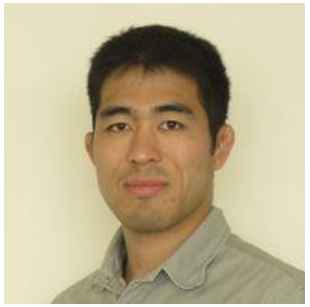
コンピュータグラフィクス論

2015年4月9日

高山 健志

教員紹介

- 高山 健志 (国立情報学研究所 特任助教)
  - http://research.nii.ac.jp/~takayama/
  - takayama@nii.ac.jp

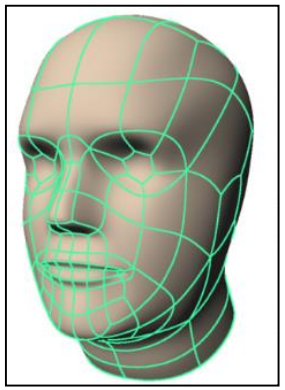


- 蜂須賀 恵也 (創造情報学専攻 講師)
  - http://www.ci.i.u-tokyo.ac.jp/~hachisuka/
  - thachisuka@siggraph.org

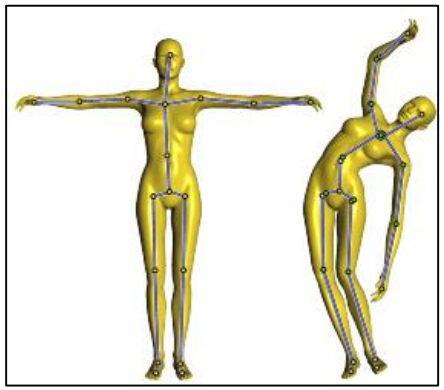
- TA：中島 一崇 (五十嵐研)
  - http://n-taka.info/intro/
  - taka@ui.is.s.u-tokyo.ac.jp

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
- レンダリングの回は蜂須賀先生が担当
- 必要に応じて演習の回を追加？

成績評価の方法

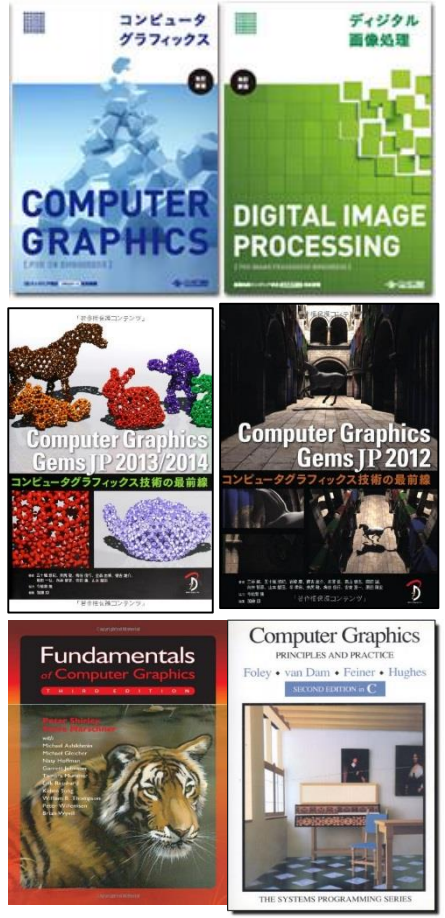
- プログラミング課題のみ、試験はしない

- 各トピックにつき、必須課題1つ＋オプション課題2つ
  - 必須課題1~4、オプション課題A~H
  - 締切は出題から2週間後

- サンプルコード (WebGL, C++?) を提供
  - 他の言語/フレームワーク等でも可

講義情報

- 講義ページ
  - http://research.nii.ac.jp/~takayama/teaching/utokyo-iscg-2015/



- 参考書
  - コンピュータグラフィクス 改訂新版 (9784903474496)
  - ディジタル画像処理 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)

蜂須賀先生より補足

研究紹介

座標変換

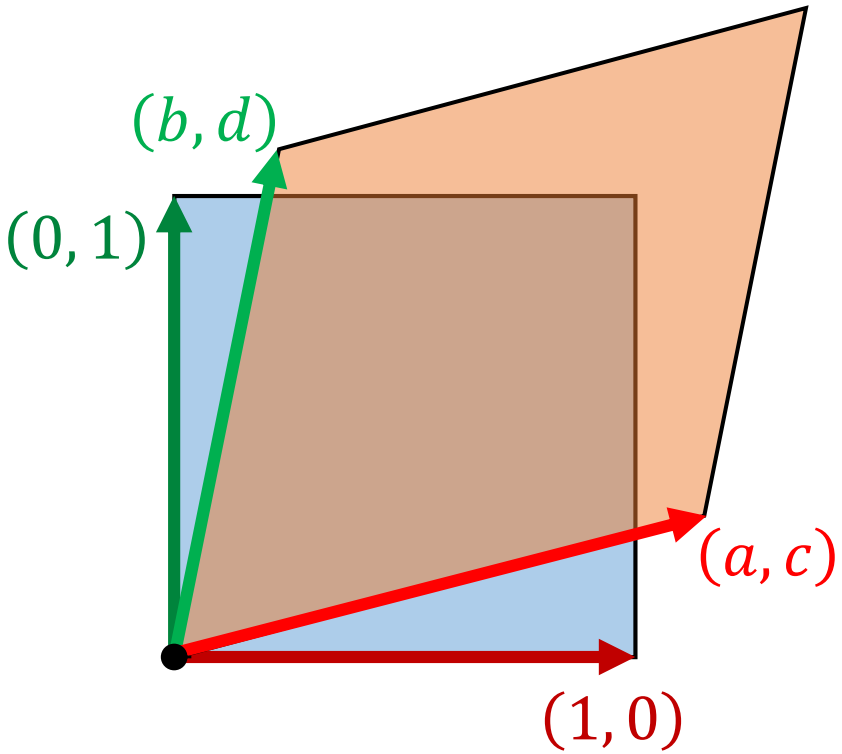
線形変換

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

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

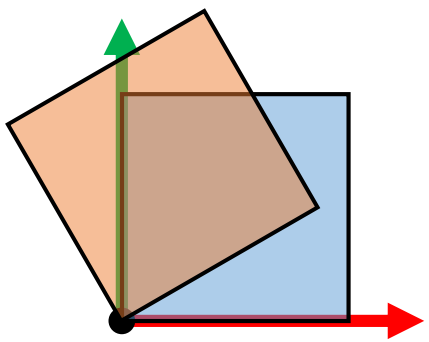
$$\begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



- 原点は動かない

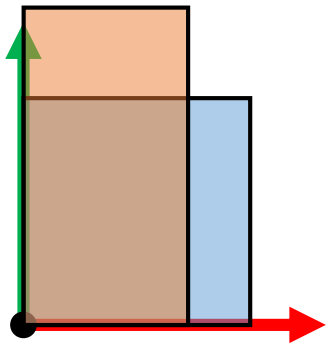
いろいろな線形変換

回転



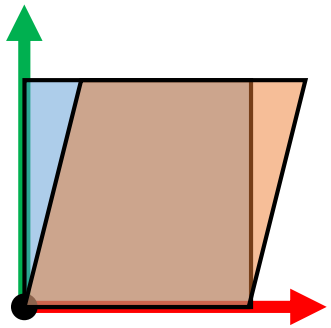
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

スケーリング



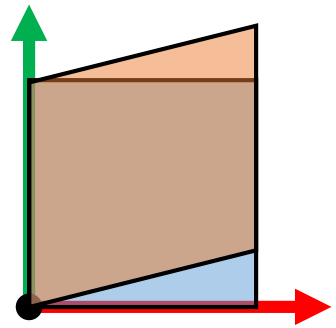
$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

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

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

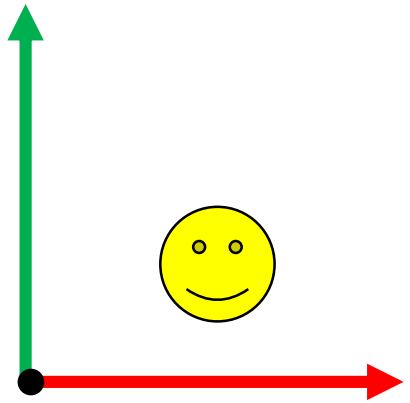
- 同次座標：2D (3D) 座標を表すのに、便宜的に3D (4D) ベクトルを使う
- 線形変換と平行移動を、行列の積として同じように表せる！
  - 実装上都合が良い

アフィン変換の合成

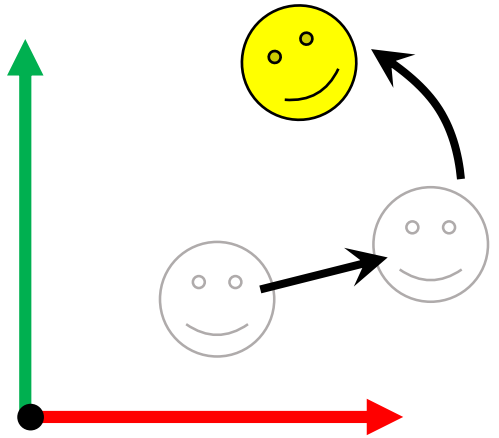
- 変換行列を掛けるだけ
- 掛ける順番に注意！

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

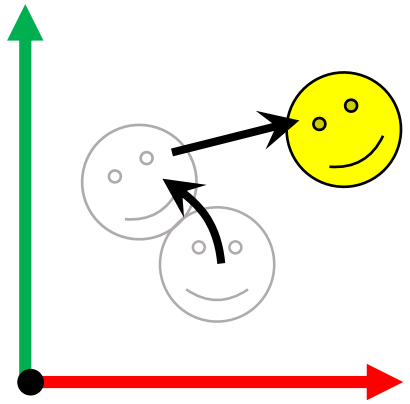
$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$



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



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



同次座標

- w≠0のとき、4D同次座標(x, y, z, w)は3D空間座標$\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$を表す

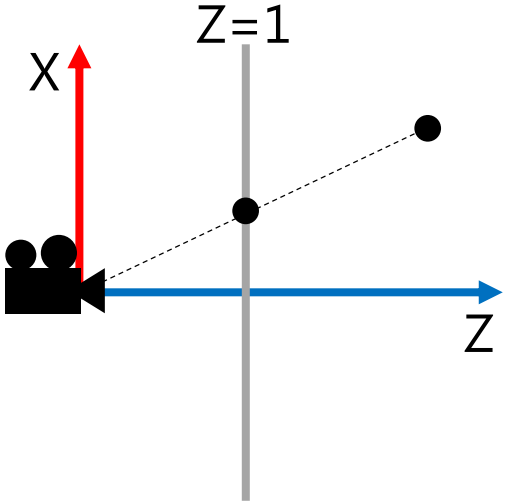
- 普通の3D空間 (ユークリッド空間) に、無限遠点を追加した空間 (射影空間) を扱える
  - w→0のとき表される3D座標は無限に遠ざかる
    ➔ $(x, y, z, 0)$で3D空間の(x, y, z)方向の無限遠点 (方向ベクトル) を表す
  - 位置ベクトル同士の差が方向ベクトルになる：
    $$(x, y, z, 1) - (x', y', z', 1) = (x - x', y - y', z - z', 0)$$
  - 同次座標$(0, 0, 0, 0)$は定義されない

- 背景に少し難解な理論 (cf. Wikipedia)

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

- 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)は$(w_x, w_y) = \left(\frac{p_x}{p_z}, \frac{p_x}{p_z}\right)$に投影される



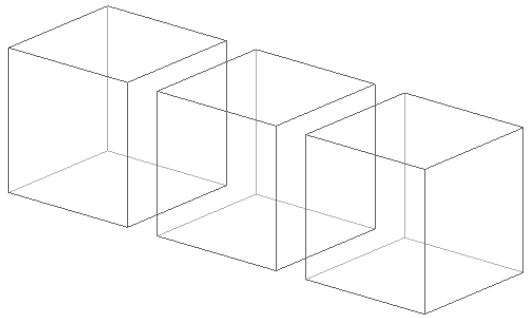
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z + 1 \\ p_z \end{bmatrix} \equiv \begin{bmatrix} p_x/p_z \\ p_y/p_z \\ 1 + 1/p_z \\ 1 \end{bmatrix} \begin{matrix} \rightarrow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

射影変換

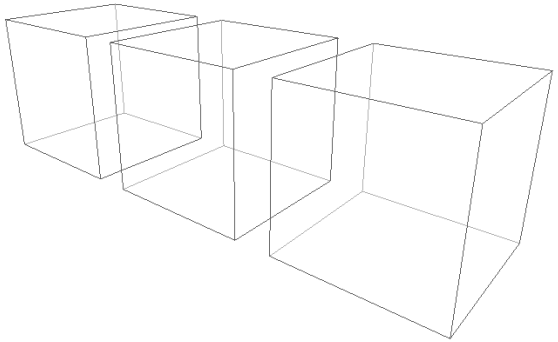
- w_z (深度値) は、前後関係の判定に使われる➔Zバッファ法

平行投影

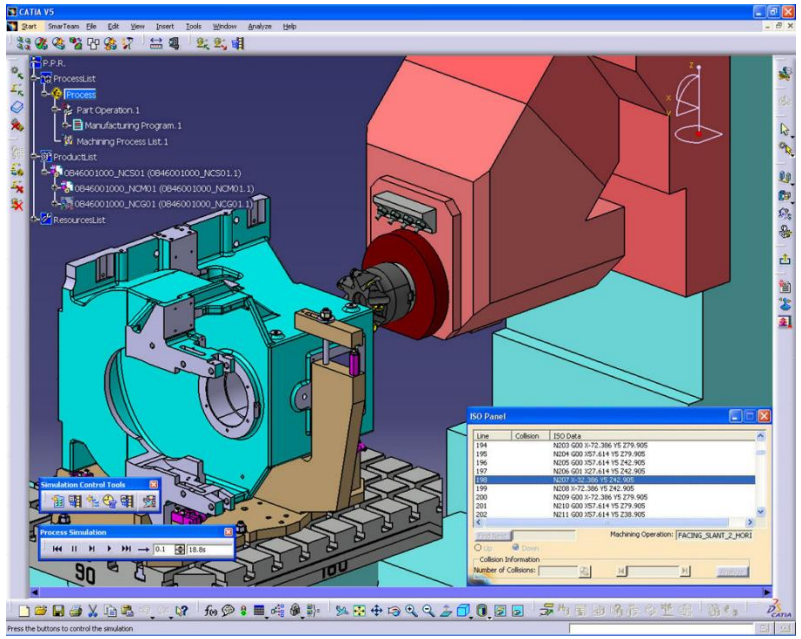
- 物体の見かけ上の大きさが、
  視点からの距離に影響されない
- 単にZ座標を無視するだけ
- 製図でよく使われる



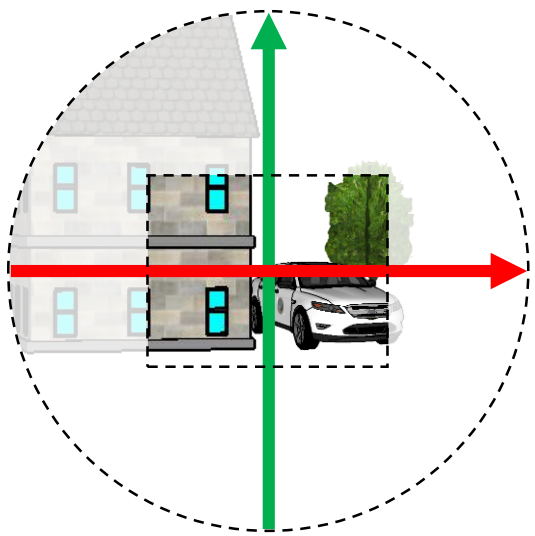
平行投影



透視投影

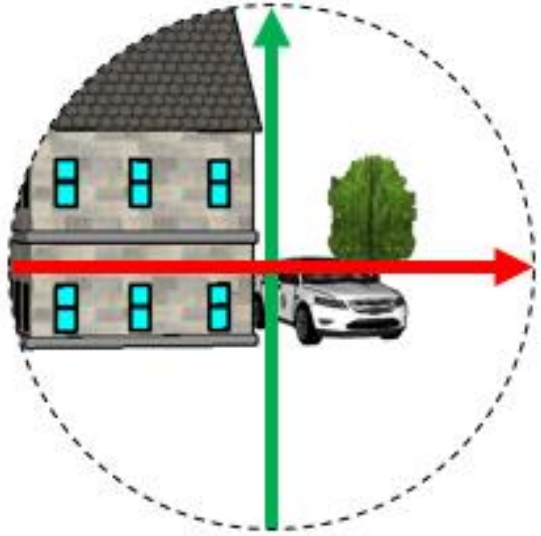


ビューイングパイプライン



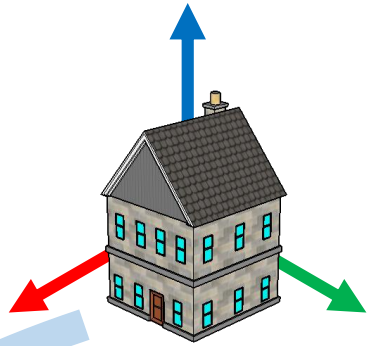
投影座標系

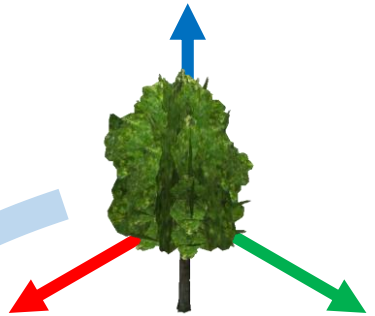
射影変換
(4×4行列)

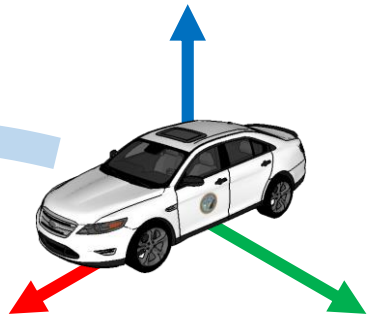


カメラ座標系

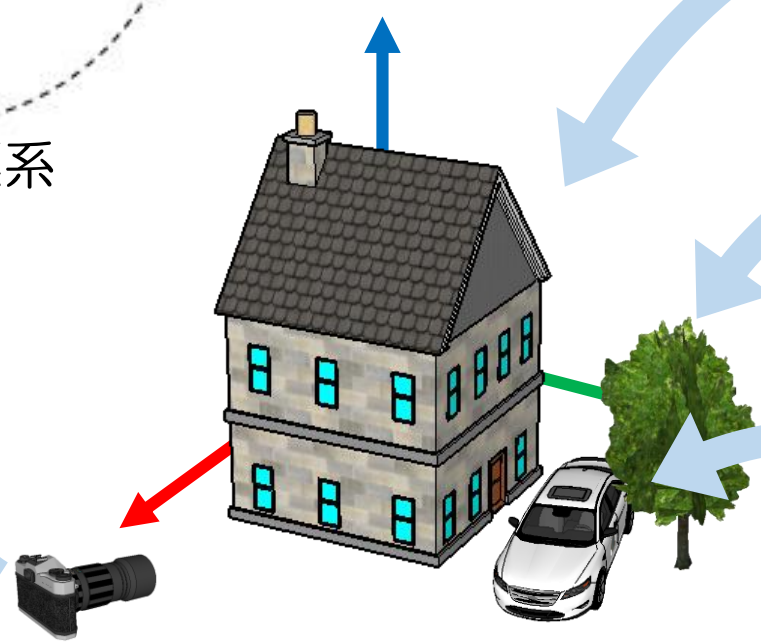
モデルビュー変換
(4×4行列)





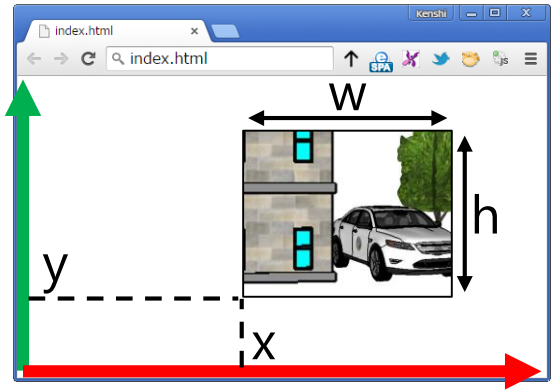


ローカル座標系



オブジェクト座標系
(ワールド座標系)

ビューポート変換
(x, y, w, h)



ウインドウ座標系

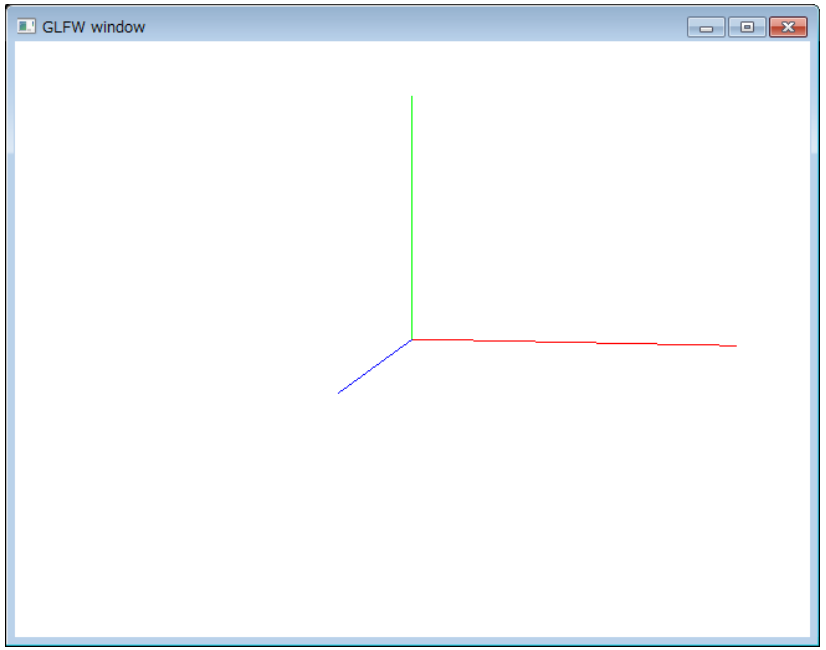
典型的なOpenGLコード

```
glViewport(0, 0, 640, 480);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(
  45.0,          // field of view
  640 / 480,     // aspect ratio
  0.1, 100.0); // depth range
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
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();
```

ビューポート変換

射影変換

モデルビュー変換

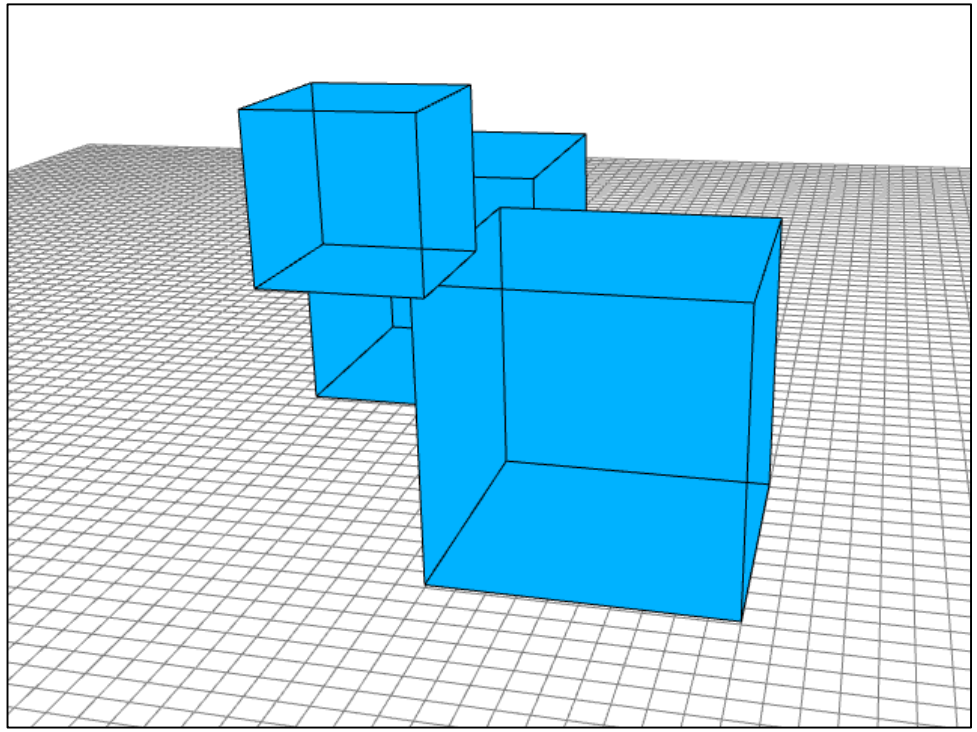


描画結果

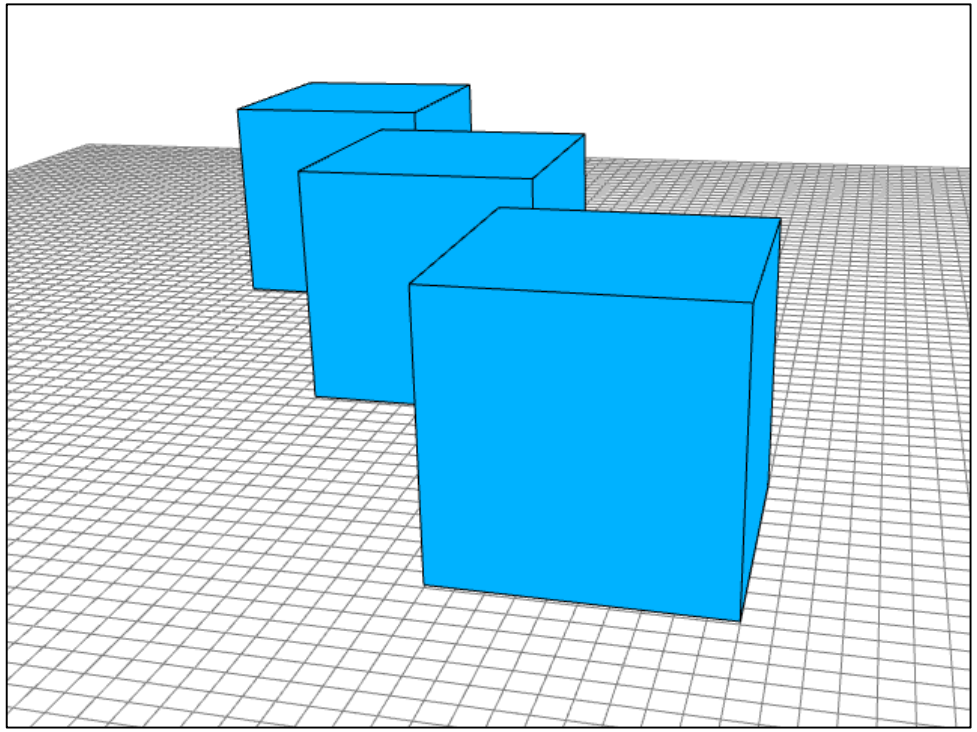
シーン内容

Zバッファ法

隠面消去



隠面消去なし

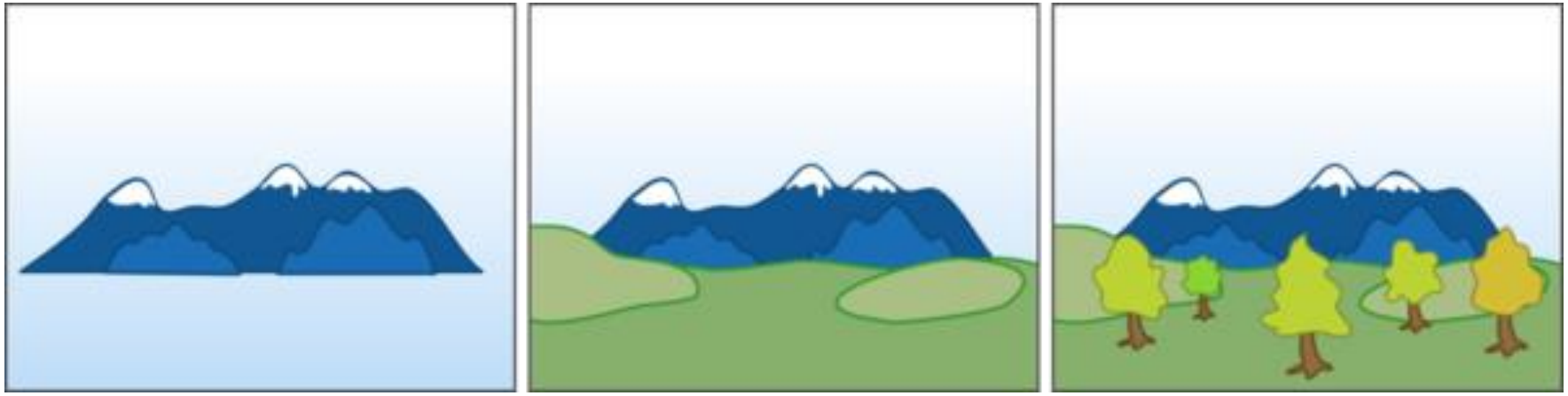


隠面消去あり

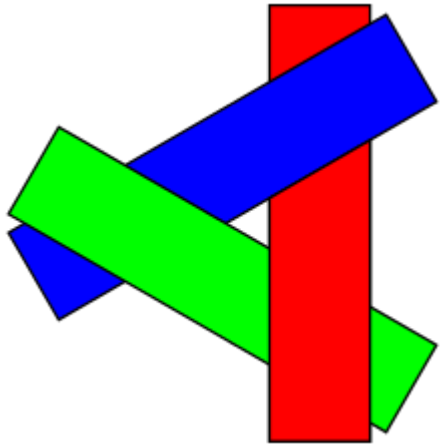
- CGの古典的な問題

画家のアルゴリズム

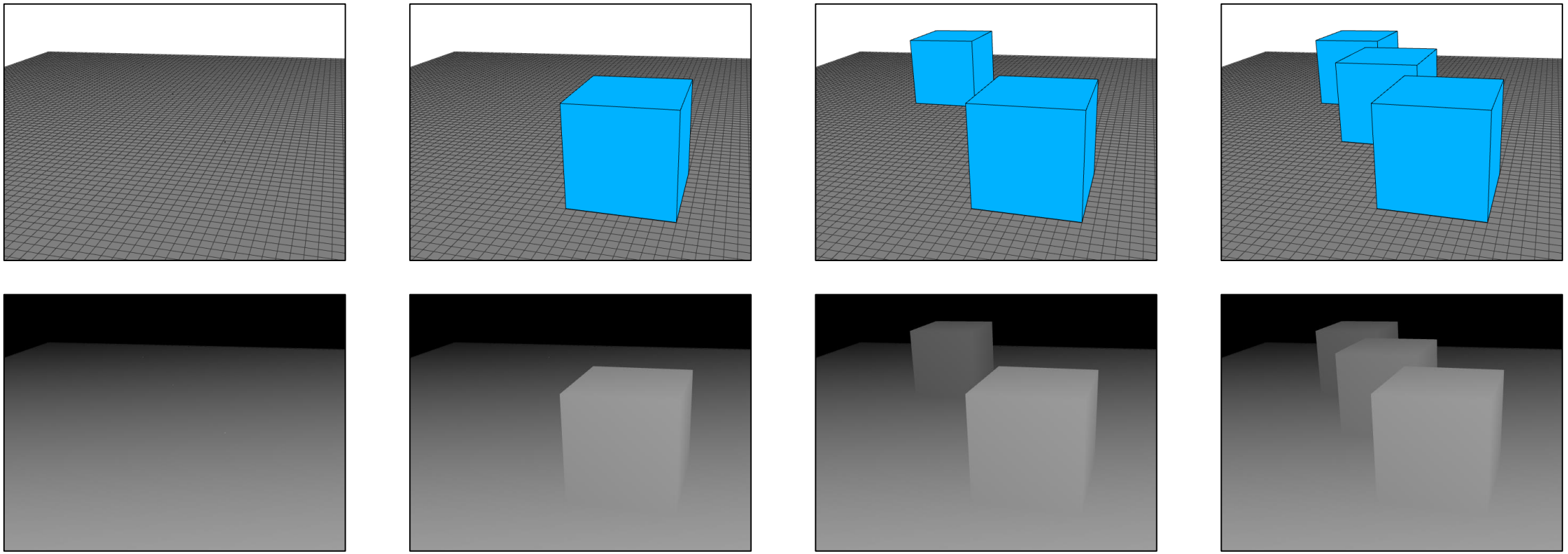
- 物体を視点からの距離でソートし、遠くものから描画



- 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



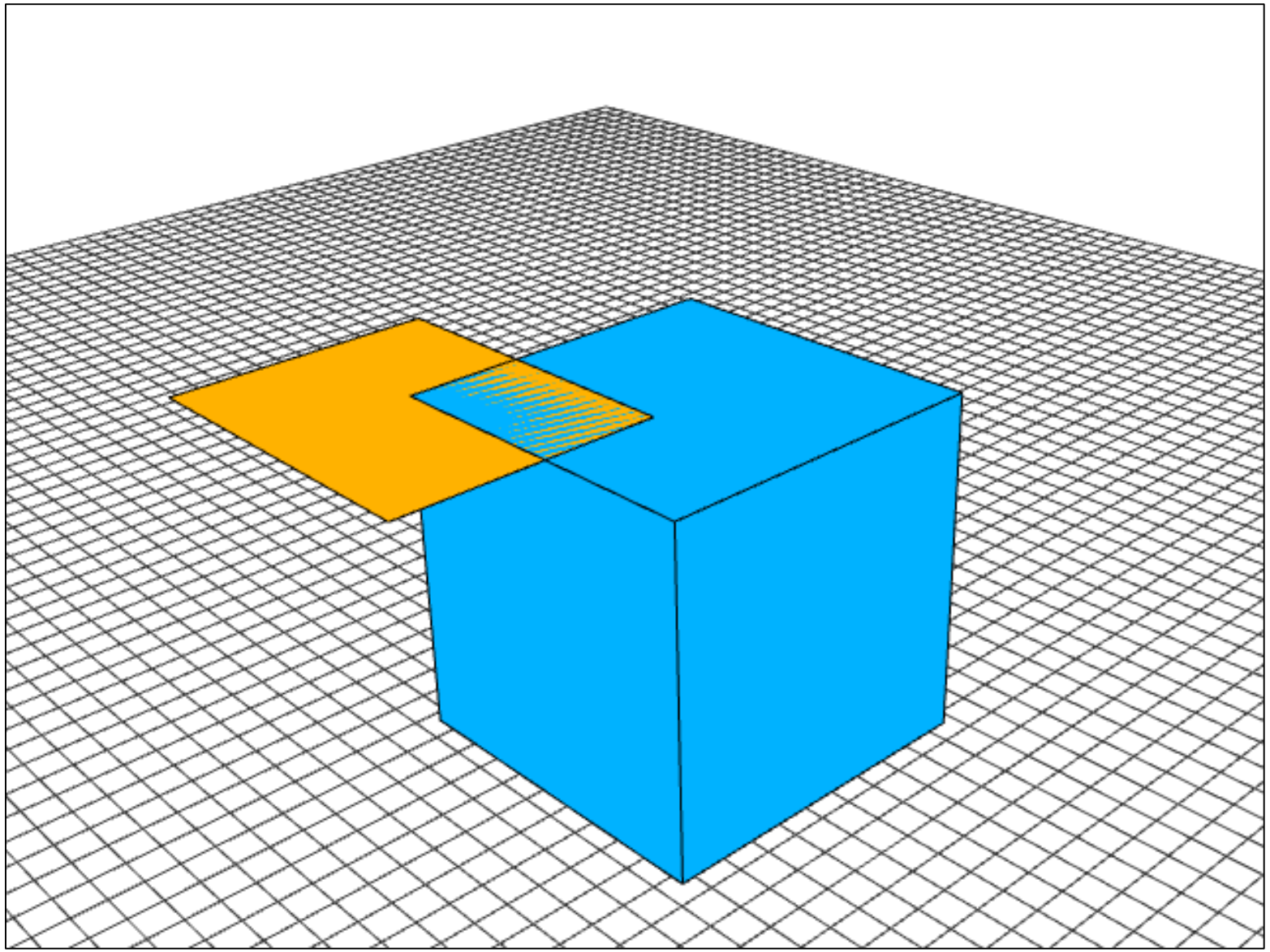
Zバッファ法



- 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が、現在のスタンダード

Zバッファの注意点：Z-fighting

- 前後の判定が
  そもそも不可能



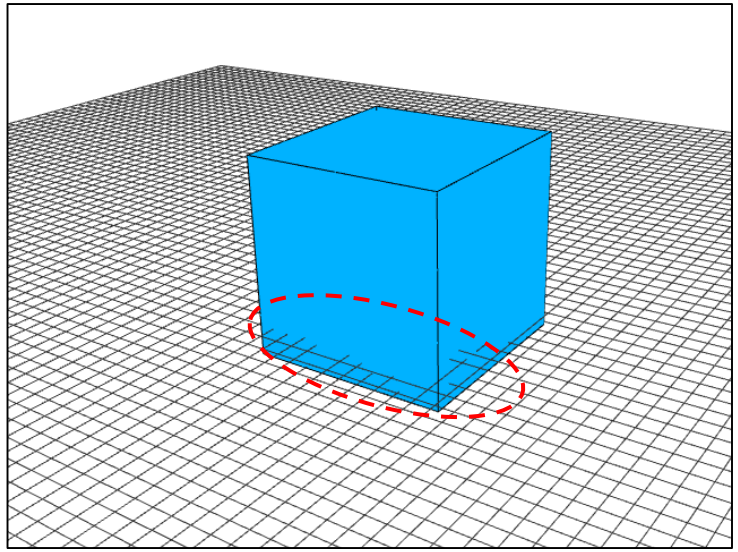
Zバッファの注意点：深度値の範囲

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

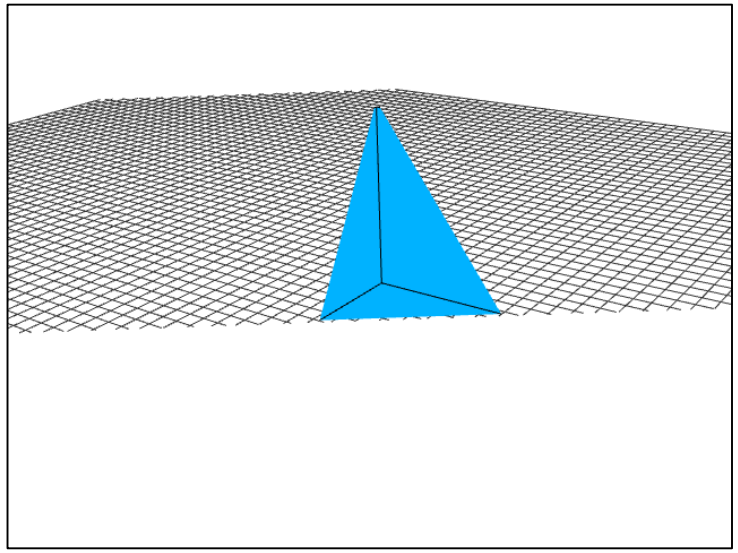
- Zバッファのビット数は固定
  - 16~24bit程度

- 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)



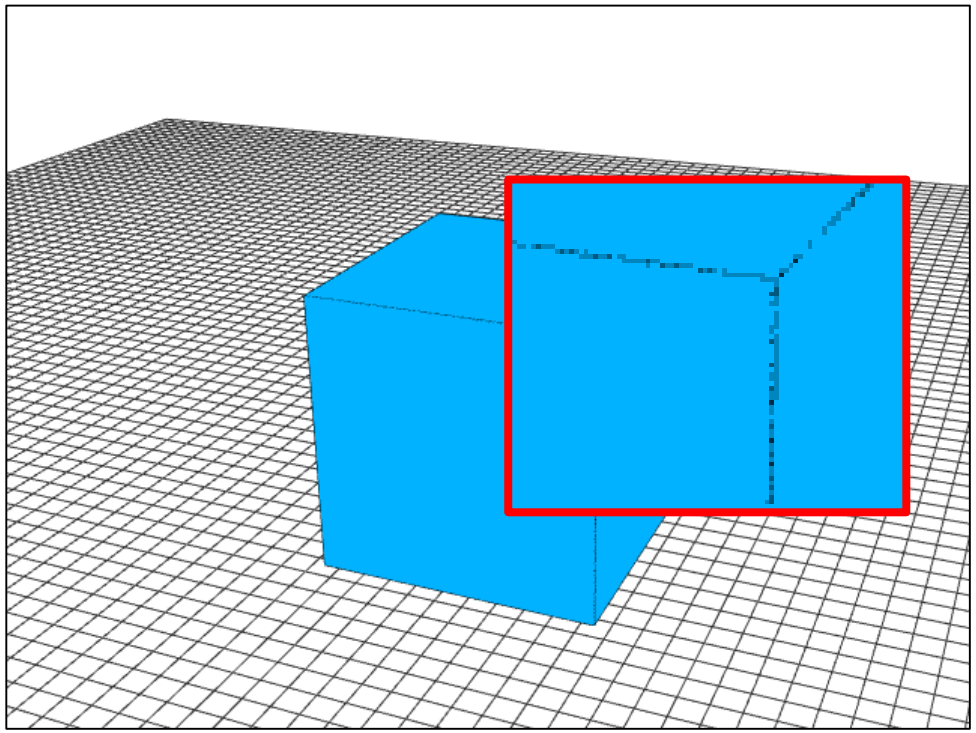
zNear=0.0001
zFar　=1000



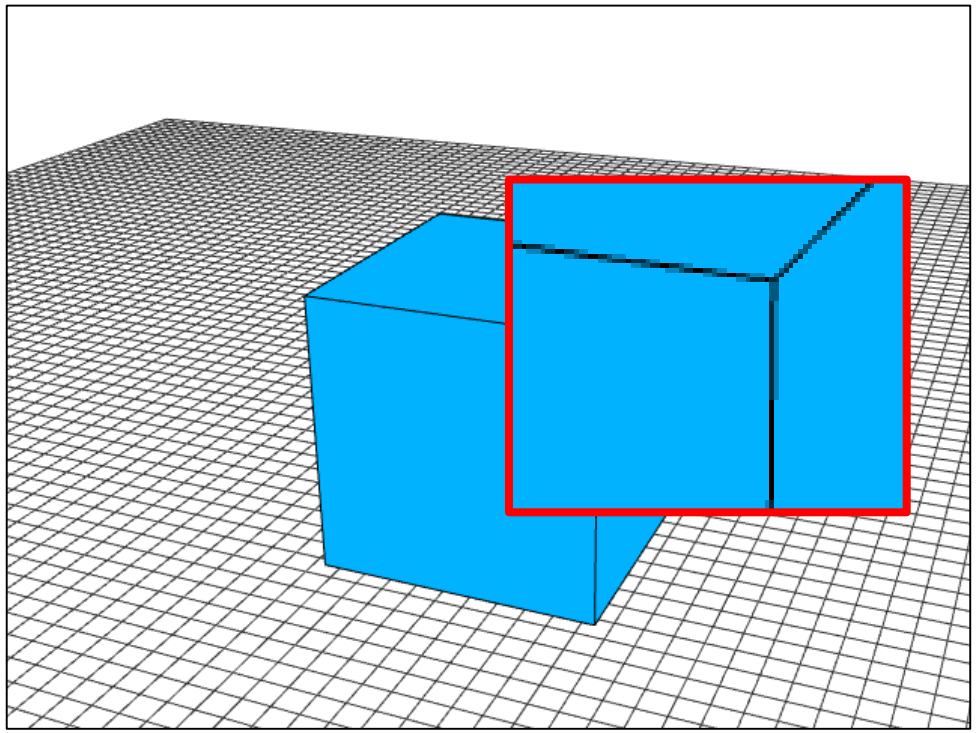
zNear=50
zFar　=100

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

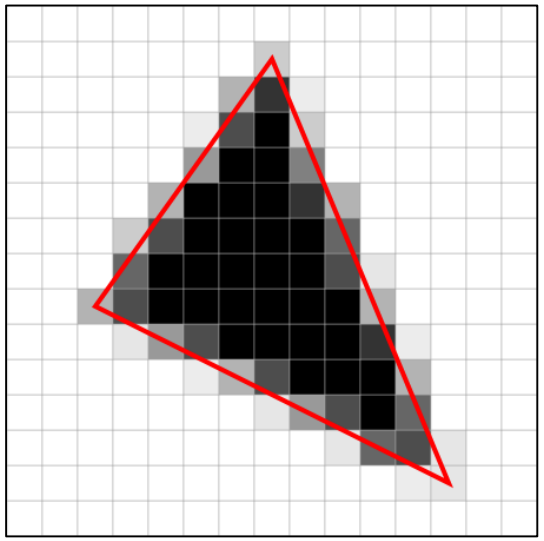
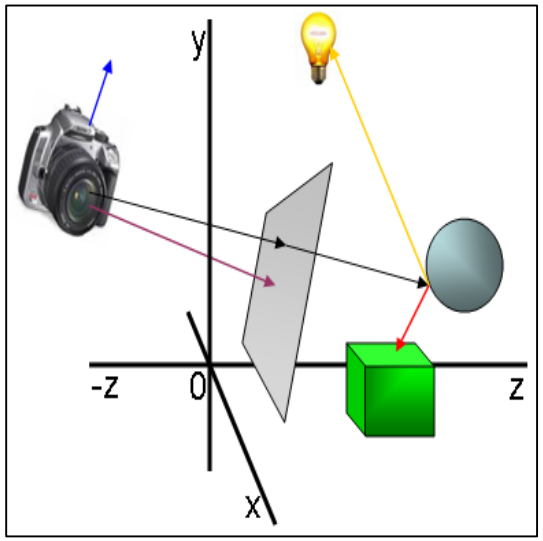


polygon offset なし



polygon offset あり

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

クオータニオン

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

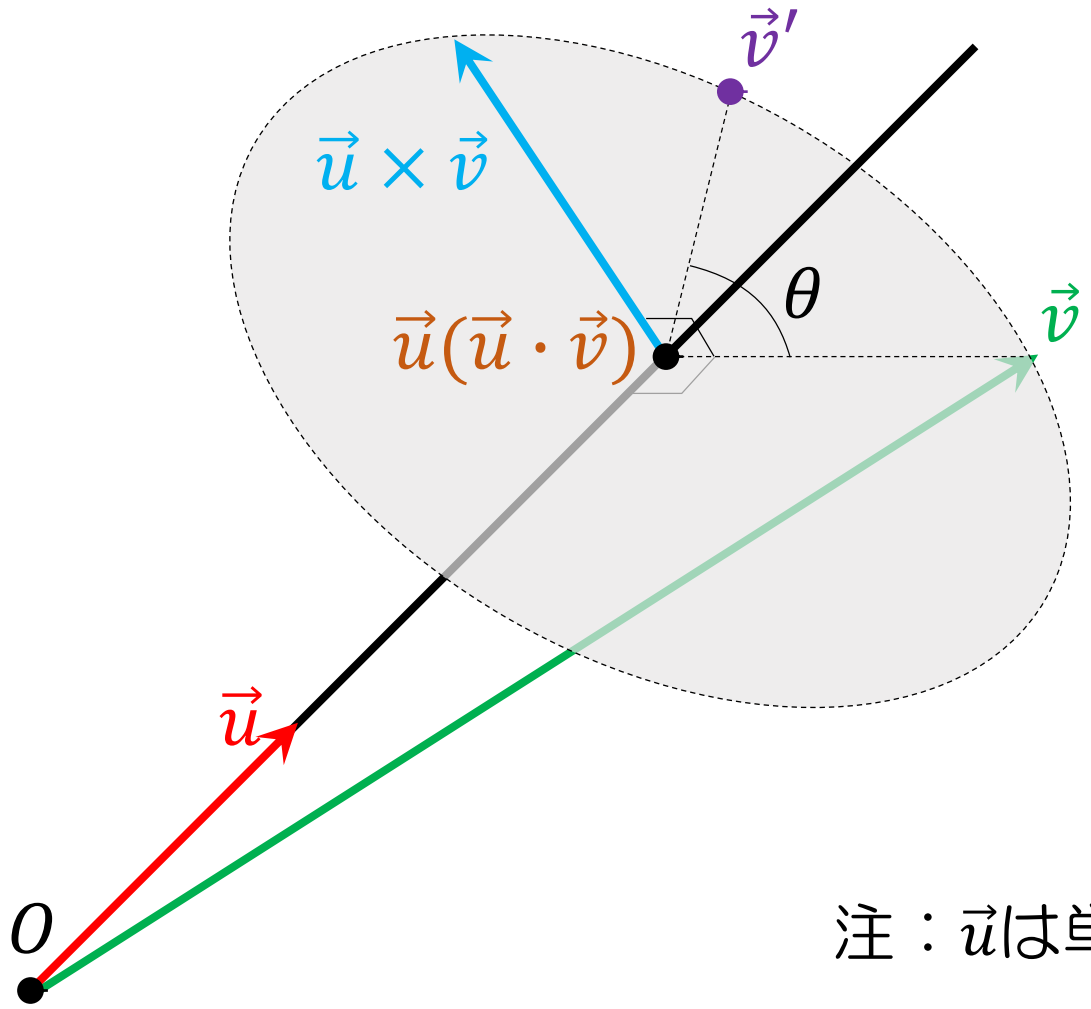
Y軸周り
$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

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

- 行列表現は無駄に複雑！
  - 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

任意軸周り回転の幾何



注：$\vec{u}$は単位ベクトル

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

クオータニオン (四元数)

- 複素数
  - $\mathbf{i}^2 = -1$
  - $\mathbf{c} = (a, b) \coloneqq a + b\,\mathbf{i}$
  - $\mathbf{c}_1\mathbf{c}_2 = (a_1, b_1)(a_2, b_2) = a_1a_2 - b_1b_2 + (a_1b_2 + b_1a_2)\,\mathbf{i}$

- クオータニオン
  - $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$
    - $\mathbf{ij} = -\mathbf{ji} = \mathbf{k}, \quad \mathbf{jk} = -\mathbf{kj} = \mathbf{i}, \quad \mathbf{ki} = -\mathbf{ik} = \mathbf{j}$　　可換ではない！
  - $\mathbf{q} = (a, b, c, d) \coloneqq a + b\,\mathbf{i} + c\,\mathbf{j} + d\,\mathbf{k}$
  - $\mathbf{q}_1\mathbf{q}_2 = (a_1, b_1, c_1, d_1)(a_2, b_2, c_2, d_2)$

$$= (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) + (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\,\mathbf{i}$$

$$+ (a_1c_2 + c_1a_2 + d_1b_2 - b_1d_2)\,\mathbf{j} + (a_1d_2 + d_1a_2 + b_1c_2 - c_1b_2)\,\mathbf{k}$$

スカラー＋3Dベクトルによる表記

- $\mathbf{q} = a + b\,\mathbf{i} + c\,\mathbf{j} + d\,\mathbf{k} \coloneqq a + (b, c, d) = a + \vec{v}$
- $\mathbf{q_1 q_2} = (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2)\,\mathbf{i}$

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

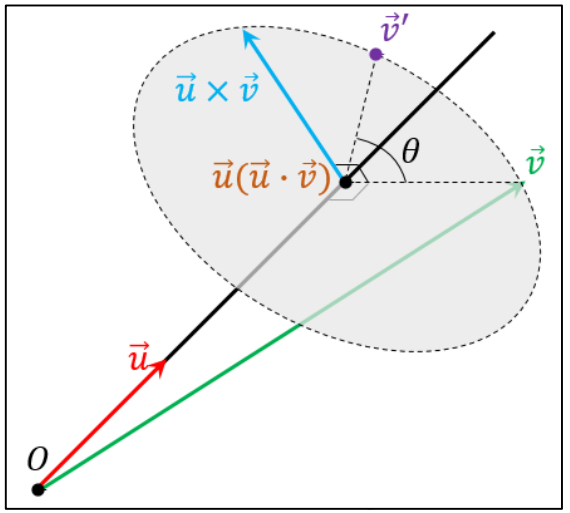
  $= (a_1 a_2 - \overrightarrow{v_1} \cdot \overrightarrow{v_2}) + a_1 \overrightarrow{v_2} + a_2 \overrightarrow{v_1} + \overrightarrow{v_1} \times \overrightarrow{v_2}$

クオ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

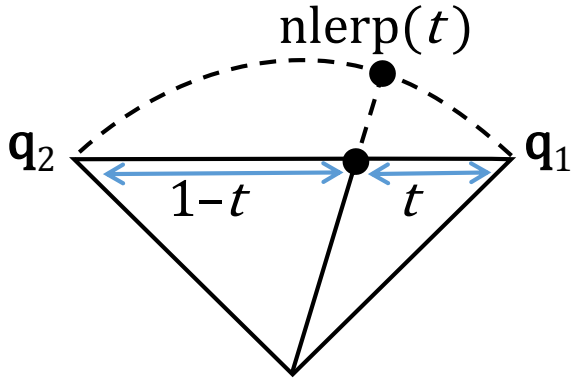
$$\begin{aligned}
\vec{v'} = q\vec{v}q^{-1} &= \left(\cos\frac{\alpha}{2} + \vec{u}\sin\frac{\alpha}{2}\right)\vec{v}\left(\cos\frac{\alpha}{2} - \vec{u}\sin\frac{\alpha}{2}\right) \\
&= \vec{v}\cos^2\frac{\alpha}{2} + (\vec{u}\vec{v} - \vec{v}\vec{u})\sin\frac{\alpha}{2}\cos\frac{\alpha}{2} - \vec{u}\vec{v}\vec{u}\sin^2\frac{\alpha}{2}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



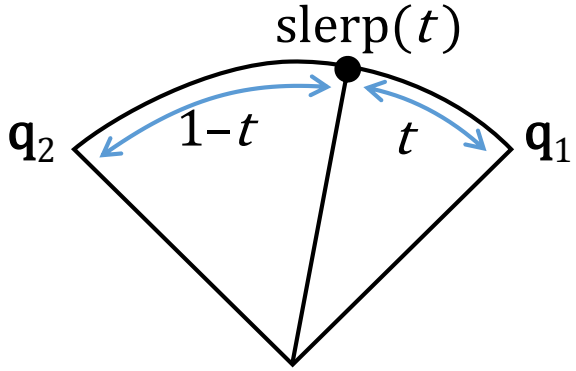
- 背景には面白い理論 (cf. Wikipedia)

クオータニオンによる回転の補間

- 線形補間＋正規化 (nlerp)
  - $\mathrm{nlerp}(\mathbf{q}_1, \mathbf{q}_2, t) \coloneqq \mathrm{normalize}\big((1-t)\mathbf{q}_1 + t\,\mathbf{q}_2\big)$
  - ☺計算が少ない、☹角速度が一定でない



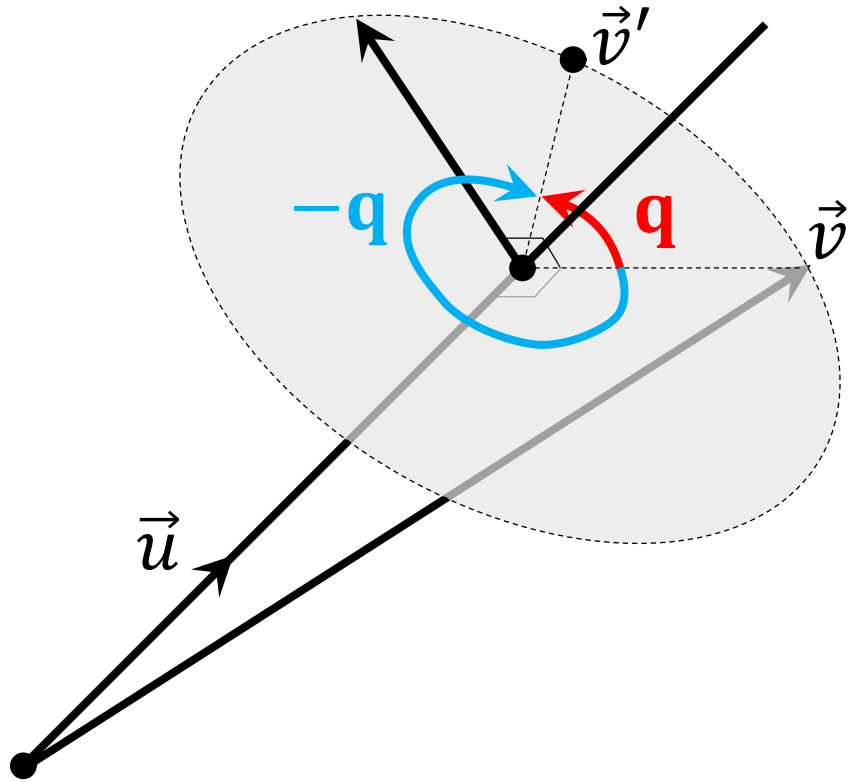
- 球面線形補間 (slerp)
  - $\Omega = \cos^{-1}(\mathbf{q}_1 \cdot \mathbf{q}_2)$
  - $\mathrm{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) \coloneqq \frac{\sin(1-t)\Omega}{\sin\Omega}\mathbf{q}_1 + \frac{\sin t\Omega}{\sin\Omega}\mathbf{q}_2$
  - ☹計算が多い、☺角速度が一定



Animating rotation with quaternion curves. Shoemake, SIGGRAPH 1985
http://number-none.com/product/Understanding%20Slerp,%20Then%20Not%20Using%20It/

正負のクオータニオン

- 回転角がθのクオータニオン：
  - $\mathbf{q} = \cos\frac{\theta}{2} + \vec{u}\sin\frac{\theta}{2}$
- 回転角が$\theta - 2\pi$のクオータニオン：
  - $\cos\frac{\theta-2\pi}{2} + \vec{u}\sin\frac{\theta-2\pi}{2} = -\mathbf{q}$



- $\mathbf{q}_1$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

WebGL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

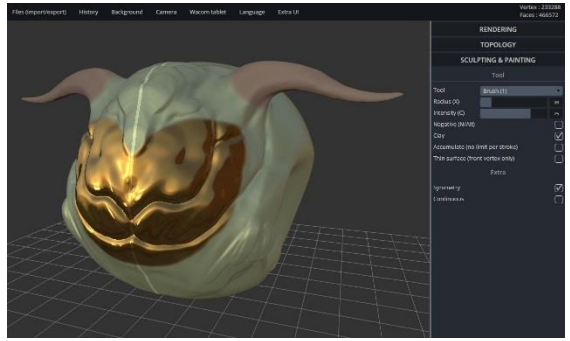
Microsoft DirectX

- 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

- 開発・実行環境の準備が若干面倒

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
- 実行速度に多少の不安？
- 最近注目が高まっている



WebGL開発のハードル：OpenGL ES (for **E**mbedded **S**ystems)

- OpenGL 1.xのAPIが使えない！
  - 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	glBegin, glVertex, glColor, glTexCoord
多角形の描画	GL_QUADS, GL_POLYGON
光と材質	glLight, glMaterial
座標変換行列	GL_MODELVIEW, GL_PROJECTION
ディスプレイリスト	glNewList
デフォルトのシェーダ	

- 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

C / OpenGL 1.x

```c
#include <GL/glut.h>
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

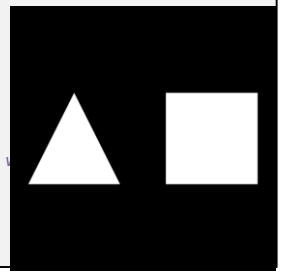
```
<html><head>
<title>Learning WebGL &mdash; lesson 1</title>
<script type="text/javascript" src="glMatrix-0.9.5.mi
<script id="shader-fs" type="x-shader/x-fragment">
precision mediump float;
void main(void) {
  gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
</script>
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;
uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
void main(void) {
  gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
}
</script>
<script type="text/javascript">
var gl;
function initGL(canvas) {
  gl = canvas.getContext("experimental-webgl");
  gl.viewportWidth = can
  gl.viewportHeight = ca
}
function getShader(gl, i
  var shaderScript = doc
var str = "";
  var k = shaderScript.f
  while (k) {
    if (k.nodeType == 3)
      str += k.textConte
    }
    k = k.nextSibling;
  }
  var shader;
  if (shaderScript.type
    shader = gl.createSh
  } else if (shaderScript
    shader = gl.createSh
  }
  gl.shaderSource(shader
  gl.compileShader(shade
  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, "uPMatrix");
  shaderProgram.mvMatrixUniform =
    gl.getUniformLocation(shaderProgram, "uMVMatrix");
}
var mvMatrix = mat4.create();
var pMatrix = mat4.create();
```

```
mat4.translate(mvMatrix, [-1.5,
gl.bindBuffer(gl.ARRAY_BUFFER,
gl.vertexAttribPointer(shaderPr
  triangleVertexPositionBuffer.
  gl.FLOAT, false, 0, 0);
setMatrixUniforms();
gl.drawArrays(gl.TRIANGLES, 0,
mat4.translate(mvMatrix, [3.0,
gl.bindBuffer(gl.ARRAY_BUFFER,
gl.vertexAttribPointer(shaderPr
  squareVertexPositionBuffer.it
  gl.FLOAT, false, 0, 0);
setMatrixUniforms();
gl.drawArrays(gl.TRIANGLE_STRIP
```

```
    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;

drawScene() {
ort(0, 0, gl.viewportWidth, gl.viewportHeight);
(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
spective(45, 1.5, 0.1, 100.0, pMatrix);
entity(mvMatrix);
anslate(mvMatrix, [-1.5, 0.0, -7.0]);
Buffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
exAttribPointer(shaderProgram.vertexPositionAttribute,
gleVertexPositionBuffer.itemSize,
OAT, false, 0, 0);
ixUniforms();
Arrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
anslate(mvMatrix, [3.0, 0.0, 0.0]);
Buffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
exAttribPointer(shaderProgram.vertexPositionAttribute,
eVertexPositionBuffer.itemSize,
OAT, false, 0, 0);
ixUniforms();
Arrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);

webGLStart() {
vas = document.getElementById("lesson01-canvas");
canvas);
ders();
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;"
height="500">
</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

three.js

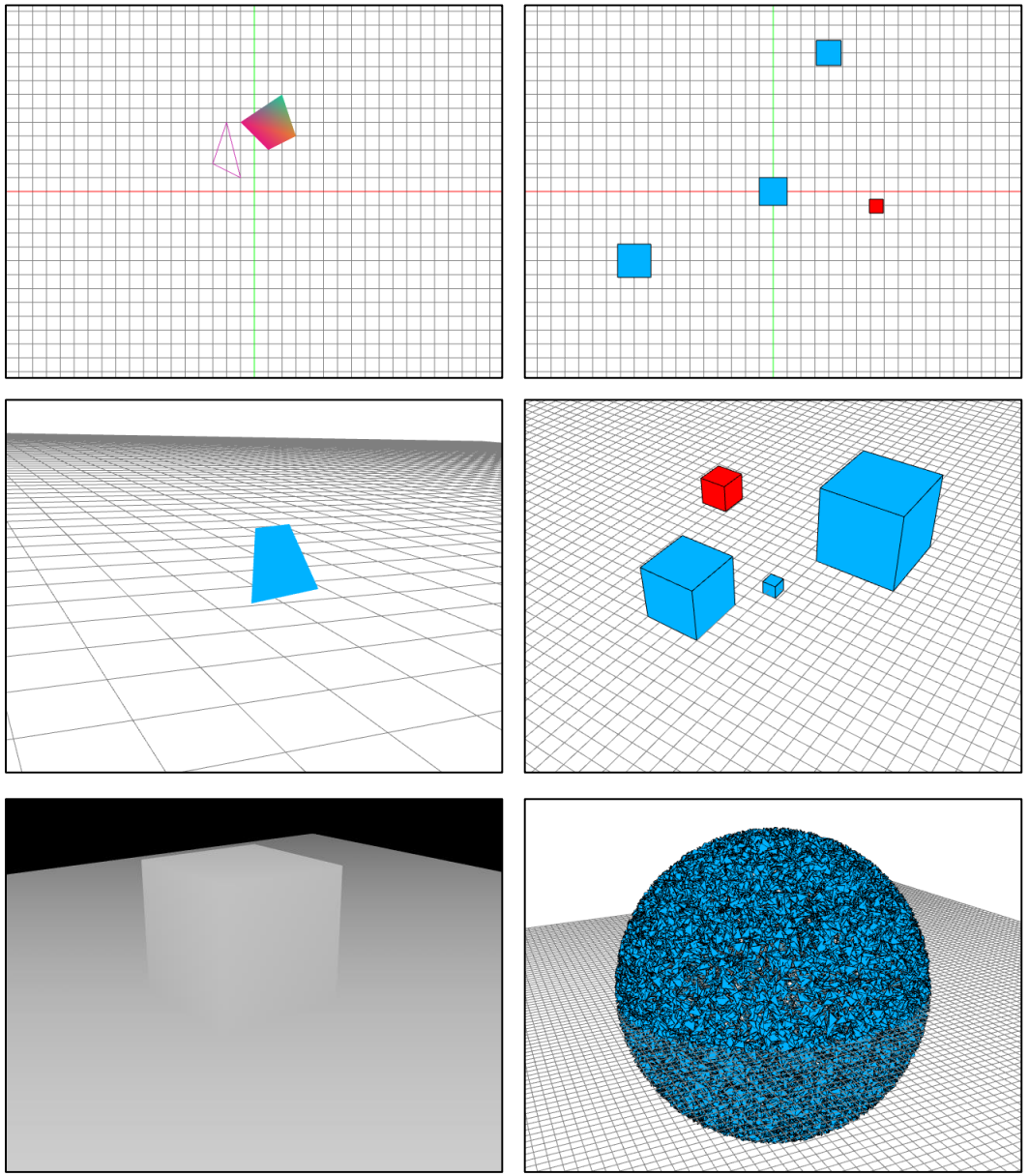
```
<script src="js/three.min.js"></script>
<script>
var camera, scene, renderer, geometry, material, mesh;
function init() {
  scene = new THREE.Scene();
  camera = new THREE.PerspectiveCamera( 75, 640 / 480, 1, 10000 );
  camera.position.z = 1000;
  geometry = new THREE.BoxGeometry( 200, 200, 200 );
  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>
```

全然OpenGLじゃない！

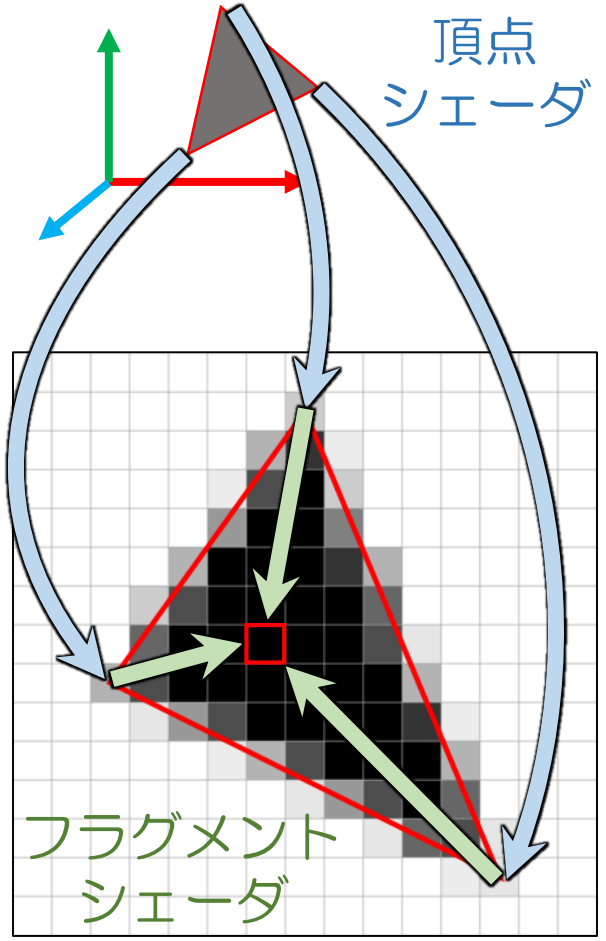
legacygl.js

- 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



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glDrawArraysの際に渡す
    - 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 (glDrawArrays)
  - 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

(最新バージョンでは文法が微妙に異なる)

```
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

WebGL開発環境

- テキストエディタ
  - Sublime Text：試用期限無し、コード補完
  - WebStorm　：有料だが最強？
  - Visual Studio：一応使える
  - Vim, Emacs, ...

  - Review: 10 JavaScript editors and IDEs put to the test
    (http://www.javaworld.com/article/2094847)

- ブラウザ：Chromeのデバッガが秀逸

JavaScript初心者 (二高山) 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/Homogeneous_coordinates
- http://en.wikipedia.org/wiki/Perspective_(graphical)
- http://en.wikipedia.org/wiki/Z-buffering
- http://en.wikipedia.org/wiki/Quaternion