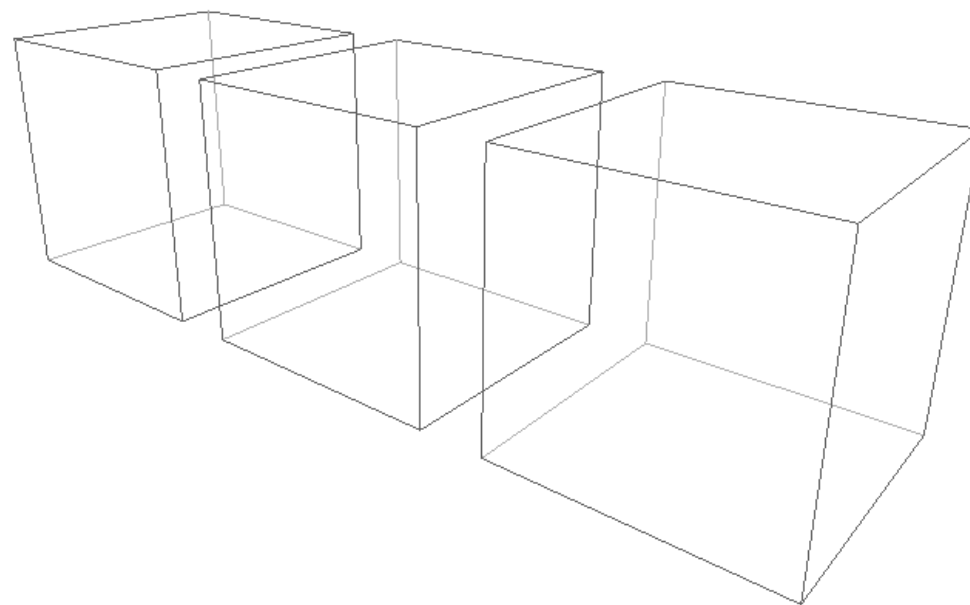
# 同次座標のもう一つの役割：透視投影

- いわゆる遠近法

- 物体のスクリーン上の見かけの大きさが、視点からの距離に反比例



平行投影



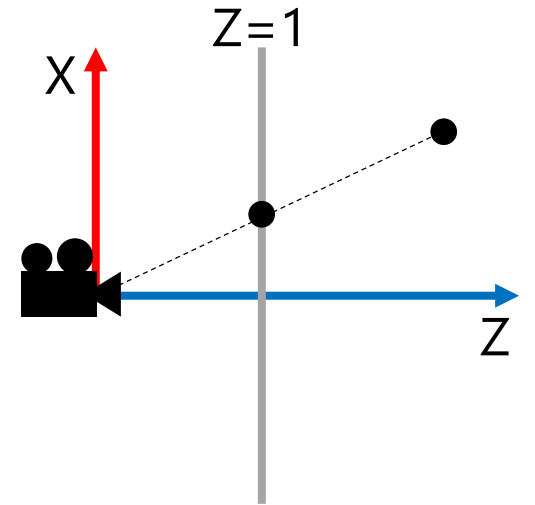
透視投影

# 同次座標による透視投影の実現

- $w \neq 0$  のとき、4D同次座標  $(x, y, z, w)$  は3D空間座標  $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$  を表す
  - 背景に奥深い理論 (cf. Wikipedia)
- 視点を原点に置き、スクリーンを平面  $Z=1$  とするとき、 $(p_x, p_y, p_z)$  を  $(w_x, w_y) = (\frac{p_x}{p_z}, \frac{p_y}{p_z})$  に投影したい

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z + 1 \\ \mathbf{p_z} \end{bmatrix} \equiv \begin{bmatrix} p_x/p_z & \rightarrow w_x \\ p_y/p_z & \rightarrow w_y \\ 1 + 1/p_z & \rightarrow w_z \\ 1 \end{bmatrix}$$

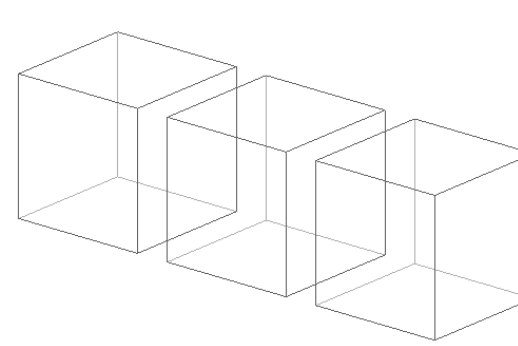
射影変換



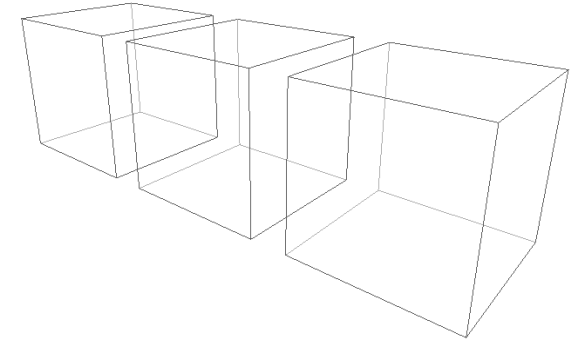
- $w_z$  (深度値) は、前後関係の判定に使われる → Zバッファ法

# 平行投影

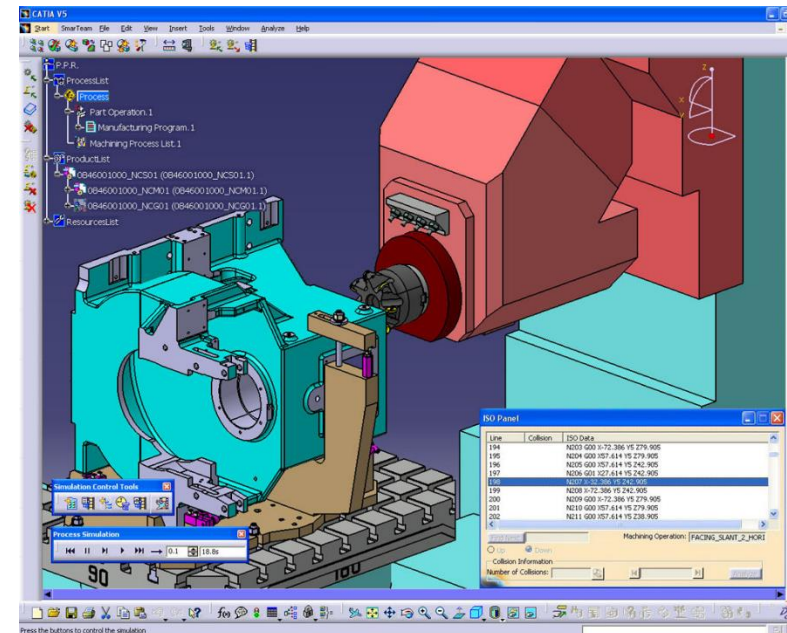
- 物体の見かけ上の大きさが、視点からの距離に影響されない
- 単にZ座標を無視するだけ
- 製図でよく使われる



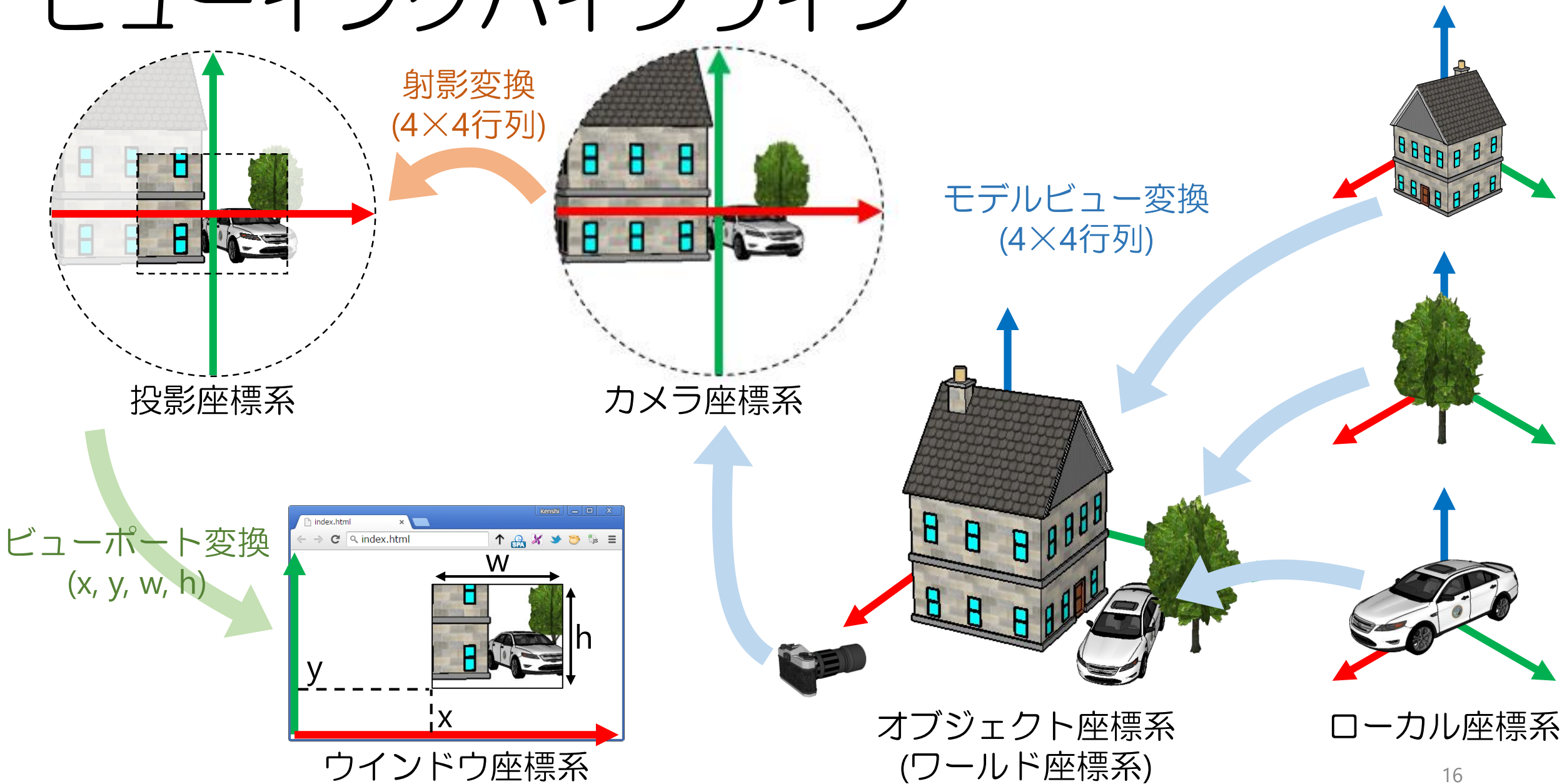
平行投影



透視投影



# ビューイングパイプライン





# 古典的なOpenGLコード

```
glViewport(0, 0, 640, 480);
```

} ビューポート変換

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

```
gluPerspective(
```

```
    45.0,          // field of view
```

```
    640 / 480,     // aspect ratio
```

```
    0.1, 100.0); // depth range
```

} 射影変換

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

```
gluLookAt(
```

```
    0.5, 0.5, 3.0, // view point
```

```
    0.0, 0.0, 0.0, // focus point
```

```
    0.0, 1.0, 0.0); // up vector
```

} モデルビュー変換

```
glBegin(GL_LINES);
```

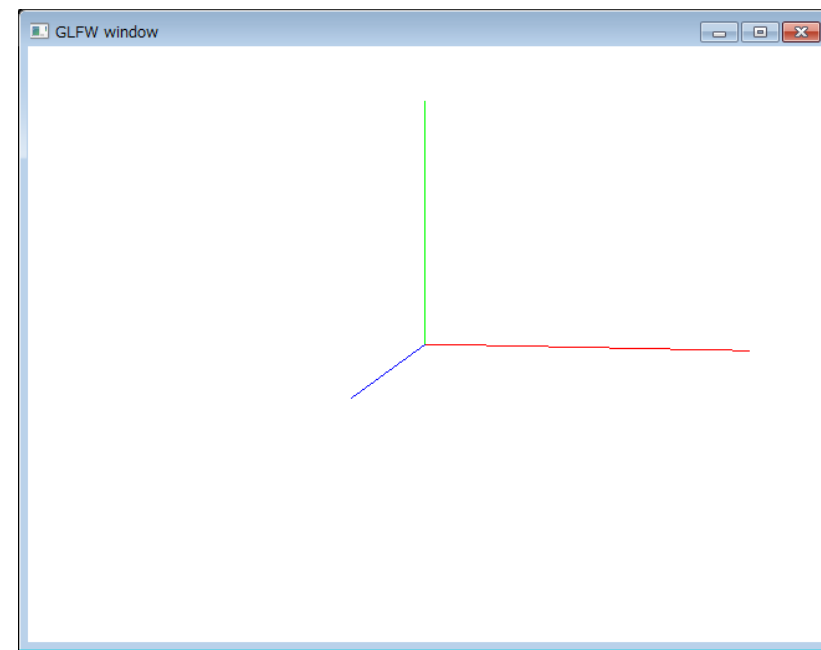
```
glColor3d(1, 0, 0); glVertex3d(0, 0, 0); glVertex3d(1, 0, 0);
```

```
glColor3d(0, 1, 0); glVertex3d(0, 0, 0); glVertex3d(0, 1, 0);
```

```
glColor3d(0, 0, 1); glVertex3d(0, 0, 0); glVertex3d(0, 0, 1);
```

```
glEnd();
```

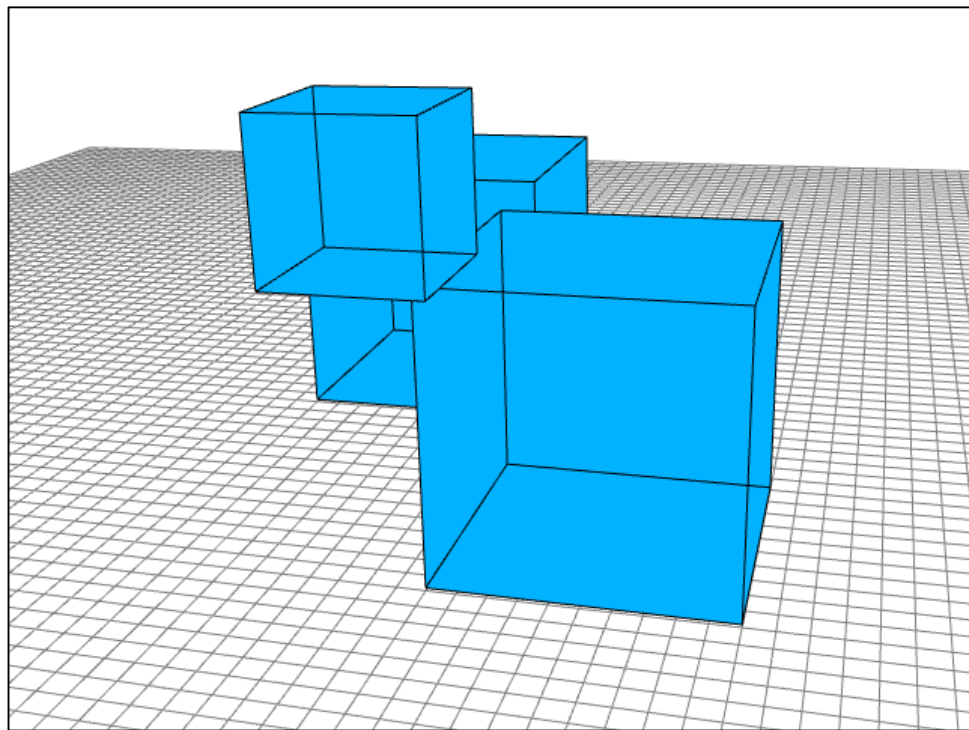
} シーン内容



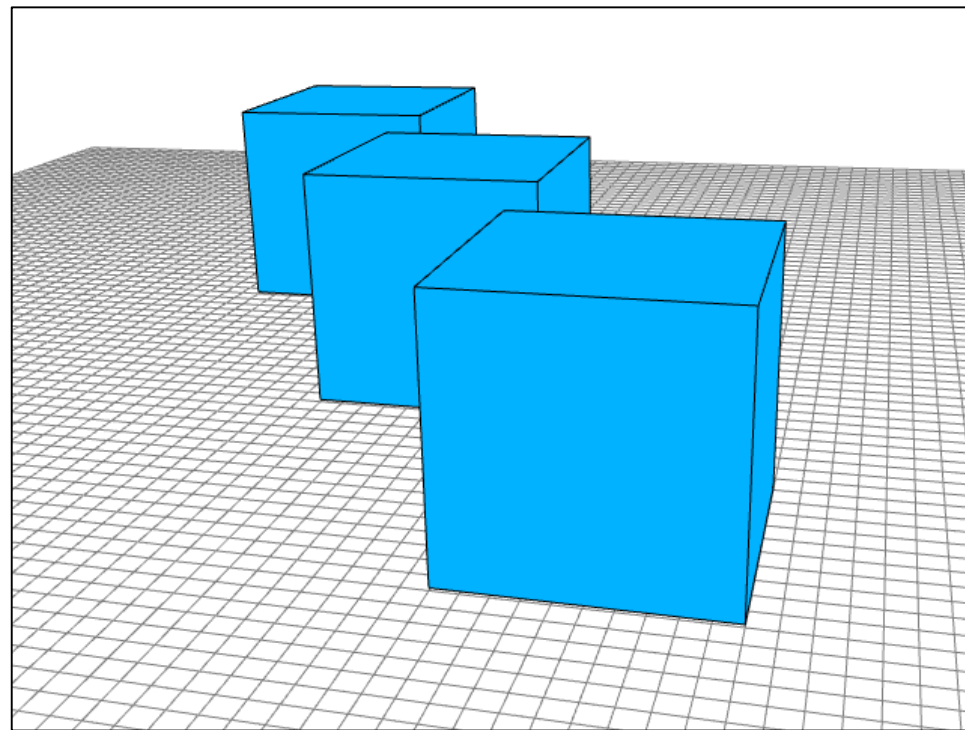
描画結果

# Zバッファ法

# 隠面消去



隠面消去なし



隠面消去あり

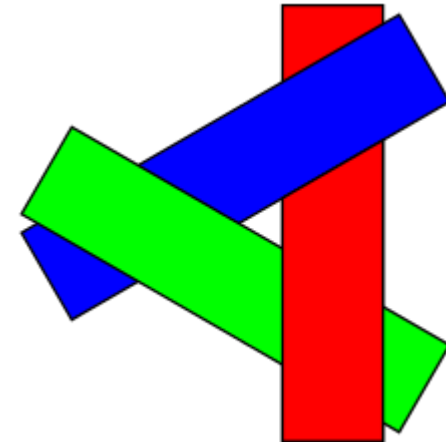
- CGの古典的な問題

# 画家のアルゴリズム

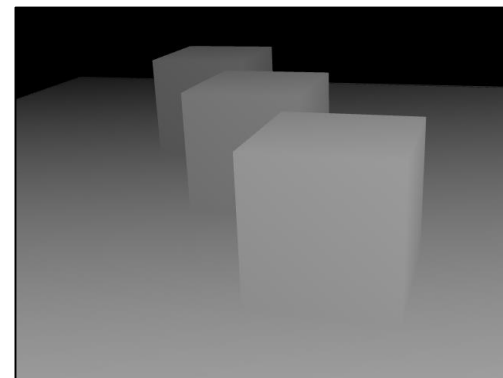
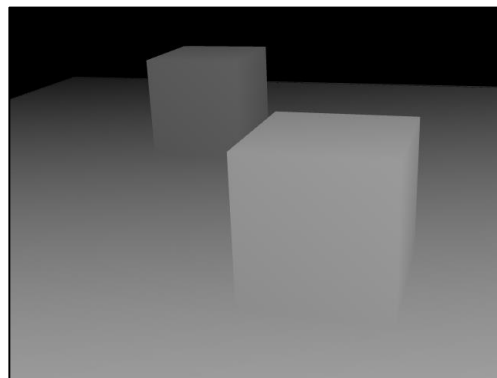
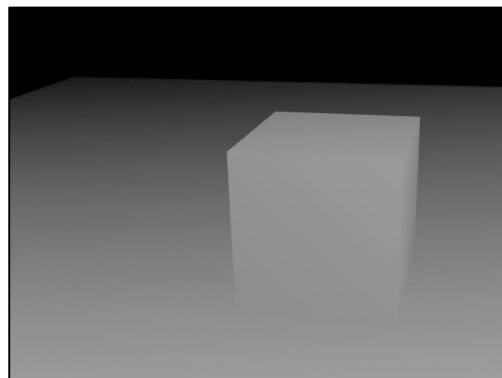
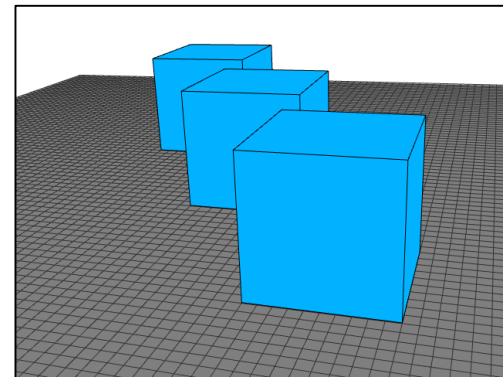
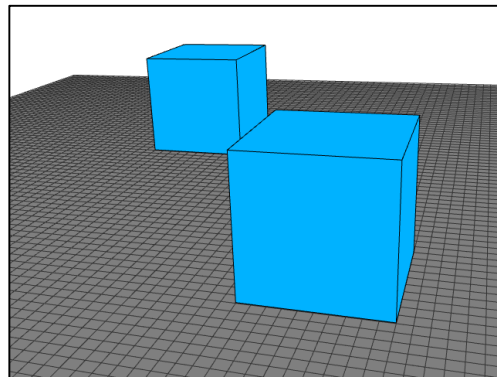
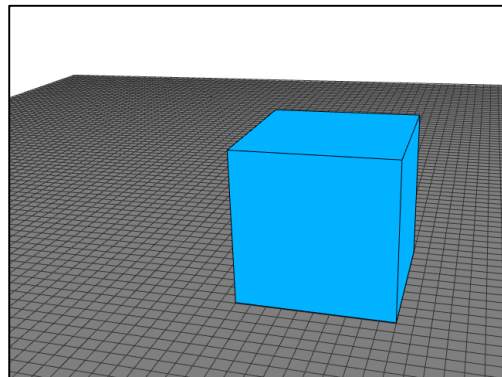
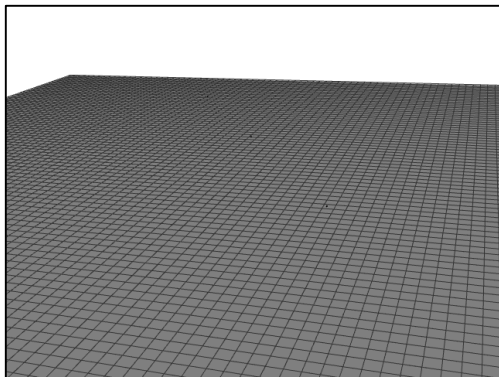
- 物体を視点からの距離でソートし、遠くものから描画



- 原理的に対応できないケースが多数
  - ソート方法も自明ではない



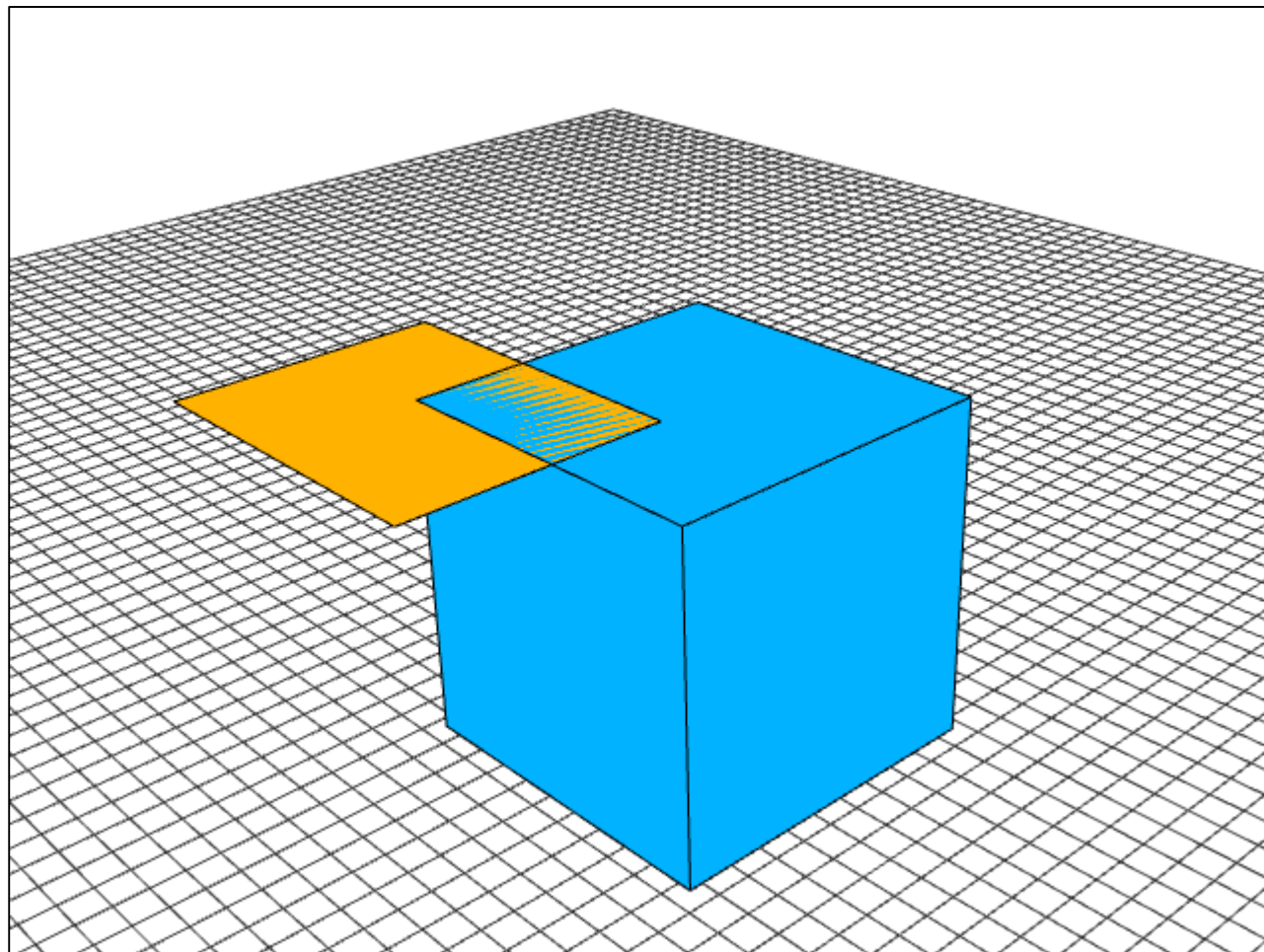
# Zバッファ法



- 各ピクセルごとに、視点から物体までの距離 (深度) を記録
- メモリ消費は大きいですが、現在のスタンダード

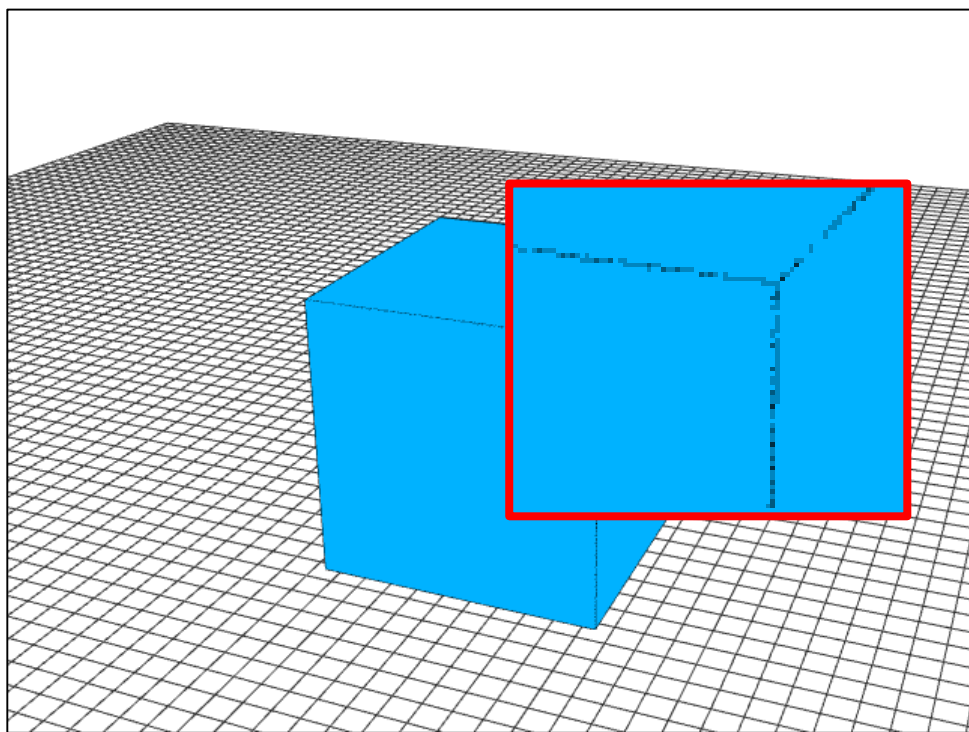
# Zバッファの注意点：Z-fighting

- 同一の位置に複数のポリゴンを描画
- 前後の判定がそもそも不可能
- 丸め誤差による変な模様

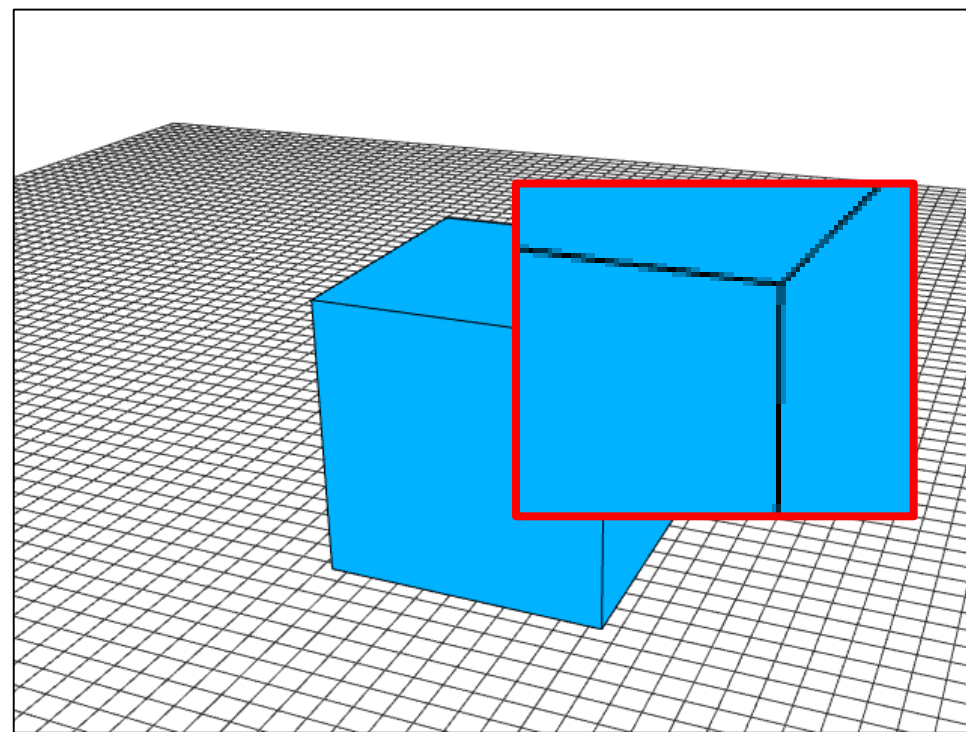


# Zバッファの注意点：面と辺の同時描画

- 専用のOpenGLトリック：glPolygonOffset



polygon offset なし



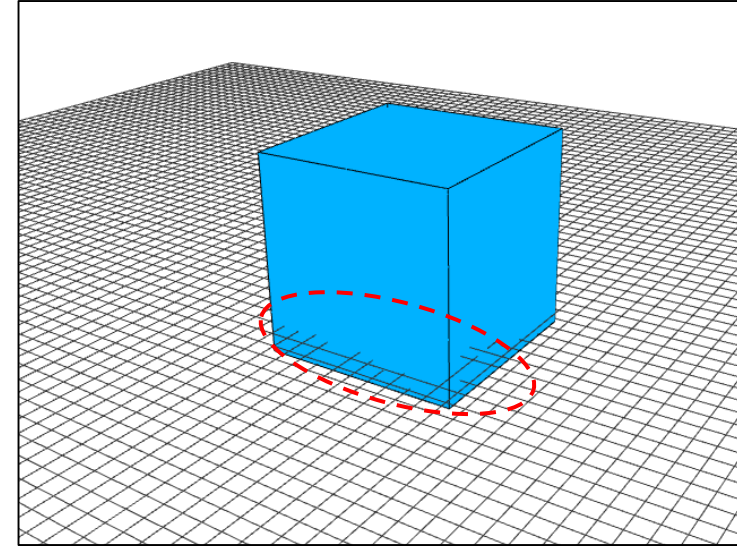
polygon offset あり



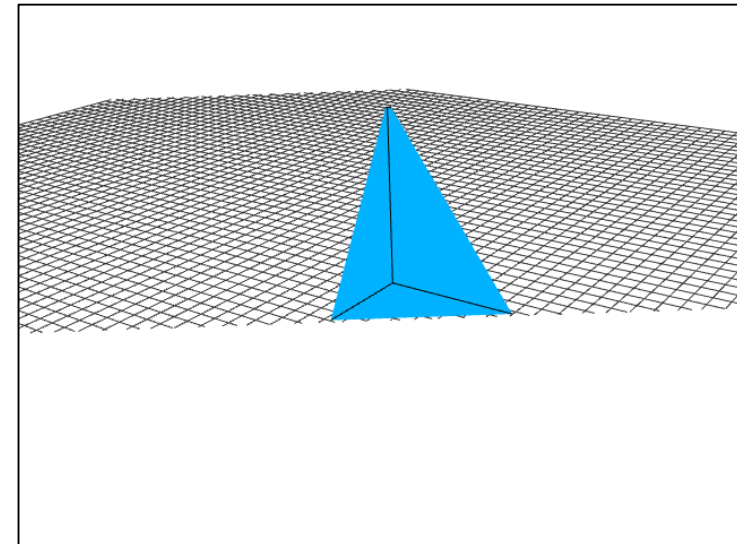
# Zバッファの注意点：深度値の範囲

```
gluPerspective(  
    45.0,          // field of view  
    640 / 480,     // aspect ratio  
    0.1, 1000.0); // zNear, zFar
```

- Zバッファのビット数は固定
  - 16~24bit程度
- 範囲を大きく取る
  - ➔ 描画範囲は広くなるが、精度が下がる
- 範囲を小さく取る
  - ➔ 精度は上がるが、描画範囲は狭くなる(クリッピングされる)



zNear=0.0001  
zFar =1000



zNear=50  
zFar =100



# ラスタライゼーション vs レイトレーシング

主な用途

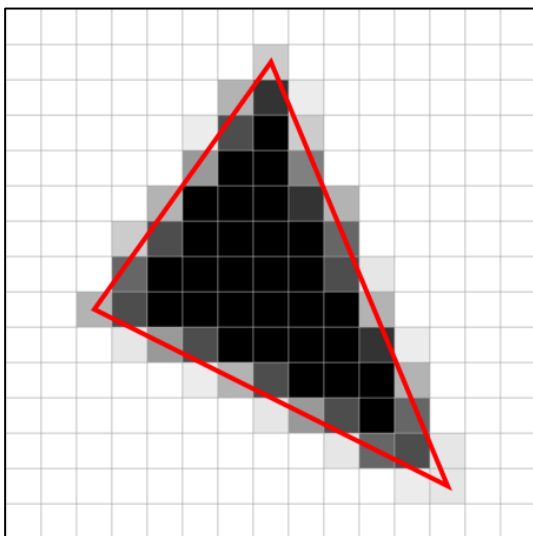
リアルタイムCG (ゲーム)

高品質CG (映画)

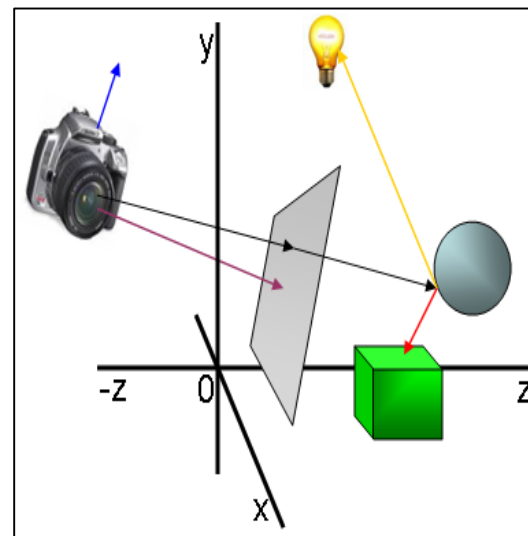
考え方

ポリゴン単位の処理

ピクセル (レイ) 単位の処理



一枚のポリゴンが  
複数のピクセル  
を更新



一本のレイが  
複数のポリゴン  
と交差

隠面消去

Zバッファ法  
(OpenGL / DirectX)

自然と実現される

詳しくは蜂須賀先生の回で

クォータニオン

# 任意軸周りの回転

- 様々な場面で必要 (e.g. カメラ操作)

X軸周り  $R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$

Y軸周り  $R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$

Z軸周り  $R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

任意軸  
周り  $R = \begin{bmatrix} \cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\ u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\ u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_z u_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2 (1 - \cos \theta) \end{bmatrix}$

$(u_x, u_y, u_z)$  : 回転軸ベクトル

- 行列表現の欠点

- 無駄に複雑！

- 本来は2自由度 (軸方向) + 1自由度 (角度) = 3自由度で表されるべき

- 補間 (混ぜ合わせ) が上手くできない

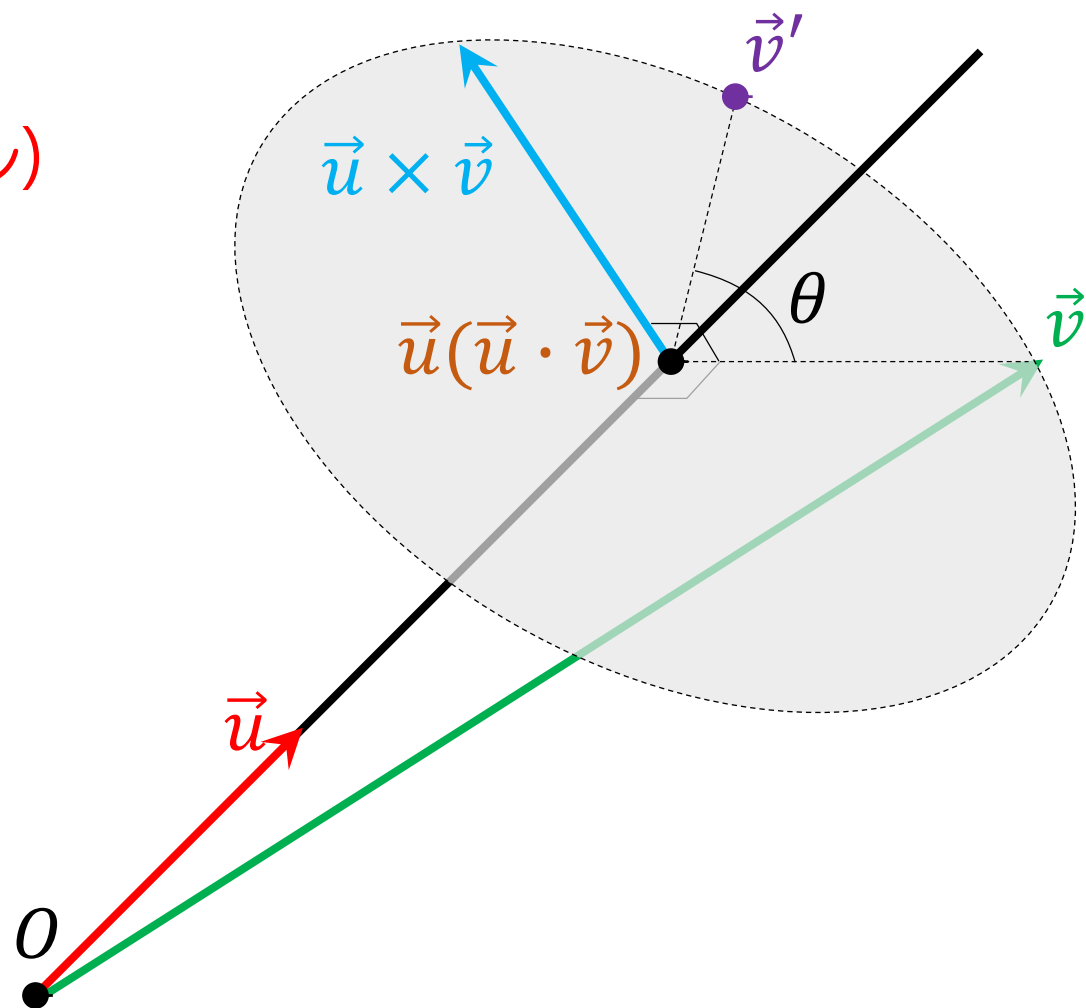
# 任意軸周り回転の幾何

$\vec{u}$ : 軸 (単位ベクトル)

$\theta$ : 角度

$\vec{v}$ : 入力座標

$\vec{v}'$ : 出力座標



$$\vec{v}' = (\vec{v} - \vec{u}(\vec{u} \cdot \vec{v})) \cos \theta + (\vec{u} \times \vec{v}) \sin \theta + \vec{u}(\vec{u} \cdot \vec{v})$$

# 複素数とクォータニオン (四元数)

- 複素数

- $\mathbf{i}^2 = -1$
- $\mathbf{c} = (a, b) := a + b \mathbf{i}$
- $\mathbf{c}_1 \mathbf{c}_2 = (a_1, b_1)(a_2, b_2) = a_1 a_2 - b_1 b_2 + (a_1 b_2 + b_1 a_2) \mathbf{i}$

- クォータニオン

- $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$ 
  - $\mathbf{ij} = -\mathbf{ji} = \mathbf{k}, \quad \mathbf{jk} = -\mathbf{kj} = \mathbf{i}, \quad \mathbf{ki} = -\mathbf{ik} = \mathbf{j}$  可換ではない！

- $\mathbf{q} = (a, b, c, d) := a + b \mathbf{i} + c \mathbf{j} + d \mathbf{k}$

- $\mathbf{q}_1 \mathbf{q}_2 = (a_1, b_1, c_1, d_1)(a_2, b_2, c_2, d_2)$

$$= (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) \mathbf{i}$$

$$+ (a_1 c_2 + c_1 a_2 + d_1 b_2 - b_1 d_2) \mathbf{j} + (a_1 d_2 + d_1 a_2 + b_1 c_2 - c_1 b_2) \mathbf{k}$$

# スカラー＋3Dベクトルによる表記

- $\mathbf{q} = a + b \mathbf{i} + c \mathbf{j} + d \mathbf{k} := a + (b, c, d) = a + \vec{v}$
- $\mathbf{q}_1 \mathbf{q}_2 = (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) \mathbf{i}$   
 $+ (a_1 c_2 + c_1 a_2 + d_1 b_2 - b_1 d_2) \mathbf{j} + (a_1 d_2 + d_1 a_2 + b_1 c_2 - c_1 b_2) \mathbf{k}$   
 $= (a_1 a_2 - \vec{v}_1 \cdot \vec{v}_2) + a_1 \vec{v}_2 + a_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2$

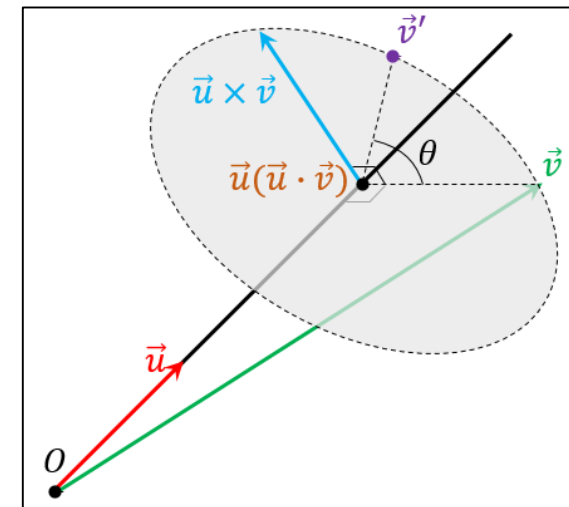
# クォータニオンによる回転

$$q = \cos \frac{\alpha}{2} + \vec{u} \sin \frac{\alpha}{2}$$

注： $\vec{u}$ は単位ベクトル

$$\begin{aligned}\vec{v}' &= q\vec{v}q^{-1} = \left(\cos \frac{\alpha}{2} + \vec{u} \sin \frac{\alpha}{2}\right) \vec{v} \left(\cos \frac{\alpha}{2} - \vec{u} \sin \frac{\alpha}{2}\right) \\&= \vec{v} \cos^2 \frac{\alpha}{2} + (\vec{u}\vec{v} - \vec{v}\vec{u}) \sin \frac{\alpha}{2} \cos \frac{\alpha}{2} - \vec{u}\vec{v}\vec{u} \sin^2 \frac{\alpha}{2} \\&= \vec{v} \cos^2 \frac{\alpha}{2} + 2(\vec{u} \times \vec{v}) \sin \frac{\alpha}{2} \cos \frac{\alpha}{2} - (\vec{v}(\vec{u} \cdot \vec{u}) - 2\vec{u}(\vec{u} \cdot \vec{v})) \sin^2 \frac{\alpha}{2} \\&= \vec{v}(\cos^2 \frac{\alpha}{2} - \sin^2 \frac{\alpha}{2}) + (\vec{u} \times \vec{v})(2\sin \frac{\alpha}{2} \cos \frac{\alpha}{2}) + \vec{u}(\vec{u} \cdot \vec{v})(2\sin^2 \frac{\alpha}{2}) \\&= \vec{v} \cos \alpha + (\vec{u} \times \vec{v}) \sin \alpha + \vec{u}(\vec{u} \cdot \vec{v})(1 - \cos \alpha) \\&= (\vec{v} - \vec{u}(\vec{u} \cdot \vec{v})) \cos \alpha + (\vec{u} \times \vec{v}) \sin \alpha + \vec{u}(\vec{u} \cdot \vec{v})\end{aligned}$$

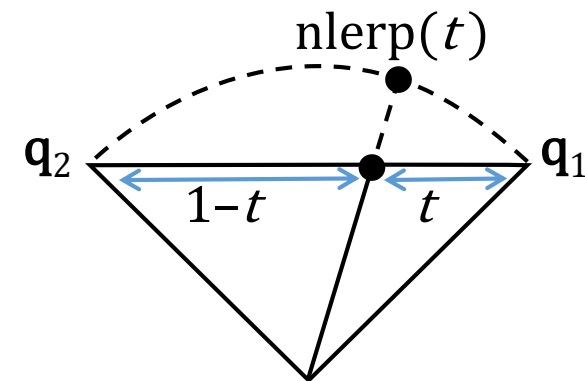
- 背景には面白い理論 (cf. Wikipedia)



# クォータニオンによる回転の補間

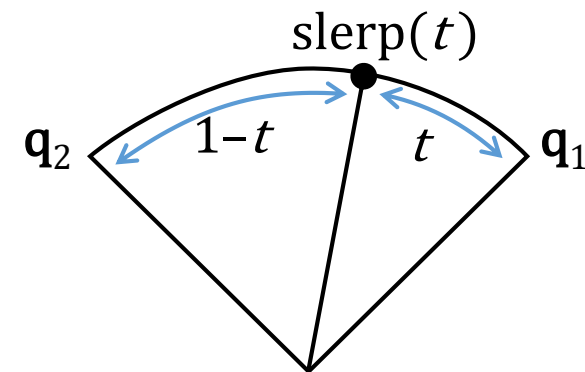
- 線形補間＋正規化 (nlerp)

- $\text{nlerp}(\mathbf{q}_1, \mathbf{q}_2, t) := \text{normalize}((1-t)\mathbf{q}_1 + t\mathbf{q}_2)$
- ☺計算が少ない、☹角速度が一定でない



- 球面線形補間 (slerp)

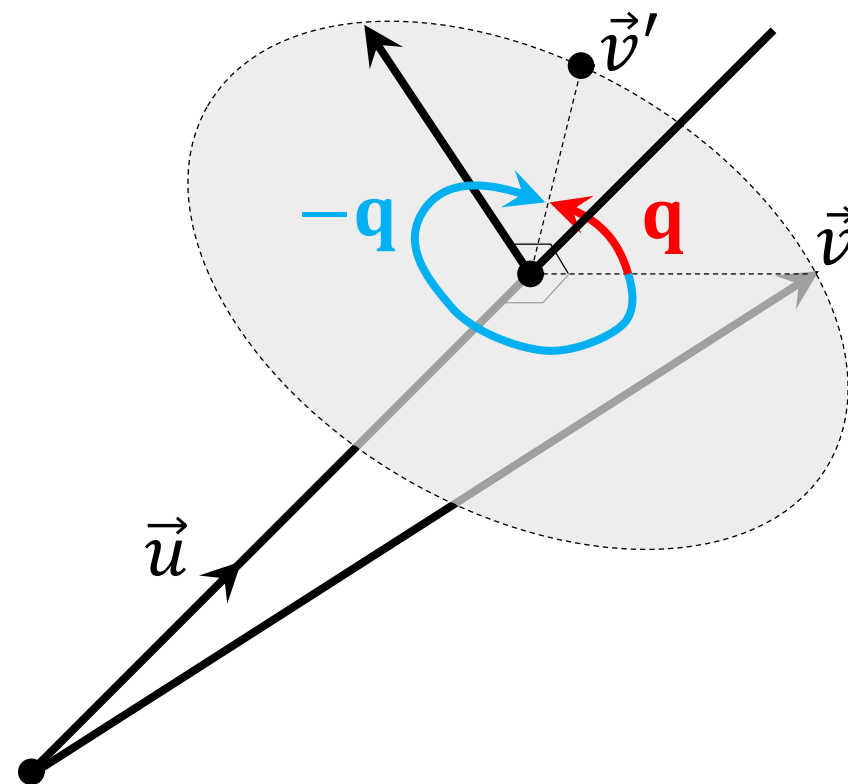
- $\Omega = \cos^{-1}(\mathbf{q}_1 \cdot \mathbf{q}_2)$
- $\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) := \frac{\sin(1-t)\Omega}{\sin \Omega} \mathbf{q}_1 + \frac{\sin t\Omega}{\sin \Omega} \mathbf{q}_2$
- ☹計算が多い、☺角速度が一定





# 正負のクォータニオン

- 回転角が $\theta$ のクォータニオン：
  - $\mathbf{q} = \cos \frac{\theta}{2} + \vec{u} \sin \frac{\theta}{2}$
- 回転角が $\theta - 2\pi$ のクォータニオン：
  - $\cos \frac{\theta - 2\pi}{2} + \vec{u} \sin \frac{\theta - 2\pi}{2} = -\mathbf{q}$
- $\mathbf{q}_1$ から $\mathbf{q}_2$ へ補間する際、 $\mathbf{q}_1 \cdot \mathbf{q}_2$  が負であれば $\mathbf{q}_2$ を反転してから補間する
  - そうしないと補間過程が最短でなくなる



# 課題提出の方法について

# リアルタイムCG実装の選択肢

- GPUのAPIとして大きく2種類：



- 異なる設計思想
- 主要なプログラミング言語では大抵両方利用できる

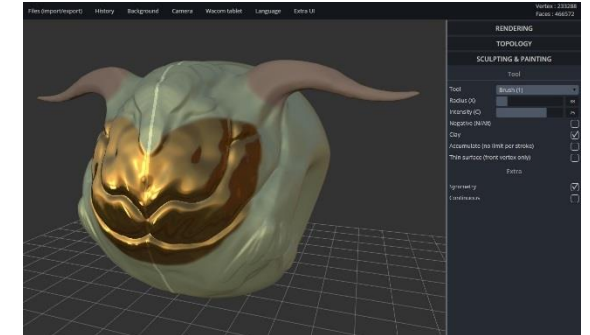
- システムや言語依存な部分にも多くの選択肢

- ウィンドウ生成、イベント処理、画像ファイルの読み書き、...
- 様々なライブラリ：
  - GUI：GLUT (C), GLFW (C), SDL (C), Qt (C++), MFC (C++), wxWidgets (C++), Swing (Java), ...
  - 画像：libpng, OpenCV, ImageMagick

- 往々にして開発・実行環境の準備が面倒

# WebGL™ = JavaScript + OpenGL®

- 多くのブラウザ (モバイル含む) で動く
- HTMLベース➡マルチメディアやGUIを簡単に扱える
- コンパイル不要！
  - 開発時の試行錯誤が非常に手軽
- 実行速度は多少落ちる
- 最近注目が高まっている



# WebGL開発のハードル：OpenGL ES (for Embedded Systems)

- OpenGL 1.xのAPIが使えない！

- 理由：

- 処理効率の悪さ
- ハードウェア開発側の負担

イミディエイトモード  
多角形の描画  
光と材質  
座標変換行列  
ディスプレイリスト  
デフォルトのシェーダ

glBegin, glVertex, glColor, glTexCoord  
GL\_QUADS, GL\_POLYGON  
glLight, glMaterial  
GL\_MODELVIEW, GL\_PROJECTION  
glNewList

- 使用可能なAPI：

大きな配列データをまとめてGPUに送り、自前シェーダで描画

シェーダの作成

glCreateShader, glShaderSource,  
glCompileShader, glCreateProgram,  
glAttachShader, glLinkProgram,  
glUseProgram

シェーダ変数の管理

glGetAttribLocation,  
glEnableVertexAttribArray,  
glGetUniformLocation, glUniform

配列の管理

glCreateBuffer, glBindBuffer,  
glBufferData, glVertexAttribPointer

配列の内容を描画

glDrawArrays

## #include <GL/glut.h> C / OpenGL 1.x

```
void disp( void ) {
    float f;
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    for(f = 0 ; f < 1 ; f += 0.1) {
        glColor3f(f , 0 , 0);
        glCallList(1);
    }
    glPopMatrix();
    glFlush();
}

void setDispList( void ) {
    glNewList(1, GL_COMPILE);
    glBegin(GL_POLYGON);
    glVertex2f(-1.2 , -0.9);
    glVertex2f(0.6 , -0.9);
    glVertex2f(-0.3 , 0.9);
    glEnd();
    glTranslatef(0.1 , 0 , 0);
    glEndList();
}

int main(int argc , char ** argv) {
    glutInit(&argc , argv);
    glutInitWindowSize(400 , 300);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("Kitty on your lap");
    glutDisplayFunc(disp);
    setDispList();
    glutMainLoop();
}
```



<http://wisdom.sakura.ne.jp/system/opengl/gl20.html>

## WebGL

```
<html><head>
<title>Learning WebGL &mdash; lesson 1</title>
<script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
<script id="shader-fs" type="x-shader/x-fragment">
precision mediump float;
void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
</script>
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;
uniform mat4 uMVMMatrix;
uniform mat4 uPMMatrix;
void main(void) {
    gl_Position = uPMMatrix * uMVMMatrix * vec4(aVertexPosition, 1.0);
}
</script>
<script type="text/javascript">
var gl;
function initGL(canvas) {
    gl = canvas.getContext("experimental-webgl");
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
}
function getShader(gl, id) {
    var shaderScript = document.getElementById(id);
    var str = "";
    var k = shaderScript.firstChild;
    while (k) {
        if (k.nodeType == 3)
            str += k.textContent;
        k = k.nextSibling;
    }
    var shader;
    if (shaderScript.type == "x-shader/x-vertex")
        shader = gl.createShader(GL_SHADER_VERTEX_SHADER);
    else if (shaderScript.type == "x-shader/x-fragment")
        shader = gl.createShader(GL_SHADER_FRAGMENT_SHADER);
    gl.shaderSource(shader, str);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, GL_COMPILE_STATUS))
        return null;
    return shader;
}
var shaderProgram;
function initShaders() {
    var fragmentShader = getShader(gl, "shader-fs");
    var vertexShader = getShader(gl, "shader-vs");
    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);
    gl.useProgram(shaderProgram);
    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMMatrix");
}
var mvMatrix = mat4.create();
var pMatrix = mat4.create();
mat4.translate(mvMatrix, [1.5, 0, 0]);
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
setMatrixUniforms();
gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
setMatrixUniforms();
gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
}
function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.enable(gl.DEPTH_TEST);
    gl.translate(45, 1.5, 0.1, pMatrix);
    gl.translate(1.5, 0.0, -7.0, mvMatrix);
    gl.bufferData(gl.ARRAY_BUFFER, triangleVertexPositionBuffer, gl.STATIC_DRAW);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
    gl.bufferData(gl.ARRAY_BUFFER, squareVertexPositionBuffer, gl.STATIC_DRAW);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
}
function WebGLStart() {
    var canvas = document.getElementById("lesson01-canvas");
    initGL(canvas);
    initShaders();
    initBuffers();
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
    drawScene();
}
</script></head>
<body onload="WebGLStart()">
<canvas id="lesson01-canvas" style="border: none;" width="400" height="300">
</canvas>
</body> </html>
```

```
var triangleVertexPositionBuffer = new Float32Array([
    0.0, 1.0, 0.0,
    -1.0, -1.0, 0.0,
    1.0, -1.0, 0.0
]);
gl.bufferData(gl.ARRAY_BUFFER,
    new Float32Array(vertices),
    gl.STATIC_DRAW);
triangleVertexPositionBuffer.itemSize = 3;
triangleVertexPositionBuffer.numItems = 3;
squareVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
vertices = [
    1.0, 1.0, 0.0,
    -1.0, 1.0, 0.0,
    1.0, -1.0, 0.0,
    -1.0, -1.0, 0.0
];
gl.bufferData(gl.ARRAY_BUFFER,
    new Float32Array(vertices),
    gl.STATIC_DRAW);
squareVertexPositionBuffer.itemSize = 3;
squareVertexPositionBuffer.numItems = 4;

function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.enable(gl.DEPTH_TEST);
    gl.translate(45, 1.5, 0.1, pMatrix);
    gl.translate(1.5, 0.0, -7.0, mvMatrix);
    gl.bufferData(gl.ARRAY_BUFFER, triangleVertexPositionBuffer,
        gl.STATIC_DRAW);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        triangleVertexPositionBuffer.itemSize,
        gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
    gl.bufferData(gl.ARRAY_BUFFER, squareVertexPositionBuffer,
        gl.STATIC_DRAW);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        squareVertexPositionBuffer.itemSize,
        gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
}

function WebGLStart() {
    var canvas = document.getElementById("lesson01-canvas");
    initGL(canvas);
    initShaders();
    initBuffers();
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
    drawScene();
}
</script></head>
<body onload="WebGLStart()">
<canvas id="lesson01-canvas" style="border: none;" width="400" height="300">
</canvas>
</body> </html>
```



<http://learningwebgl.com/blog/?p=28>

# WebGL開発を簡単にするライブラリ

- 有力なものが複数：
  - three.js, O3D, OSG.JS, ...
- どれもハイレベルなAPIで、OpenGLとはかけ離れている☹
- 手軽に使うには良いが、CGの原理を学ぶのにはあまり適さない (?)

```
<script src="js/three.min.js"></script>
<script>
var camera, scene, renderer, geometry, material, mesh;
function init() {
  scene = new THREE.Scene();
  camera = new THREE.PerspectiveCamera( 75, 640 / 480, 1, 10000 );
  camera.position.z = 1000;
  geometry = new THREE.BoxGeometry( 200, 200, 200 );
  material = new THREE.MeshBasicMaterial({color:0xff0000, wireframe:true});
  mesh = new THREE.Mesh( geometry, material );
  scene.add( mesh );
  renderer = new THREE.WebGLRenderer();
  renderer.setSize(640, 480);
  document.body.appendChild( renderer.domElement );
}
function animate() {
  requestAnimationFrame( animate );
  render();
}
function render() {
  mesh.rotation.x += 0.01;
  mesh.rotation.y += 0.02;
  renderer.render( scene, camera );
}
init();
animate();
</script>
```

three.js

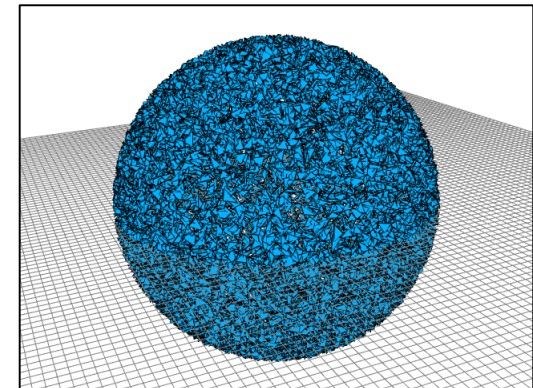
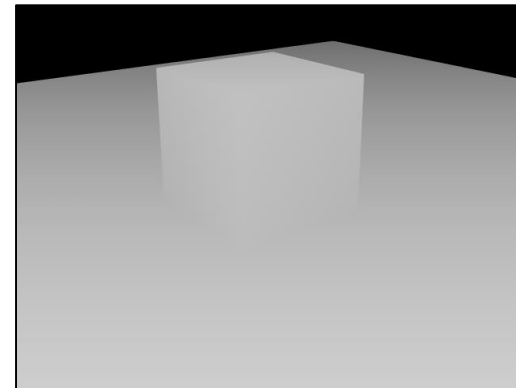
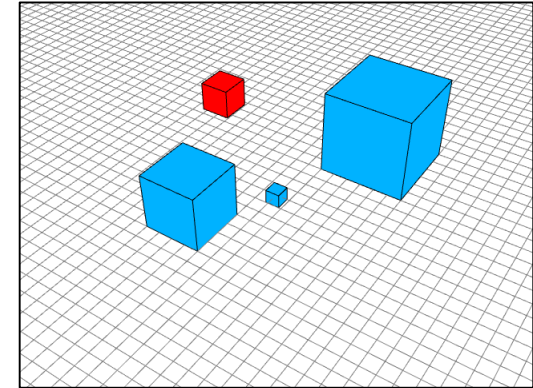
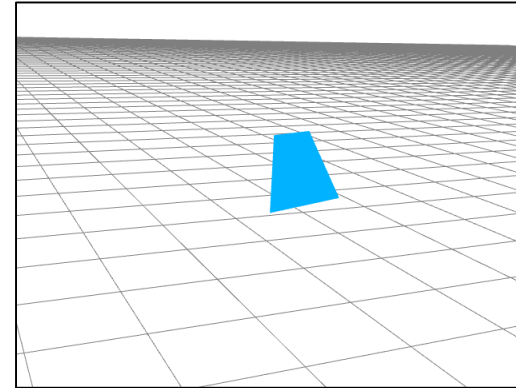
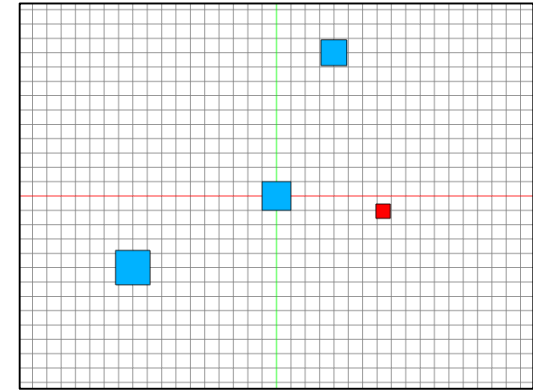
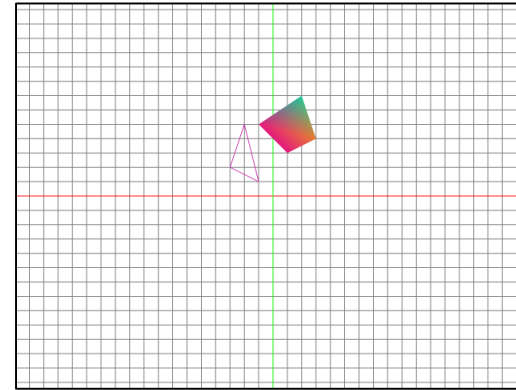
ハイレベルなAPI





# legacygl.js

- 本講義用に高山が開発
  - <https://bitbucket.org/kenshi84/legacygl.js>
  - デモとチュートリアル (英語)
- 課題のサンプルコードはこれを使う
  - 各自動かしてみて、仕組みを大まかに把握しておくこと





# Mozilla ThimbleによるWebGL開発

<https://thimble.mozilla.org/>

- js/html/cssを置く空間を無料で提供
- 手軽に編集してプレビューできる

utokyo-iscg-2016

HelpEnglish (US)Hi, kenshi84Publish

FILES

EDITOR - assignm...line.html

PREVIEW

▼ assignments

▶ data

assignment-1-spline.html

assignment-2-kinematics

assignment-3-filter.html

assignment-a-subdivisor

assignment-c-pbd.html

assignment-d-fluid.html

assignment-e-poisson.ht

assignment-f-patchmatch

▼ demo

▶ data

displist.html

hello2d.html

hello3d.html

meshviewer.html

pick2d.html

pick3d.html

z-buffer.html

boundingbox.js

1 ▼ <html>

2 ▼ <head>

3 <title id="title">Assignment 1: Spline</title>

4 <script src="../../gl-matrix.js"></script>

5 <script src="../../gl-matrix-util.js"></script>

6 <script src="../../legacygl.js"></script>

7 <script src="../../drawutil.js"></script>

8 <script src="../../camera.js"></script>

9 <script src="../../util.js"></script>

10 <script src="../../glu.js"></script>

11 <script src="../../numeric-1.2.6.js"></script>

12 <script type="text/javascript">

13 var gl;

14 var canvas;

15 var legacygl;

16 var drawutil;

17 var camera;

18 var p0, p1, p2;

19 var selected = null;

20

21 ▼ function eval\_quadratic\_bezier(p0, p1, p2, t) {

22 return numeric.add(numeric.mul(1 - t, p0),

numeric.mul(t, p2)); // TODO

23 }

24

25 ▼ function draw() {

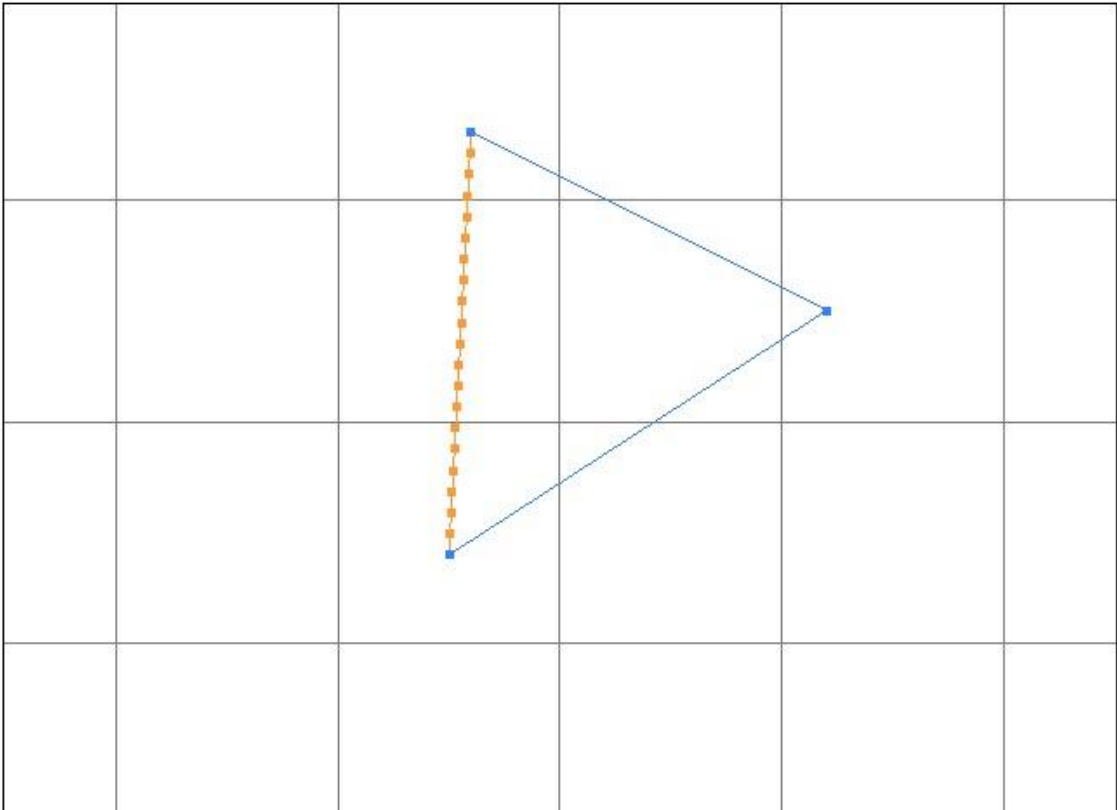
26 gl.clear(gl.COLOR\_BUFFER\_BIT |

gl.DEPTH\_BUFFER\_BIT);

// projection & camera position

27

Assignment 1: Spline

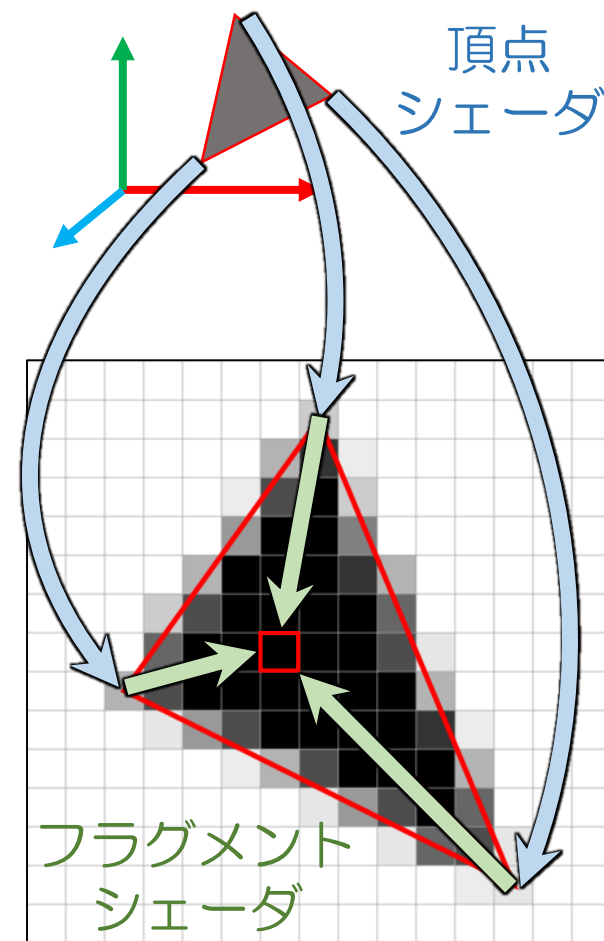


# 課題提出の方法

- WebGLで実装してWeb上にアップロードし、そのURLをTAにメール
  - Thimbleを使うのが簡単だが、自前のサーバ上でも良い
  - HTMLページ内に、実装内容の説明・考察・感想等を含めること
  - three.js等他のWebGLライブラリで実装しても良い
- C++等で実装しても良い
  - ただし、一般的な環境でビルド・実行できること
  - ソース・バイナリ・説明文を一つのZIPファイルにまとめて提出
- 分からない場合はTAまたは私まで相談すること

# シェーダについて

- 頂点シェーダ：頂点ごとの処理
  - 様々なデータをglBufferDataによって渡す
    - 座標値、色、テクスチャ座標、...
  - 必須の処理：座標変換後のピクセル位置の指定 (gl\_Position)
- フラグメントシェーダ：ピクセルの塗りつぶし処理
  - 頂点のデータを線形補間
  - 必須の処理：描画するピクセルの色の指定 (gl\_FragColor)
- GLSL (Open**GL** **S**hading **L**anguage) ソースを文字列としてGPUに渡し、**実行時にコンパイル**



# シェーダ変数

- uniform変数

- 頂点シェーダ・フラグメントシェーダで読み取り可
- 頂点配列とは別にGPUに渡す (glUniform)
- 例：座標変換行列、条件付き処理のフラグ

- attribute変数

- 頂点シェーダで読み取りのみ可
- 頂点配列としてGPUに渡す (glBufferData)
- 例：位置XYZ、色RGB、テクスチャUV

- varying変数

- 頂点シェーダで書き込み、フラグメントシェーダで読み取る
- 頂点での値を各ピクセルで線形補間

```
uniform mat4 u_modelview;
uniform mat4 u_projection;
attribute vec3 a_vertex;
attribute vec3 a_color;
varying vec3 v_color;
void main(void) {
    gl_Position = u_projection
                  * u_modelview
                  * vec4(a_vertex, 1.0);
    v_color = a_color;
}
```

頂点シェーダ

```
precision mediump float;
varying vec3 v_color;
void main(void) {
    gl_FragColor.rgb = v_color;
    gl_FragColor.a   = 1.0;
}
```

フラグメントシェーダ

(バージョンによって文法が多少異なる)

# JavaScript初心者 (=高山) のためのヒント

- 型：文字列 / ブール / 数値 / 関数 / オブジェクト / null / undefined
  - C++的な型システムではない
- 数値：すべて倍精度 (整数と実数を区別しない)
- オブジェクト：文字列をキーとした連想配列
  - `x.abc` は `x["abc"]` と等価 (「メンバ」的な見かけ)
  - `{ abc : y }` は `{ "abc" : y }` と等価
  - 文字列以外のキーは暗黙に文字列に変換される
- 配列はキーが連続した整数であるオブジェクト
  - ただし特別な機能を持つ： `.length` , `.push()` , `.pop()` , `.forEach()`
- 代入や引数はすべて値渡し
  - 「ディープコピー」のための文法は無い
- 迷ったらすぐ `console.log(x)`

# 参考

- OpenGL
  - 床井研究室  
<http://marina.sys.wakayama-u.ac.jp/~tokoi/oglarticles.html>
  - OpenGL入門  
<http://wisdom.sakura.ne.jp/system/opengl/>
  - 公式リファレンス  
<https://www.opengl.org/sdk/docs/man/html/indexflat.php>
- WebGL/JavaScript/HTML5
  - Learning WebGL  
<http://learningwebgl.com/blog/?p=11>
  - 公式リファレンス  
<https://www.khronos.org/registry/webgl/specs/1.0/#5.14>
  - Mozilla Developer Network
    - <https://developer.mozilla.org>
  - An Introduction to JavaScript for Sophisticated Programmers  
<http://casual-effects.blogspot.jp/2014/01/>
  - Effective JavaScript  
<http://effectivejs.com/>

# 参考

- [http://en.wikipedia.org/wiki/Affine\\_transformation](http://en.wikipedia.org/wiki/Affine_transformation)
- [http://en.wikipedia.org/wiki/Homogeneous\\_coordinates](http://en.wikipedia.org/wiki/Homogeneous_coordinates)
- [http://en.wikipedia.org/wiki/Perspective \(graphical\)](http://en.wikipedia.org/wiki/Perspective_(graphical))
- <http://en.wikipedia.org/wiki/Z-buffering>
- <http://en.wikipedia.org/wiki/Quaternion>