

# Introduction to Computer Graphics

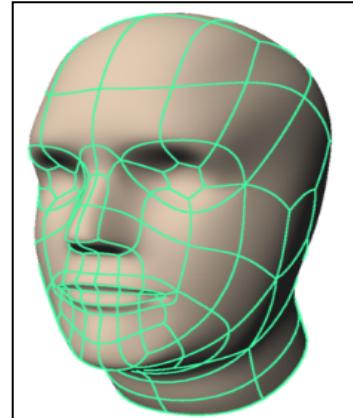
April 9, 2020

Kenshi Takayama

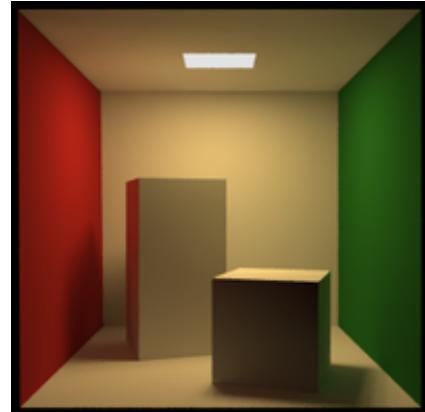
# Course overview

- Lecture basic stuff on 4 topics
- 2~3 lectures per topic, 12 lectures in total

Modeling



Rendering



Animation

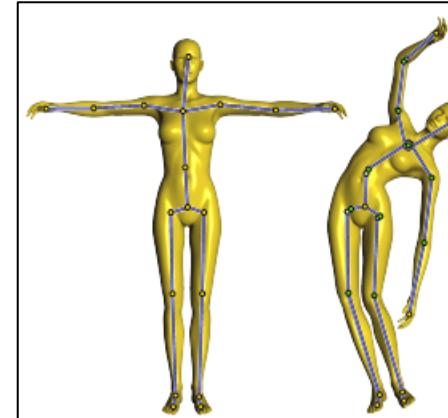
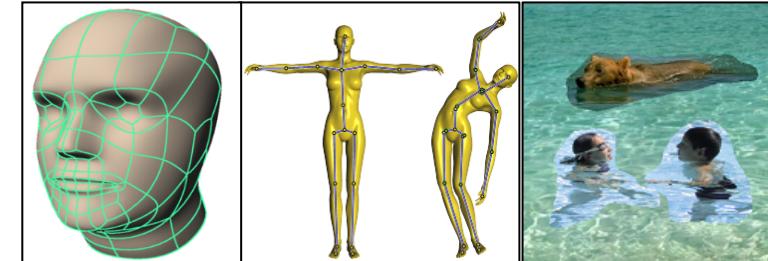


Image processing



# Lecturers

- Kenshi Takayama (Assistant Prof., @NII)
  - <http://research.nii.ac.jp/~takayama/>
  - [takayama@nii.ac.jp](mailto:takayama@nii.ac.jp)
- Toshiya Hachisuka (Associate Prof. @UTokyo)
  - <http://www.ci.i.u-tokyo.ac.jp/~hachisuka/>
  - [thachisuka@siggraph.org](mailto:thachisuka@siggraph.org)
  - Move to Waterloo on Sep 2020
- Ryoichi Ando (Assistant Prof., @NII)
  - <http://research.nii.ac.jp/~rand/>
  - [rand@nii.ac.jp](mailto:rand@nii.ac.jp)
- Nobuyuki Umetani (Associate Prof. @UTokyo)
  - <http://nobuyuki-umetani.com/>
  - [n.umetani@gmail.com](mailto:n.umetani@gmail.com)



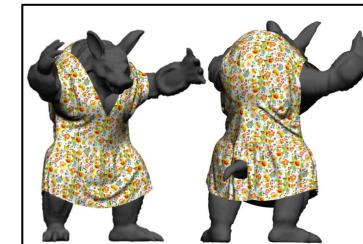
Modeling  
Animation  
Image processing



Rendering



Animation (fluids)



Animation (deformables)

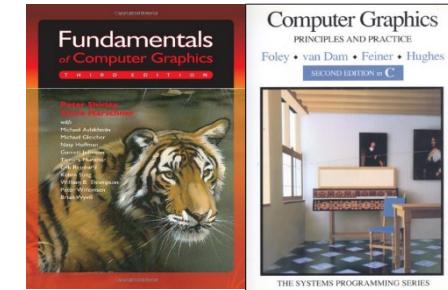
TA : Daiki Harada @Igarashi Lab  
[rijicho.cgta2020@gmail.com](mailto:rijicho.cgta2020@gmail.com)

# Grading

- Programming assignments only
  - No exam, no attendance check
- Two types of assignments: Basic & Advanced
  - Basic: 1 assignment per topic (4 in total), very easy
  - Advanced: For motivated students
- Deadline:
  - Basic: 2 weeks after announcement
  - Advanced: end of July
- Evaluation criteria
  - 1 assignment submitted → **C** (bare minimum for the degree)
  - 4 assignments submitted → **B** or higher
  - Distribution of **S** & **A** will be decided based on the quality/creativity of submissions and the overall balance in the class
- More details explained later

# References

- Course website
  - <http://research.nii.ac.jp/~takayama/teaching/utokyo-iscg-2020/>
- Famous textbooks (not used in the class)
  - Fundamentals of Computer Graphics (9781568814698)
  - Computer Graphics: Principles and Practice in C (9780201848403)



# Coordinate transformations

# Linear transformation

$$\text{In 2D: } \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

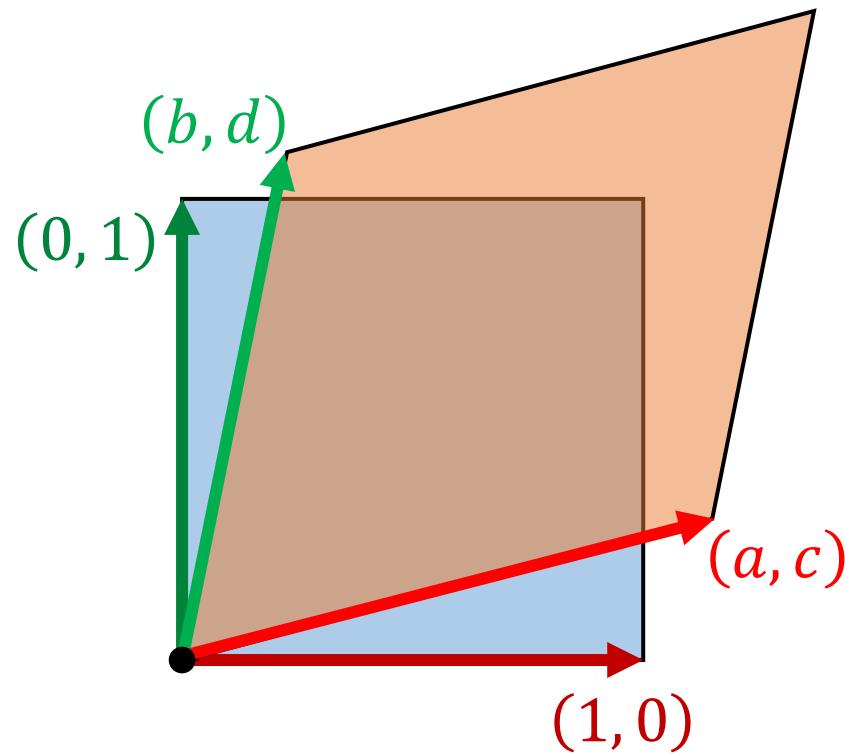
$$\text{In 3D: } \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Intuition: Mapping of coordinate axes

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

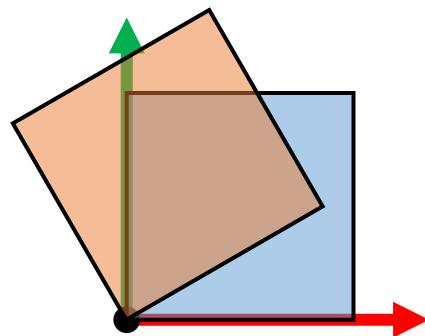
$$\begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Origin stays put

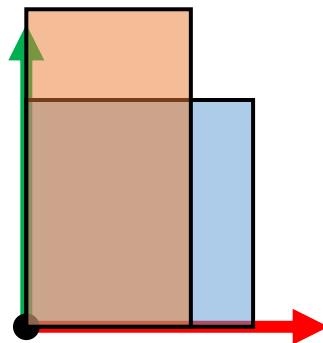


# Special linear transformations

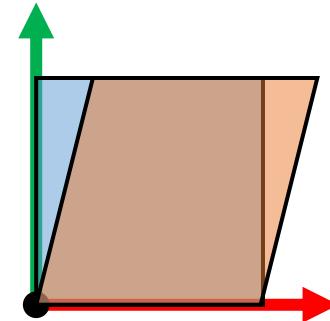
Rotation



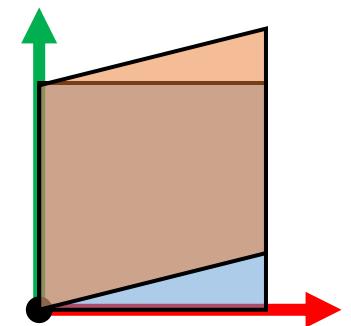
Scaling



Shearing (X dir.)



Shearing (Y dir.)



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

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

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

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

# Linear transformation + translation = Affine transformation

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

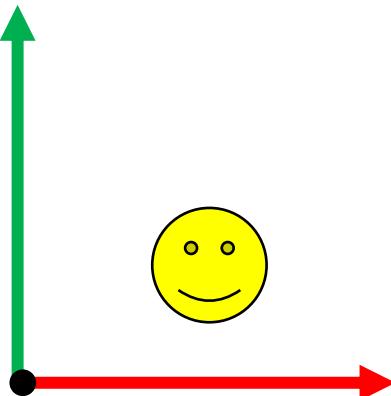
- **Homogeneous coordinates:** Use a 3D (4D) vector to represent a 2D (3D) point
- Can concisely represent linear transformation & translation as matrix multiplication
  - Easier implementation

# Combining affine transformations

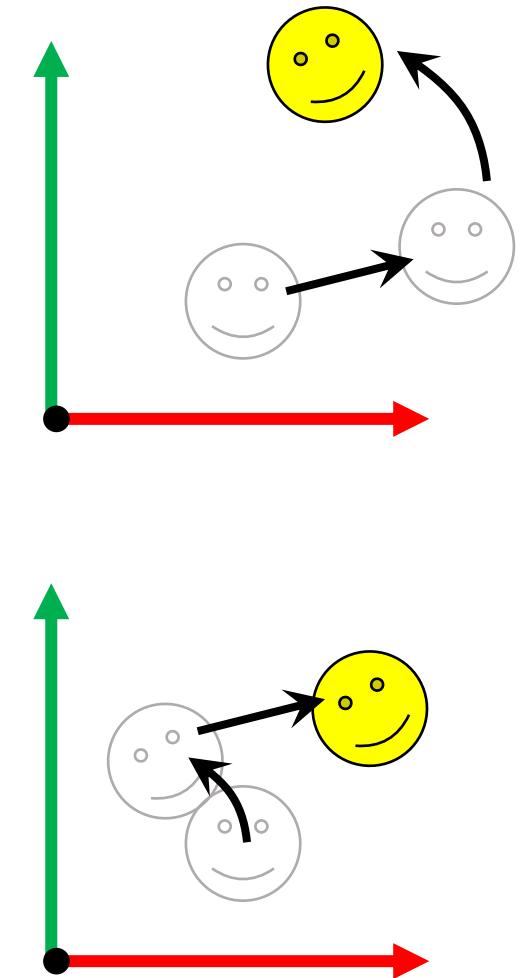
- Just multiply matrices
- Careful with the ordering!

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

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

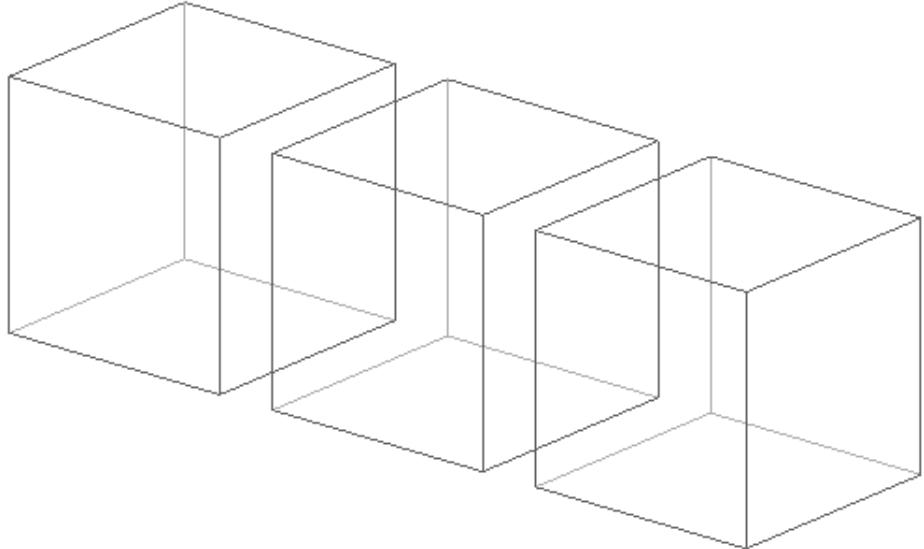


$$\mathbf{x}' = R T \mathbf{x}$$
$$\mathbf{x}' = T R \mathbf{x}$$

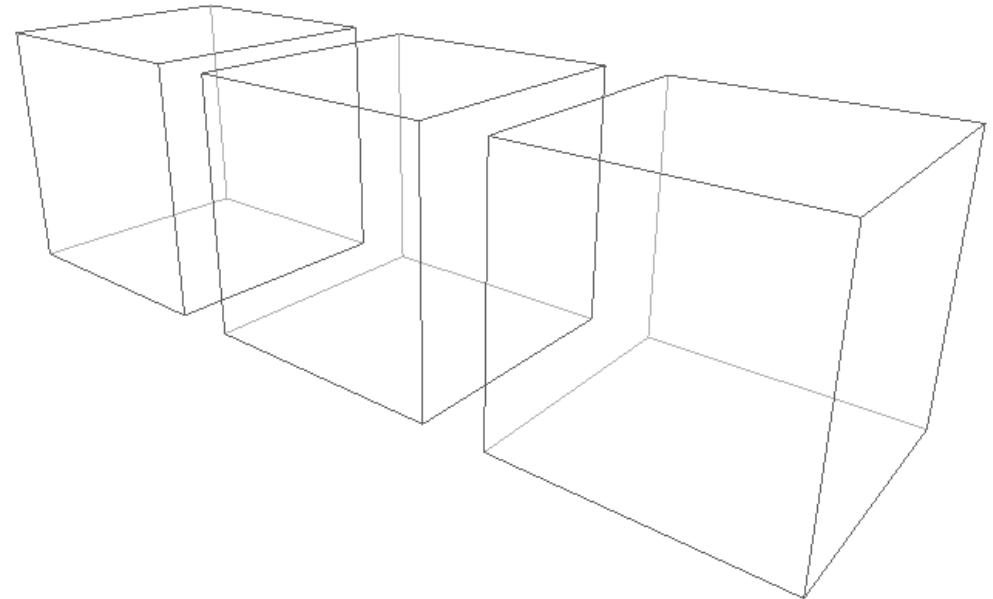


# Another role of homogeneous coordinates: Perspective projection

- Objects' apparent sizes on the screen are inversely proportional to the object-camera distance



Orthographic projection



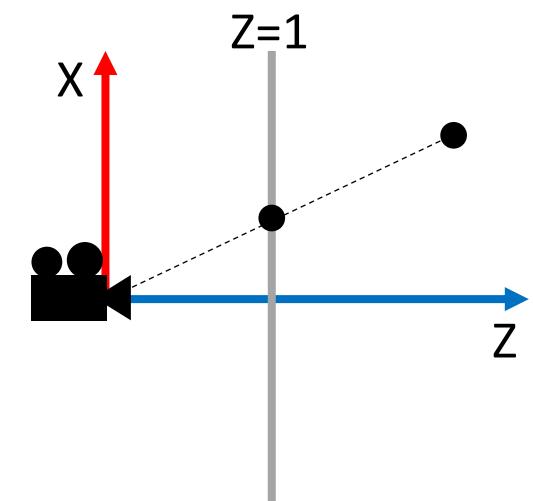
Perspective projection

# Another role of homogeneous coordinates: Perspective projection

- When  $w \neq 0$ , 4D homogeneous coordinate  $(x, y, z, w)$  represents a 3D position  $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$
- Camera at the origin, screen on the plane  $Z=1$   
→  $(p_x, p_y, p_z)$  is projected to  $(w_x, w_y) = \left(\frac{p_x}{p_z}, \frac{p_y}{p_z}\right)$

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

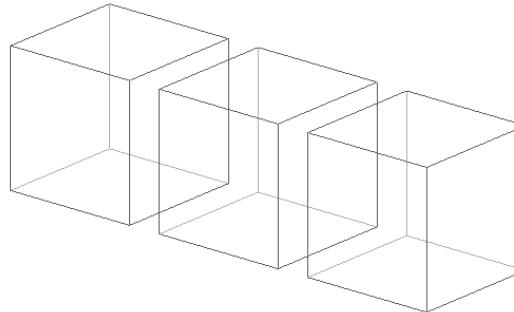
Projection matrix



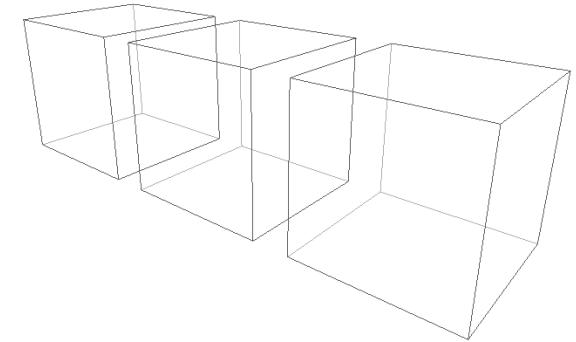
- $w_z$  (depth value) is used for occlusion test → Z-buffering

# Orthographic projection

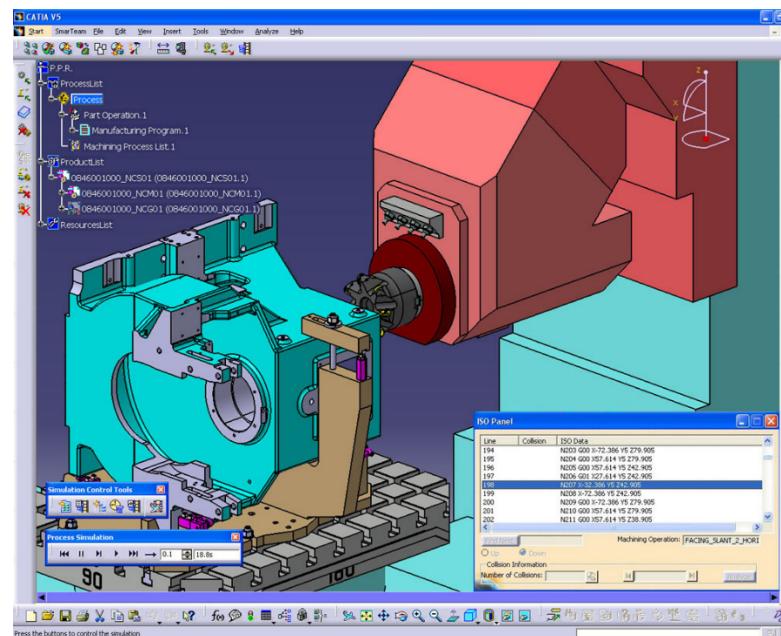
- Objects' apparent sizes don't depend on the camera position
- Simply ignore Z coordinates
- Frequently used in CAD



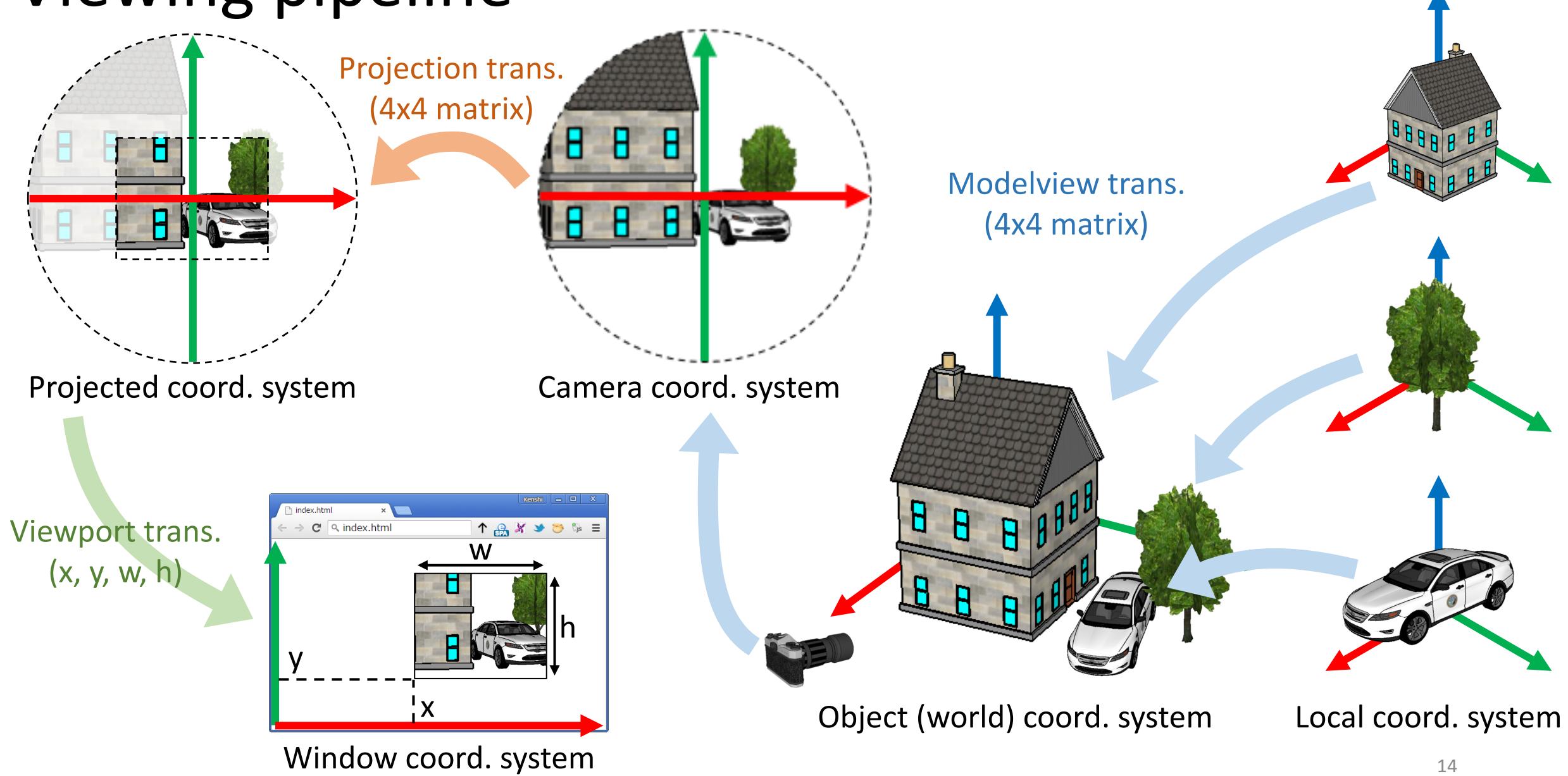
Orthographic



Perspective



# Viewing pipeline



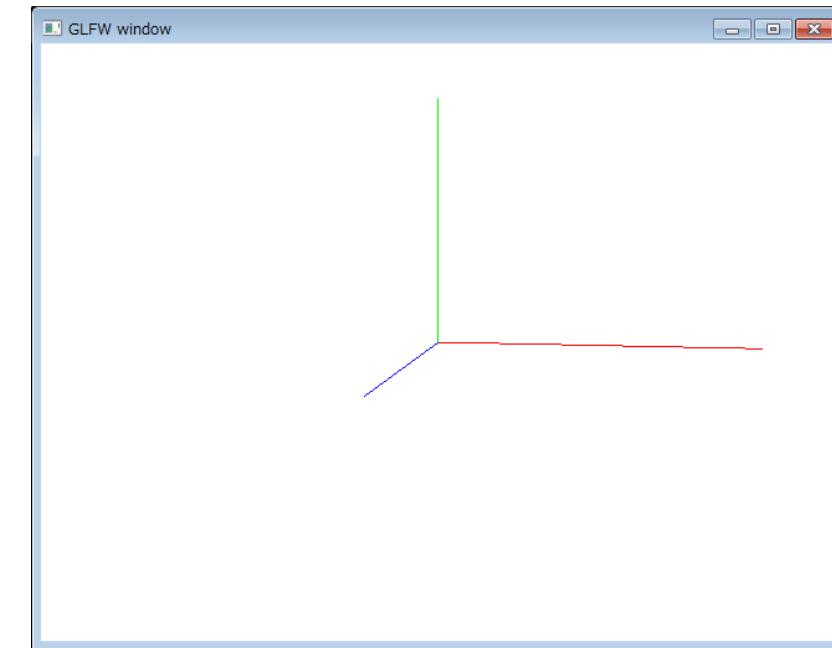
# Classical OpenGL code

```
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();
```

} Viewport transform

} Projection transform

} Modelview transform

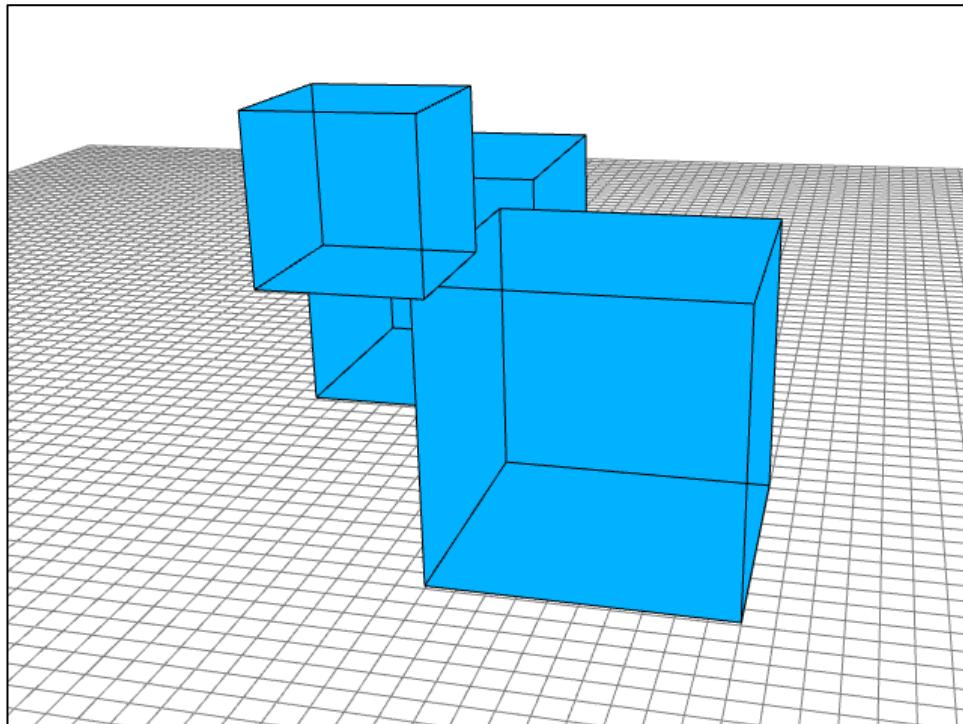


Output

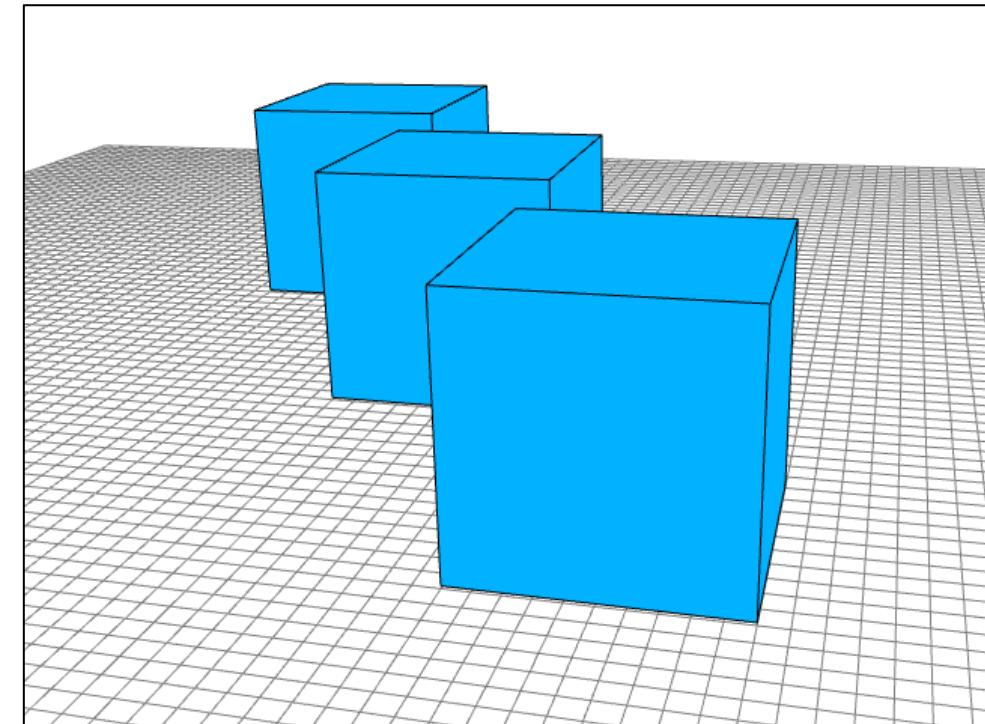
} Scene content

# Z-buffering

# Hidden surface removal



Without hidden surface removal

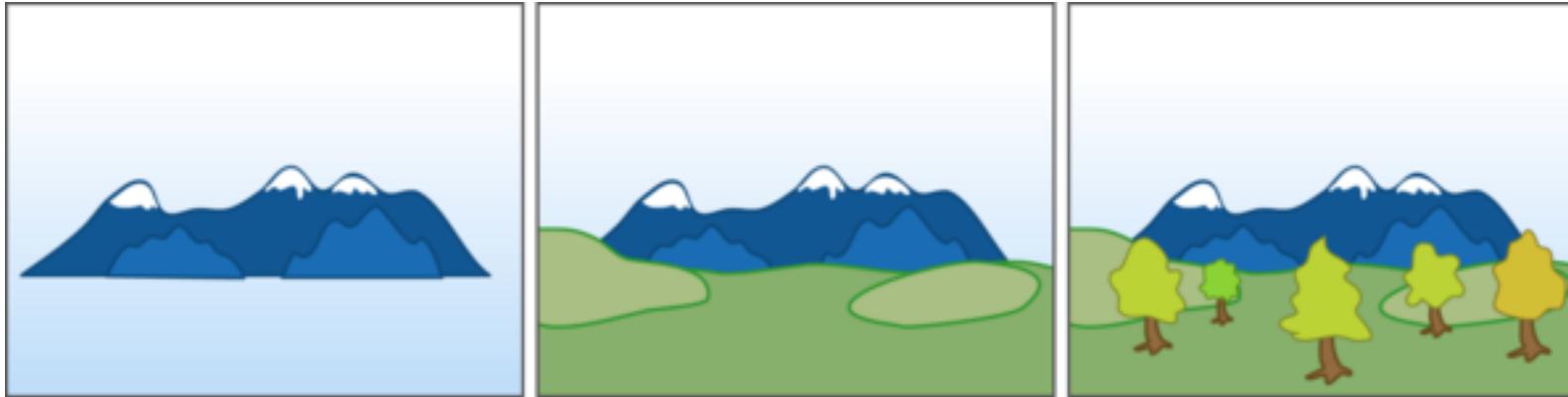


With hidden surface removal

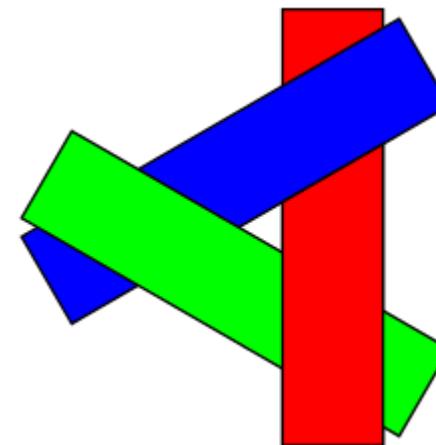
- Classic problem in CG

# Painter's algorithm

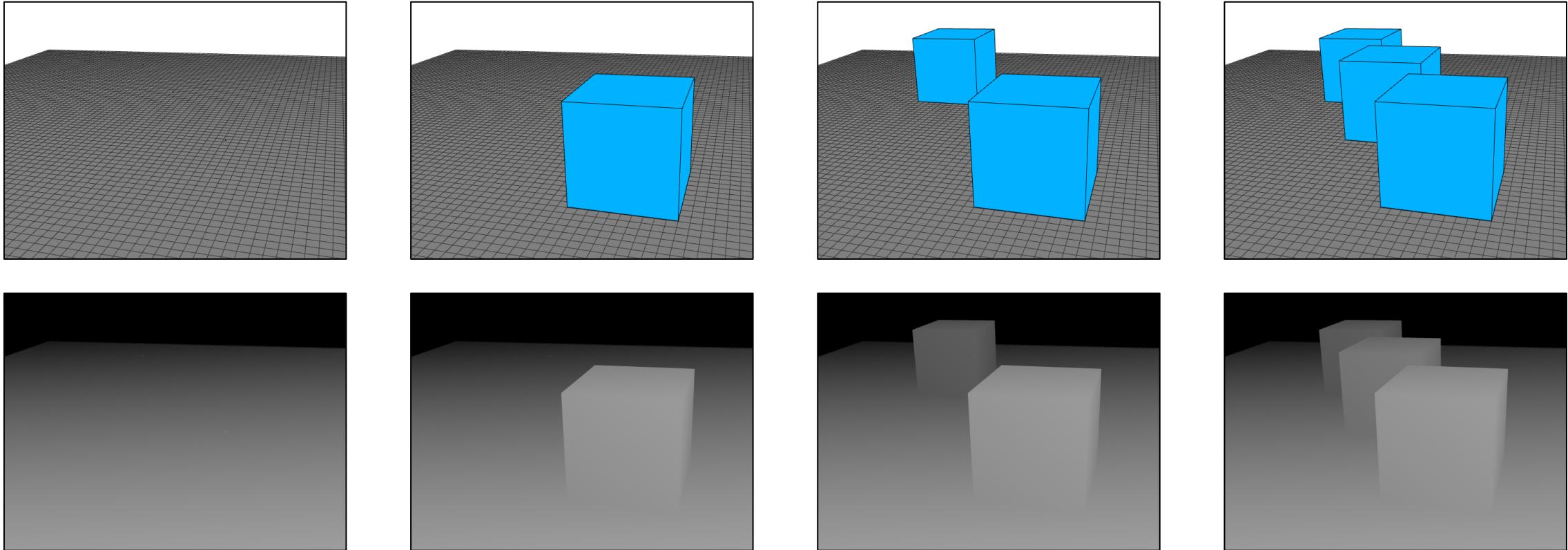
- Sort objects according to distances to camera, then draw them in the back-to-front order



- Fundamentally ill-suited for many cases
  - Sorting is also not always straightforward



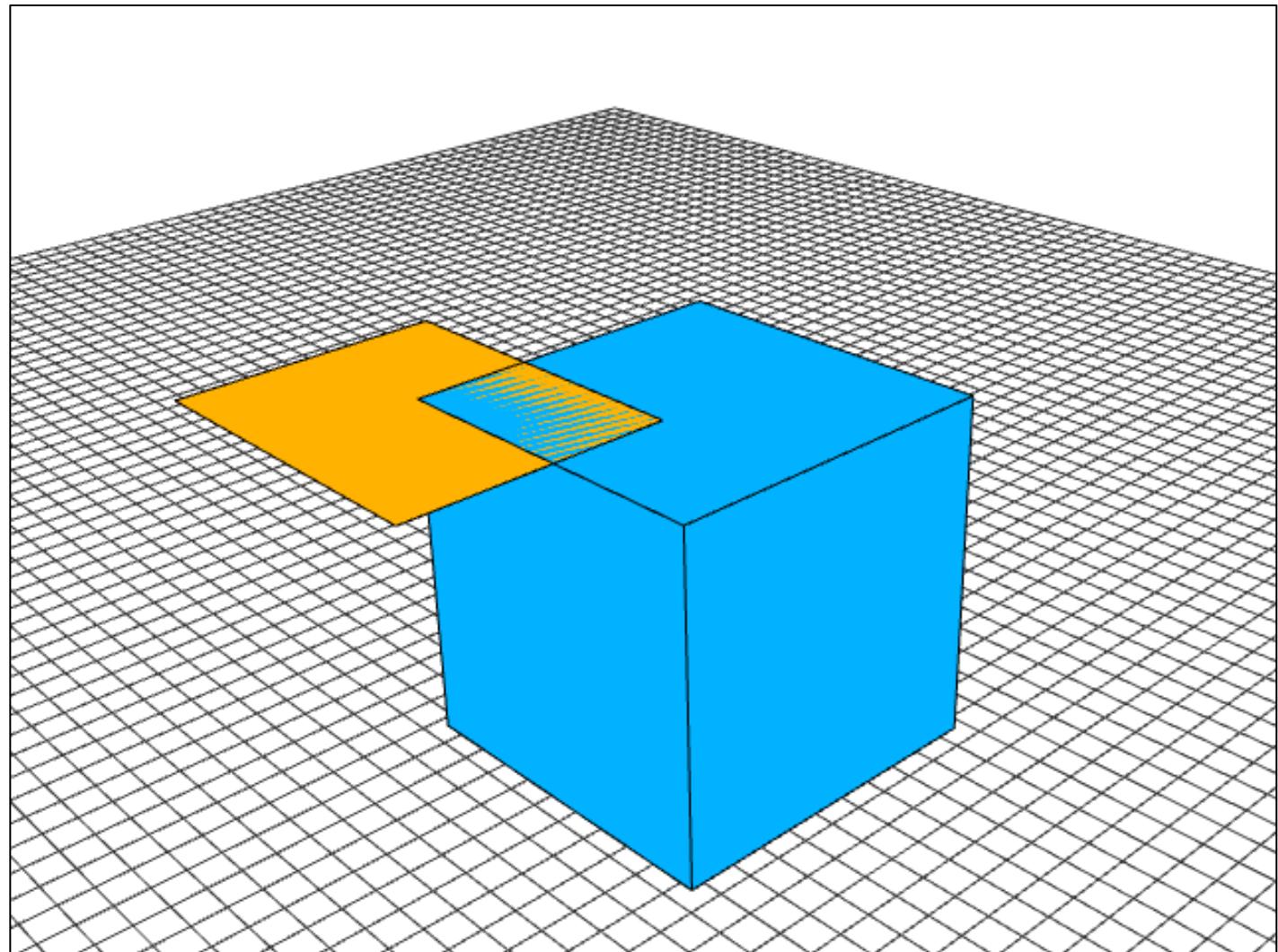
# Z-buffering



- For each pixel, store distance to the camera (depth)
- More memory-consuming, but today's standard

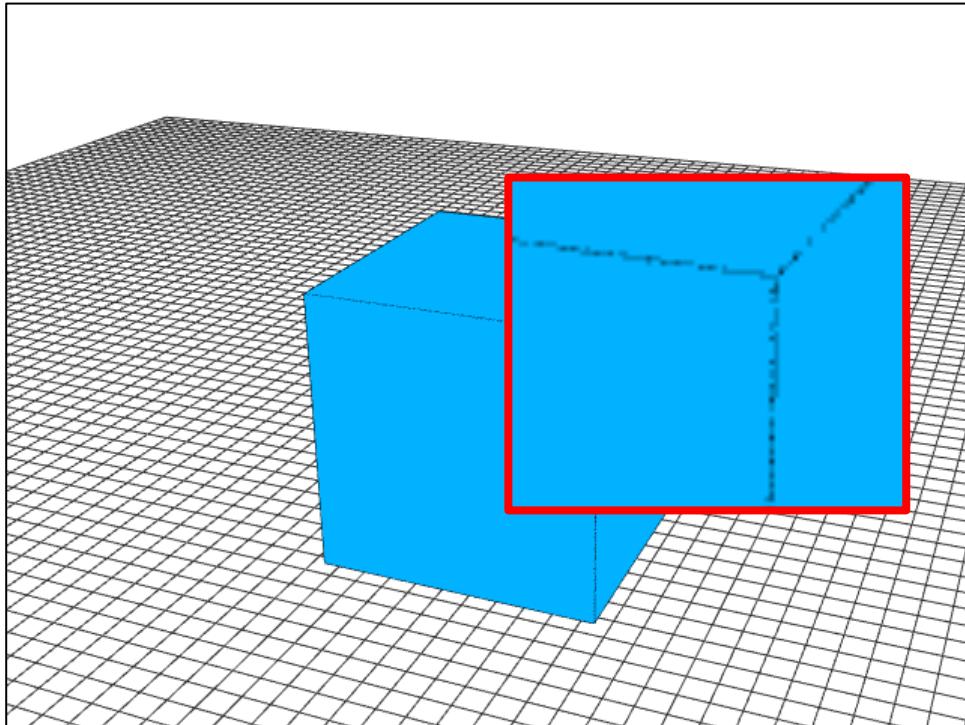
# Typical issues with Z-buffering: Z-fighting

- Multiple polygons at exact same position
- Impossible to determine which is front/back
- Strange patterns due to rounding errors

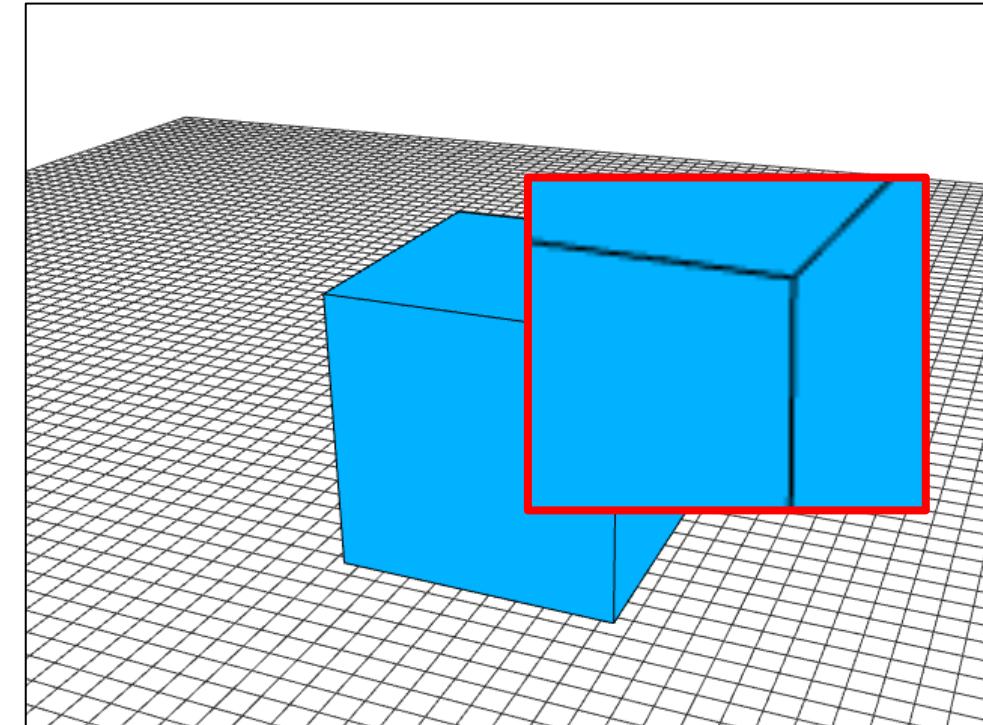


# Typical issues with Z-buffering: Simultaneous drawing of faces and lines

- Dedicated OpenGL trick: `glPolygonOffset`



Without polygon offset

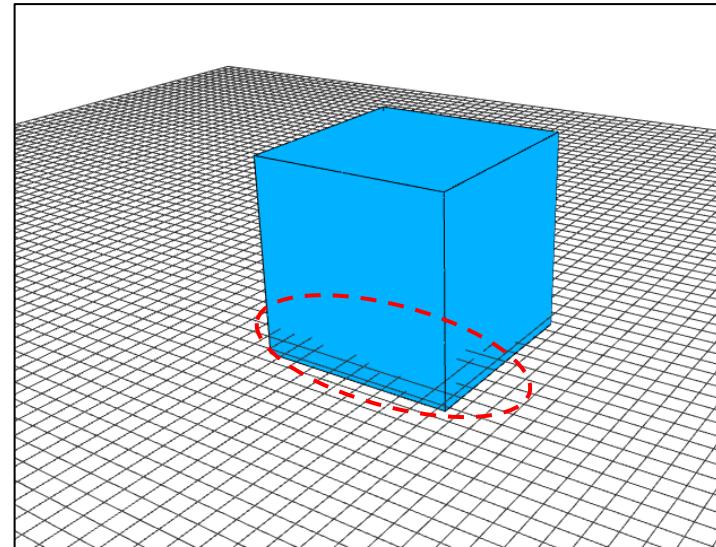


With polygon offset

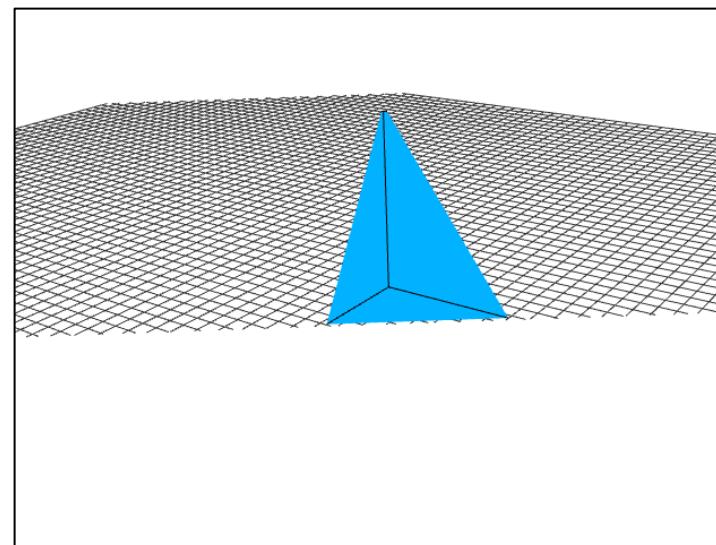
# Typical issues with Z-buffering: Depth range

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

- Fixed bits for Z-buffer
  - Typically, 16~24bits
- Larger depth range
  - Larger drawing space, less accuracy
- Smaller depth range
  - More accuracy, smaller drawing space (clipped)



zNear=0.0001  
zFar =1000



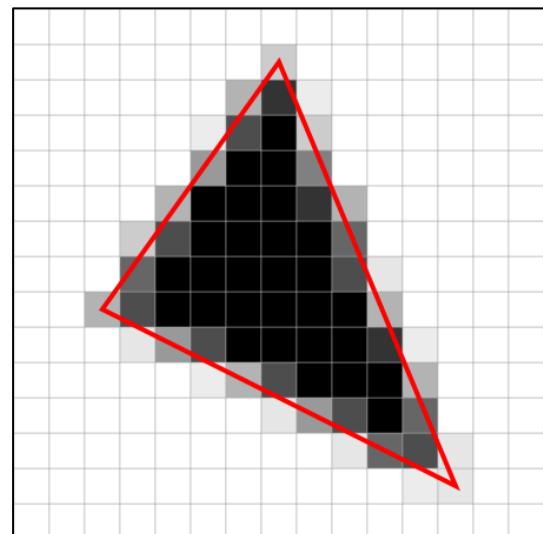
zNear=50  
zFar =100

# Rasterization vs Ray-tracing

Purpose

Real-time CG (games)

Idea



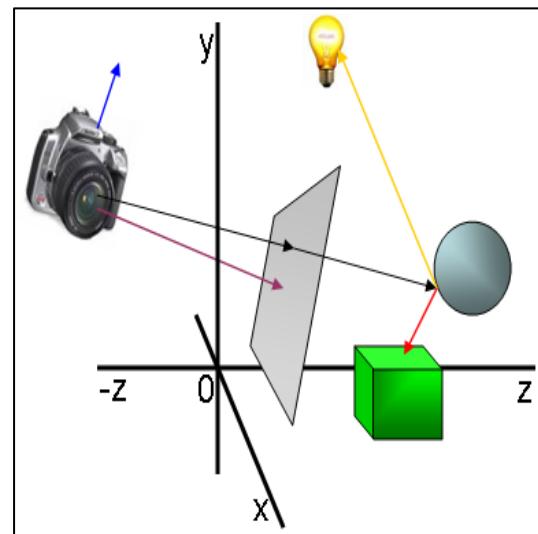
One polygon  
updates multiple  
pixels

Hidden surface  
removal

Z-buffering  
(OpenGL / DirectX)

High-quality CG (movies)

Per-pixel (ray) processing



One ray interacts  
with multiple  
polygons

By nature

More details by Prof. Hachisuka

# Quaternions

# Rotation about arbitrary axis

- Needed in various situations (e.g. camera manipulation)

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \text{about X-axis}$$
$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \text{about Y-axis}$$
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{about Z-axis}$$

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

(u<sub>x</sub>, u<sub>y</sub>, u<sub>z</sub>) : axis vector

- Problems with matrix representation

- Overly complex!

Degree of Freedom

- Should be represented by 2 DoF (axis direction) + 1 DoF (angle) = 3 DoF
- Can't handle interpolation (blending) well

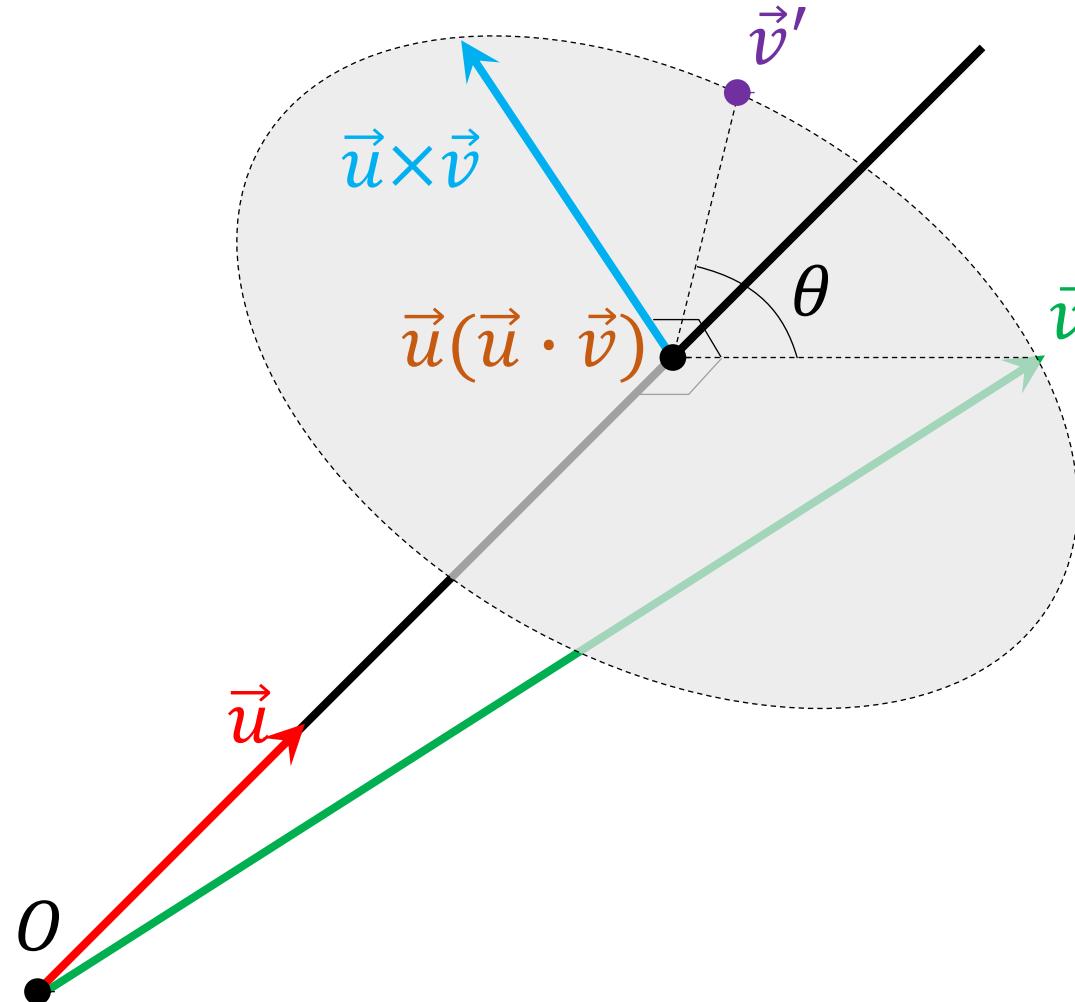
# Geometry of axis-angle rotation

$\vec{u}$ : axis (unit vector)

$\theta$ : angle

$\vec{v}$ : input position

$\vec{v}'$ : output position



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

# Complex number & quaternion

- Complex number
  - $\mathbf{i}^2 = -1$
  - $\mathbf{c} = (a, b) := a + b \mathbf{i}$
  - $\mathbf{c}_1 \mathbf{c}_2 = (a_1, b_1)(a_2, b_2) = a_1 a_2 - b_1 b_2 + (a_1 b_2 + b_1 a_2) \mathbf{i}$
- Quaternion
  - $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$
  - $\mathbf{ij} = -\mathbf{ji} = \mathbf{k}, \quad \mathbf{jk} = -\mathbf{kj} = \mathbf{i}, \quad \mathbf{ki} = -\mathbf{ik} = \mathbf{j}$  Not commutative!
  - $\mathbf{q} = (a, b, c, d) := a + b \mathbf{i} + c \mathbf{j} + d \mathbf{k}$
  - $\mathbf{q}_1 \mathbf{q}_2 = (a_1, b_1, c_1, d_1)(a_2, b_2, c_2, d_2)$ 
$$= (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) \mathbf{i}$$
$$+ (a_1 c_2 + c_1 a_2 + d_1 b_2 - b_1 d_2) \mathbf{j} + (a_1 d_2 + d_1 a_2 + b_1 c_2 - c_1 b_2) \mathbf{k}$$

# Notation by scalar + 3D vector

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

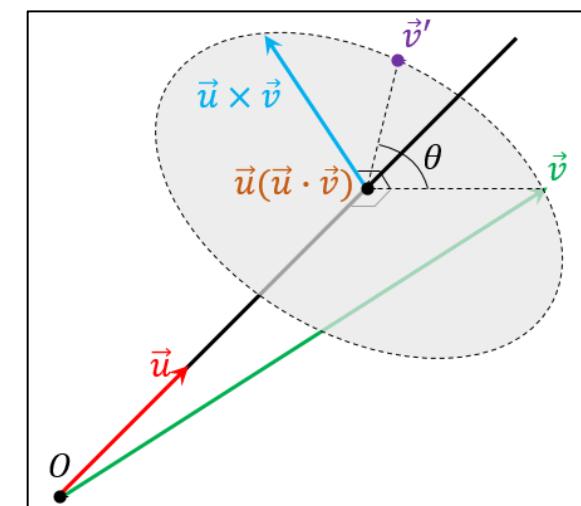
# Rotation using quaternions

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

Note:  $\vec{u}$  is a unit vector

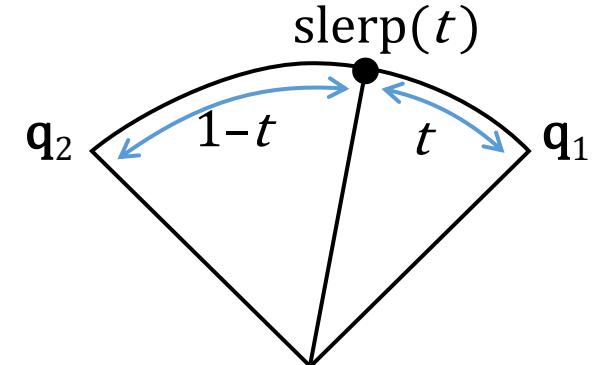
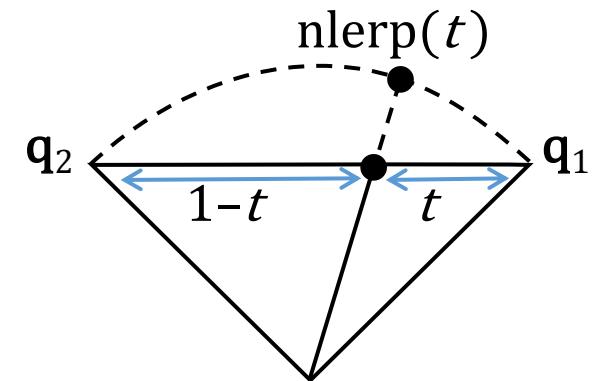
$$\begin{aligned}\vec{v}' &= q\vec{v}q^{-1} = \left(\cos \frac{\alpha}{2} + \vec{u} \sin \frac{\alpha}{2}\right) \vec{v} \left(\cos \frac{\alpha}{2} - \vec{u} \sin \frac{\alpha}{2}\right) \\ &= \vec{v} \cos^2 \frac{\alpha}{2} + (\vec{u}\vec{v} - \vec{v}\vec{u}) \sin \frac{\alpha}{2} \cos \frac{\alpha}{2}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

- Interesting theory behind
  - Clifford algebra
  - Geometric algebra
- Also important for physics & robotics



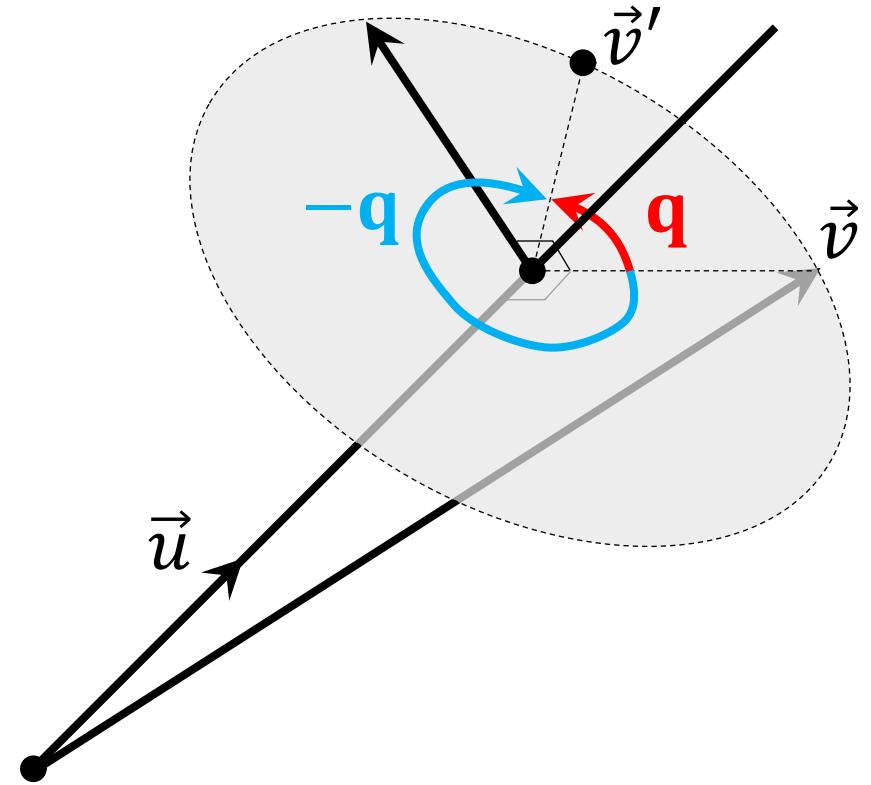
# Rotation interpolation using quaternions

- Linear interp + normalization (nlerp)
  - $\text{nlerp}(\mathbf{q}_1, \mathbf{q}_2, t) := \text{normalize}((1 - t)\mathbf{q}_1 + t \mathbf{q}_2)$
  - ☺less computation, ☹non-uniform angular speed
- Spherical linear interpolation (slerp)
  - $\Omega = \cos^{-1}(\mathbf{q}_1 \cdot \mathbf{q}_2)$
  - $\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, t) := \frac{\sin(1-t)\Omega}{\sin \Omega} \mathbf{q}_1 + \frac{\sin t\Omega}{\sin \Omega} \mathbf{q}_2$
  - ☹more computation, ☺constant angular speed



# Signs of quaternions

- Quaternion with angle  $\theta$ :
  - $\mathbf{q} = \cos \frac{\theta}{2} + \vec{u} \sin \frac{\theta}{2}$
- Quaternion with angle  $\theta - 2\pi$ :
  - $\cos \frac{\theta-2\pi}{2} + \vec{u} \sin \frac{\theta-2\pi}{2} = -\mathbf{q}$
- When interpolating from  $\mathbf{q}_1$  to  $\mathbf{q}_2$ , negate  $\mathbf{q}_2$  if  $\mathbf{q}_1 \cdot \mathbf{q}_2$  is negative
  - Otherwise, the interpolation path becomes longer



# How to work on assignments

# Choices for implementing real-time CG

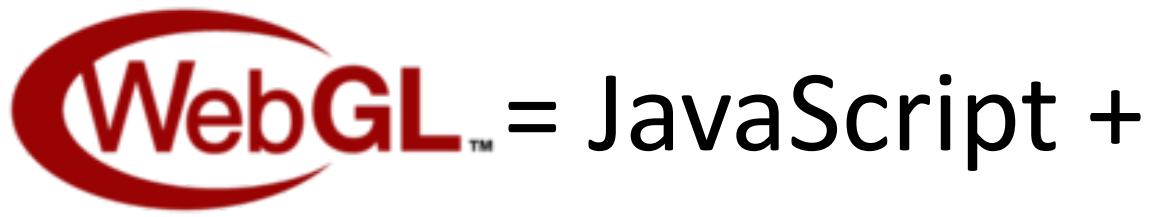
- Two kinds of APIs for using GPU
  - Different API designs (slightly?)
  - Both supported by most popular programming languages
- Many choices for system- & language-dependent parts
  - GUI management, handling images, ...
  - Many libraries:
    - GUI: GLUT (C), GLFW (C), SDL (C), Qt (C++), MFC (C++), wxWidgets (C++), Swing (Java), ...
    - Images: libpng, OpenCV, ImageMagick
- Often quite some work to get started



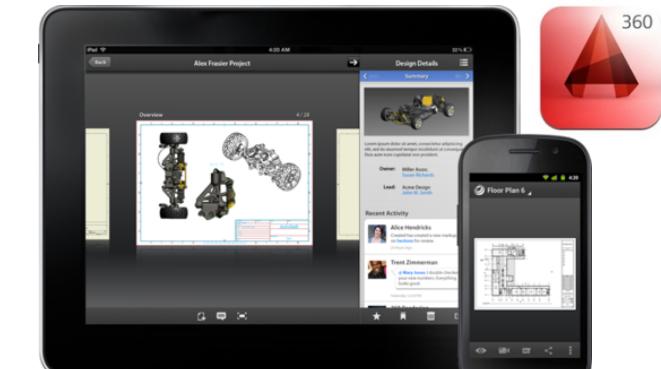
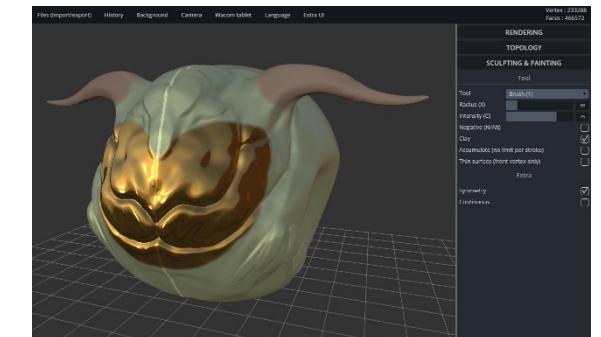
OpenGL®



Microsoft  
DirectX



- Runs on many (mobile) browsers
- HTML-based → can easily handle multimedia & GUI
- No compiling!
  - Quick trial & error
- Some performance concerns
- Increasingly popular today



# Hurdle in WebGL development: OpenGL ES

- No support for legacy OpenGL API
- Reasons:
  - Less efficient
  - Burden on hardware vendors

- Allowed API:

Prepare arrays, send them to GPU, draw them using custom shaders

Immediate mode  
Polygonal primitives  
Light & material  
Transform. matrices  
Display list  
Default shaders

Shaders  
Shader variables  
Arrays  
Drawing

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

glCreateShader, glShaderSource,  
glCompileShader, glCreateProgram,  
glAttachShader, glLinkProgram,  
glUseProgram  
glGetAttribLocation,  
glEnableVertexAttribArray,  
glGetUniformLocation, glUniform  
glCreateBuffer, glBindBuffer,  
glBufferData, glVertexAttribPointer  
glDrawArrays

```
#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(display);
    setDispList();
    glutMainLoop();
}
```



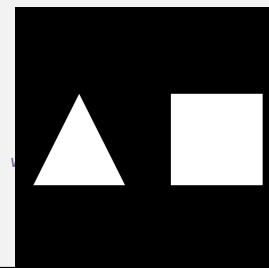
## C / OpenGL 1.x

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

```
<html><head>
<title>Learning WebGL &mdash; lesson 1</title>
<script type="text/javascript" src="glMatrix-0.9.5.min.js">
<script id="shader-fs" type="x-shader/x-fragment">
precision medium 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("webgl");
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
}
function getShader(gl, i) {
    var shaderScript = document.getElementById('shader'+i);
    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-fragment")
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    else if (shaderScript.type == "x-shader/x-vertex")
        shader = gl.createShader(gl.VERTEX_SHADER);
    gl.shaderSource(shader, str);
    gl.compileShader(shader);
    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();
var vertices = [
    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;
</script></head>
<body onload="webGLStart();">
<canvas id="lesson01-canvas" style="border: none;" width="400" height="300">
</canvas>
</body> </html>
```

## WebGL

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



# Libraries for easing WebGL development

- Many popular ones:
  - three.js, O3D, OSG.JS, ...
- All APIs are high-level, quite different from legacy OpenGL API 😞
- Good for casual users, but maybe not for CS students (?)

```
<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, 1000 );
    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>
```

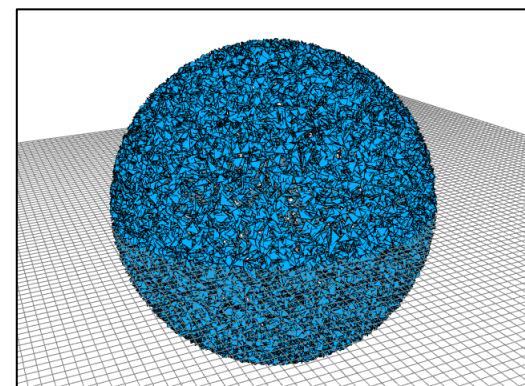
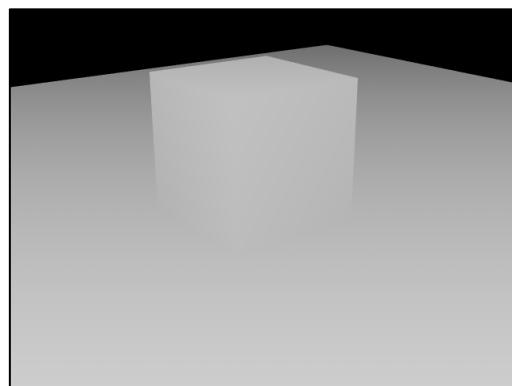
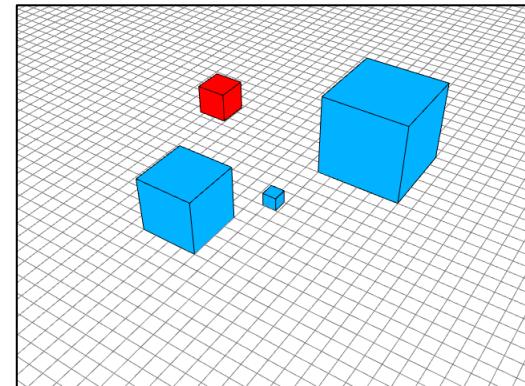
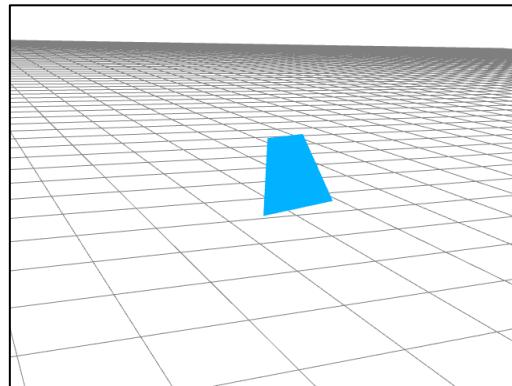
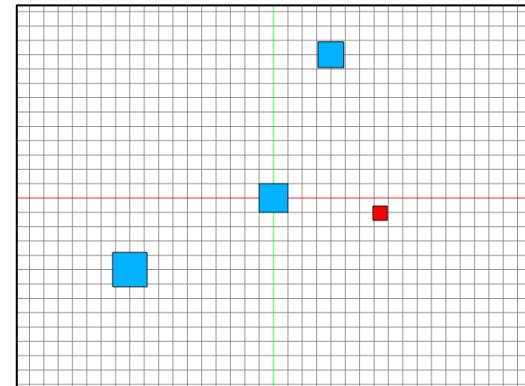
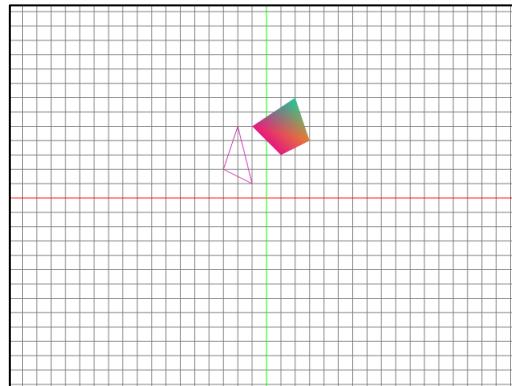
three.js

High-level API



# legacygl.js

- Developed by me for this course
  - <https://bitbucket.org/kenshi84/legacygl.js>
  - Demos & tutorial
- Assignemnts' sample codes will be mostly using this
  - Try playing with it and see how it works



# WebGL development using Glitch

- A free web space for putting js/html/css
- Online editing and quick previewing

<https://glitch.com/>

The screenshot shows the Glitch web interface. At the top, the URL is https://glitch.com/edit/#!/legacygl-js?path=demo/hello2d.html:1:0. The page title is "legacygl.js Demo: Hello World 2D". On the left, there's a file tree with "assets" and several "demo" files: "displist.html", "hello2d.html" (selected), "hello3d.html", "meshviewer.html", "pick2d.html", "pick3d.html", "quaternion... c.html", "z-buffer.html", "boundingbox.js", "camera.js", "colormap.js", "drawutil.js", "gl-matrix-util.js", "gl-matrix.js", "glu.js", and "halfedge.js". The main area contains the source code for "hello2d.html":

```
1<html>
2
3<head>
4<title>legacygl.js Demo: Hello World 2D</title>
5<script src="../gl-matrix.js"></script>
6<script src="../gl-matrix-util.js"></script>
7<script src="../legacygl.js"></script>
8<script src="../drawutil.js"></script>
9<script src="../camera.js"></script>
10<script src="../util.js"></script>
11<script type="text/javascript">
12var gl;
13var canvas;
14var legacygl;
15var drawutil;
16var camera;
17
18function draw() {
19    gl.clear(gl.COLOR_BUFFER_BIT);
20    // projection and camera position
21    var eyez = camera.eye[2];
22    mat4.perspective(legacygl.uniforms.projection.value, Math
23    camera.lookAt(legacygl.uniforms.modelview.value);
24    // xy-grid
25    legacygl.color(0.5, 0.5, 0.5);
```

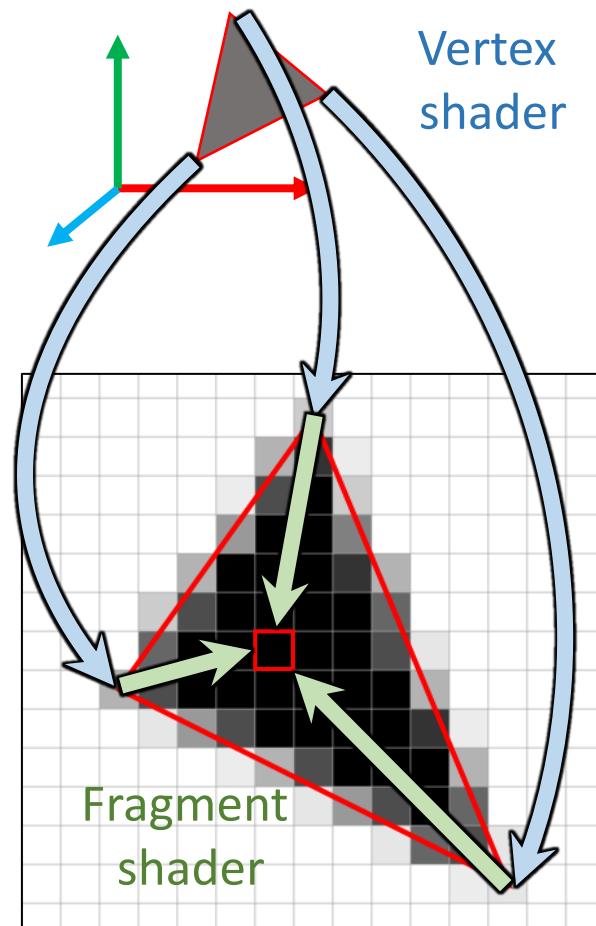
The preview window shows a 2D coordinate system with a grid. A pink triangle is positioned in the lower-left quadrant, and a colorful 3D cube is positioned in the upper-right quadrant.

# How to work on assignments

- Implement your solution using WebGL, upload it to the web, email the URL to the TA
  - Glitch is recommended, but other means (e.g. your own server) is also OK
  - Include some descriptions/discussions/etc in the HTML page
  - OK to use other WebGL libraries (e.g. three.js)
- Other programming languages (e.g. C++) are also allowed
  - Should compile and run on typical computing systems
  - Include source+binary+doc in a single ZIP file
- If you have any questions, don't hesitate to contact TA or me!

# Shaders

- Vertex shader: per-vertex processing
  - Per-vertex data passed by `glBufferData`
    - Vertex position, color, texture coordinate, ...
  - Mandatory operation: Specify vertex location on the screen after coordinate transformation (`gl_Position`)
- Fragment shader: per-pixel processing
  - Do something with rasterized (=linearly interpolated) data
  - Mandatory operation: Specify pixel color to be drawn (`gl_FragColor`)
- GLSL (OpenGL Shading Language) codes passed to GPU as strings  
→ compiled at runtime



# Shader variables

- uniform variables
  - Readable from vertex/fragment shaders
  - Passed to GPU separately from vertex arrays (`glUniform`)
  - Examples: modelview/projection matrices, flags
- attribute variables
  - Readable only from vertex shaders
  - Vertex array data passed to GPU via `glBufferData`
  - Examples: XYZ position, RGB color, UV texcoord
- varying variables
  - Written by vertex shader, read by fragment shader
  - Per-vertex data linearly interpolated at this pixel

(Grammar differs slightly with versions)

```
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;
}
```

Vertex shader

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

Fragment shader

# Tips for JavaScript beginners (=me)

- 7 types: String / Bool / Number / Function / Object / null / undefined
  - Unlike C++
- Number: always double precision
  - No distinction between integer & floating point
- Object: associative map with string keys
  - `x.abc` is equivalent to `x["abc"]` (as if a “member”)
  - `{ abc : y }` is equivalent to `{ "abc" : y }`
  - Non-string keys are implicitly converted to strings
- Arrays are special objects with keys being consecutive integers
  - With additional capabilities: `.length` `.push()` `.pop()` `.forEach()`
- Always pass-by-value when assigning & passing arguments
  - No language support for “deep copy”
- When in doubt, use `console.log(x)`

# References

- OpenGL
  - Official spec
    - <https://www.opengl.org/sdk/docs/man/html/indexflat.php>
- WebGL/JavaScript/HTML5
  - Learning WebGL
    - <http://learningwebgl.com/blog/?p=11>
  - Official spec
    - <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/>

# References

- [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>