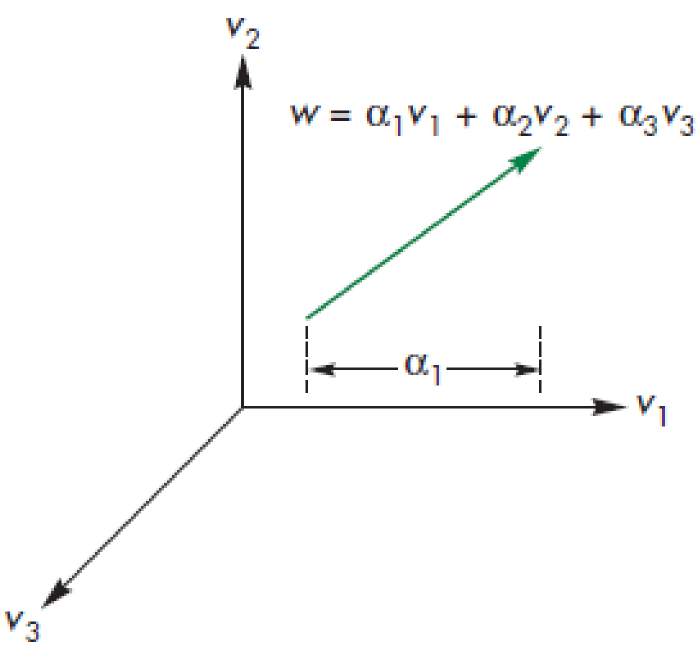
COSC 414/519I: Computer Graphics

2023W2

Shan Du

Coordinate Systems and Frames

- In a 3D vector space, we can represent any vector *w* uniquely in terms of any three linearly independent vector v_1, v_2, and v_3.



Vector derived from three basis vectors.

The scalars α_1, α_2, and α_3 are the components of *w* with respect to the basis v_1, v_2, and v_3.

Coordinate Systems and Frames

- We can rewrite the representation of w with respect to this basis as the column matrix

$$\boldsymbol{a} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

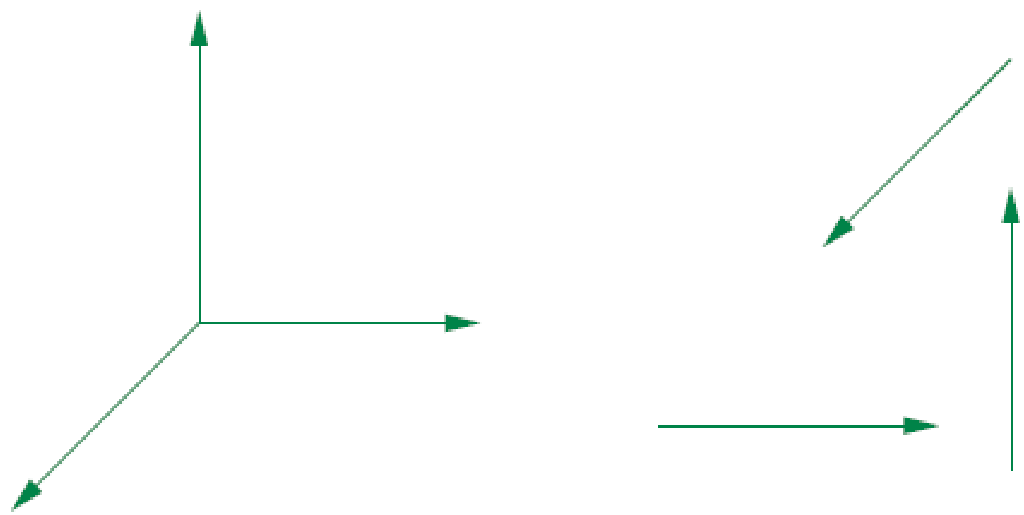
Then we can rewrite

$$w = \boldsymbol{a}^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \boldsymbol{a}^T \boldsymbol{v}$$

We usually think of the basis vector v_1, v_2, and v_3 as defining a coordinate system.

Coordinate Systems and Frames

- Besides basis vectors, we also need an origin. The origin and the basis vectors determine a ***frame***.



(a) (b)

Coordinate systems. (a) Vectors emerging from a common point. (b) Vectors moved.

Coordinate Systems and Frames

- Within a given frame, every vector can be written uniquely as $w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = \boldsymbol{a}^T \boldsymbol{v}$
- Every point can be written uniquely as

$$P = P_0 + \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3 = P_0 + \boldsymbol{b}^T \boldsymbol{v}$$

Representations and N-Tuples

- Suppose vectors e_1, e_2, and e_3 form a basis. The representation of any vector, *v*, is given by the component $(\alpha_1, \alpha_2$, and $\alpha_3)$ of a vector **a** where v$= \alpha_1 e_1 + \alpha_2 e_2$+$\alpha_3 e_3$.
- We can denote e_1, e_2, and e_3 by

$$e_1 = (1,0,0)^T$$
$$e_2 = (0,1,0)^T$$
$$e_3 = (0,0,1)^T$$

Representations and N-Tuples

- We can write the representation of any vector *v* as a column vector **a** or the 3-tuple $(\alpha_1, \alpha_2, \alpha_3)$.

Change of Coordinate Systems

- How the representation of a vector changes when we change the basis vectors?

model frame -> world frame -> camera or eye frame

- The conversion from the object frame to the eye frame is done by the model-view matrix.

Change of Coordinate Systems

- Suppose that $\{v_1, v_2, v_3\}$ and $\{u_1, u_2, u_3\}$ are two bases. Each basis vector in the second set can be represented in terms of the first basis (and vice versa). Hence, there exist nine scalar components, $\{\gamma_{ij}\}$, such that

$$u_1 = \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3$$
$$u_2 = \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3$$
$$u_3 = \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3$$

Change of Coordinate Systems

- The 3×3 matrix

$$\boldsymbol{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}$$

is defined by these scalars, and

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \text{ or } \boldsymbol{u} = \boldsymbol{M}\boldsymbol{v}$$

Change of Coordinate Systems

- The matrix **M** contains the information to go from a representation of a vector in one basis to its representation in the second basis. The inverse **M** gives the matrix representation of the change from $\{u_1, u_2, u_3\}$ to $\{v_1, v_2, v_3\}$.

Change of Coordinate Systems

- Consider a vector *w* that has the representation $\{\alpha_1, \alpha_2, \alpha_3\}$ with respect to $\{v_1, v_2, v_3\}$; that is $w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = \boldsymbol{a}^T \boldsymbol{v}$.
- Assume **b** is the representation of *w* with respect to $\{u_1, u_2, u_3\}$; that is $w = \beta_1 u_1 + \beta_2 u_2 + \beta_3 u_3 = \boldsymbol{b}^T \boldsymbol{u}$.

Change of Coordinate Systems

- $\boldsymbol{a} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$ $\boldsymbol{b} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$ then we have

$$w = \boldsymbol{b}^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \boldsymbol{b}^T \boldsymbol{M} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \boldsymbol{a}^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

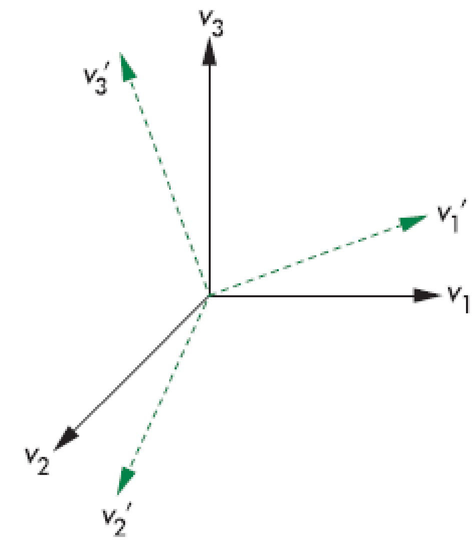
Thus, $\boldsymbol{a} = \boldsymbol{M}^T \boldsymbol{b}$

Change of Coordinate Systems

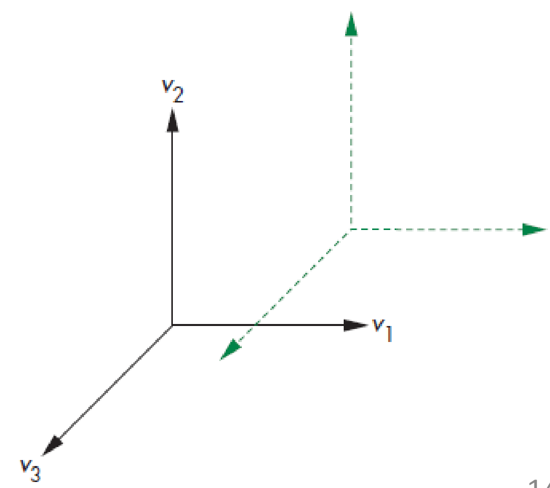
- The matrix $\boldsymbol{T} = (\boldsymbol{M}^T)^{-1}$ takes us from **a** to **b**, through the simple matrix equation

$$\boldsymbol{b} = \boldsymbol{Ta}$$

- The above changes in basis leave the origin unchanged. We can use them to represent rotation and scaling.



Rotation and scaling of a basis.



Translation of a basis.

Example: Change of Representation

- Suppose $w = v_1 + 2v_2$+$3v_3$, convert it to a new basis system $\{u_1, u_2, u_3\}$ where

$$u_1 = v_1$$

$$u_2 = v_1 + v_2$$

$$u_3 = v_1 + v_2 + v_3$$

Homogeneous Coordinates

- Using a 4-dimensional representation for both points and vectors in 3D.
- In the frame specified by (v_1, v_2, v_3, P_0), any point P can be written uniquely as

$$p = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + P_0$$

We can express this relation by a matrix product

$$P = [\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 1]\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

Homogeneous Coordinates

- We can represent P by $[\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 1]^T$

$$= \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix}$$

- In the same frame, any vector w can be written as

$$w = \delta_1 v_1 + \delta_2 v_2 + \delta_3 v_3 = [\delta_1 \quad \delta_2 \quad \delta_3 \quad 0] \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

Homogeneous Coordinates

- We can represent *w* by $[\delta_1 \quad \delta_2 \quad \delta_3 \quad 0]^T$

$$= \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ 0 \end{bmatrix}$$

Change of Frames

- If (v_1, v_2, v_3, P_0) and (u_1, u_2, u_3, Q_0) are two frames, then we can express the basis vectors and reference point of the second frame in terms of the first one as

$$u_1 = \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3$$
$$u_2 = \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3$$
$$u_3 = \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3$$
$$Q_0 = \gamma_{41}v_1 + \gamma_{42}v_2 + \gamma_{43}v_3 + P_0$$

Change of Frames

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

where now *M* is a 4×4 matrix

$$\boldsymbol{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{bmatrix}$$

M is called the matrix representation of the changes of frame.

Change of Frames

- We can use *M* to compute the changes in the representations directly.
- Suppose that **a** and **b** are homogeneous-coordinate representations for a point (or a vector) in two frames. Then

$$\boldsymbol{b}^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = \boldsymbol{b}^T \boldsymbol{M} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = \boldsymbol{a}^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

Change of Frames

- Hence, $\boldsymbol{a} = \boldsymbol{M}^T \boldsymbol{b}$
- Normally, we have more interest in $\boldsymbol{M}^T$

$$\boldsymbol{M}^T = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Why Homogeneous Coordinates?

- All affine transformation can be represented as matrix multiplication.
- We can have successive transformations by using a product matrix.

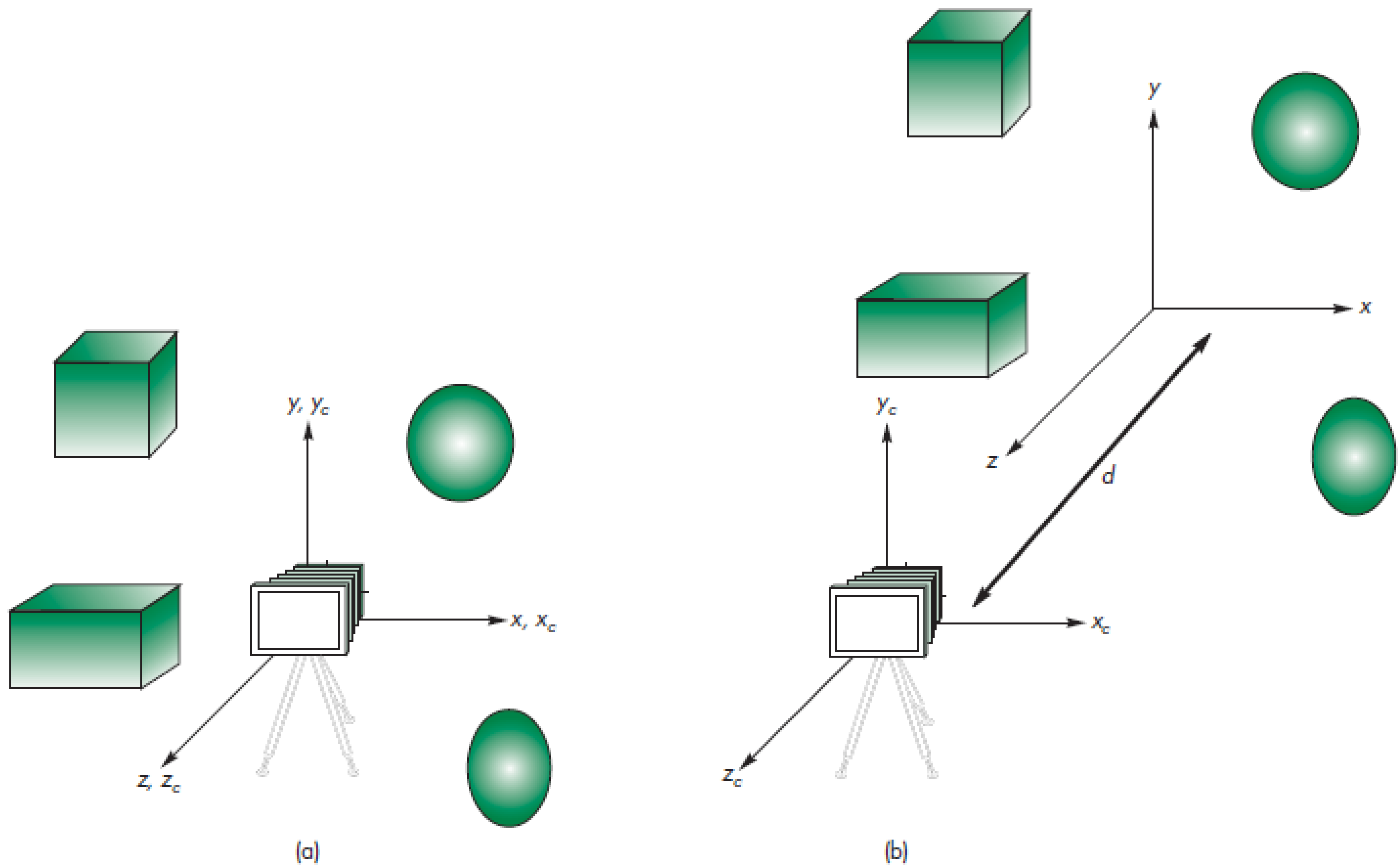
Example: Change of Frames

- Suppose $w = v_1 + 2v_2$+$3v_3$, convert it to a new basis system $\{u_1, u_2, u_3\}$ where

$$u_1 = v_1$$
$$u_2 = v_1 + v_2$$
$$u_3 = v_1 + v_2 + v_3$$

Frames in WebGL

- Traditionally, we have six frames in the pipeline. In each of these frames, a vertex has different coordinates. The following is the usual order in which the frames occur in the pipeline:
    1. Model coordinates
    2. Object (world) coordinates
    3. Eye (or camera) coordinates
    4. Clip coordinates
    5. Normalized device coordinates
    6. Window (or screen) coordinates



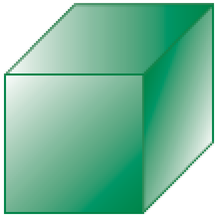
Camera and object frame. (a) In default positions.
(b) After applying model-view matrix.

Matrix and Vector Types

```
var a = vec3();            // create a vec3 with all components set to 0
var b = vec3(1, 2, 3);     // create a vec3 with the components 1, 2, 3
var c = vec3(b);           // copy the vec3 'b' by copying vec3 'c'

var d = mat3();            // create a mat3 identity matrix
var e = mat3(0, 1, 2,
             3, 4, 5,
             6, 7, 8);     // create a mat3 from 9 elements
var f = mat3(e);           // create the mat3 'f' by copying mat3 'e'
```

```
a = add(b,c);        // adds vectors 'b' and 'c' and puts result in 'a'
d = mat4();          // sets 'd' to an identity matrix
d = transpose(e);    // sets 'd' to the transpose of 'e'
f = mult(e, d);      // sets 'f' to the product of 'e' and 'd'
```

Modeling A Colored Cube



- Modeling the faces:

We regard a cube either as the intersection of six planes or as the six polygons, called facets, that define its faces.

We assume the vertices of the cube are available through an array *vertices*.

We can then use the list of points to specify the faces of the cube (e.g.,0,3,2,1).

```
var vertices = [
  vec3(-0.5, -0.5,  0.5),
  vec3(-0.5,  0.5,  0.5),
  vec3(0.5,  0.5,  0.5),
  vec3(0.5, -0.5,  0.5),
  vec3(-0.5, -0.5, -0.5),
  vec3(-0.5,  0.5, -0.5),
  vec3(0.5,  0.5, -0.5),
  vec3(0.5, -0.5, -0.5)
];
```

or

```
var vertices = [
  vec4(-0.5, -0.5,  0.5, 1.0),
  vec4(-0.5,  0.5,  0.5, 1.0),
  vec4(0.5,  0.5,  0.5, 1.0),
  vec4(0.5, -0.5,  0.5, 1.0),
  vec4(-0.5, -0.5, -0.5, 1.0),
  vec4(-0.5,  0.5, -0.5, 1.0),
  vec4(0.5,  0.5, -0.5, 1.0),
  vec4(0.5, -0.5, -0.5, 1.0)
];
```

Modeling A Colored Cube

- Inward- and outward-Pointing Faces

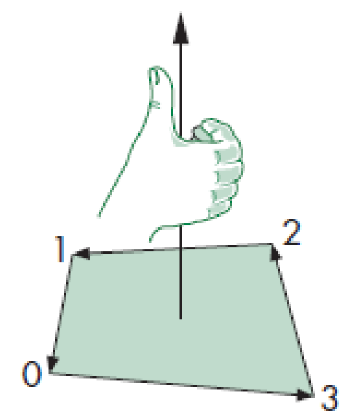
When we specify a polygon, the order of the vertices is important because each polygon has two sides.

We need a consistent way to distinguish between the two faces of a polygon.

Modeling A Colored Cube

- Inward- and outward-Pointing Faces

We call a face outward facing if the vertices are traversed in a counter-clockwise order when the face is viewed from the outside.



Traversal of the edges of a polygon.

Modeling A Colored Cube

- Data Structures for Object Representation
  - We can define 6 faces

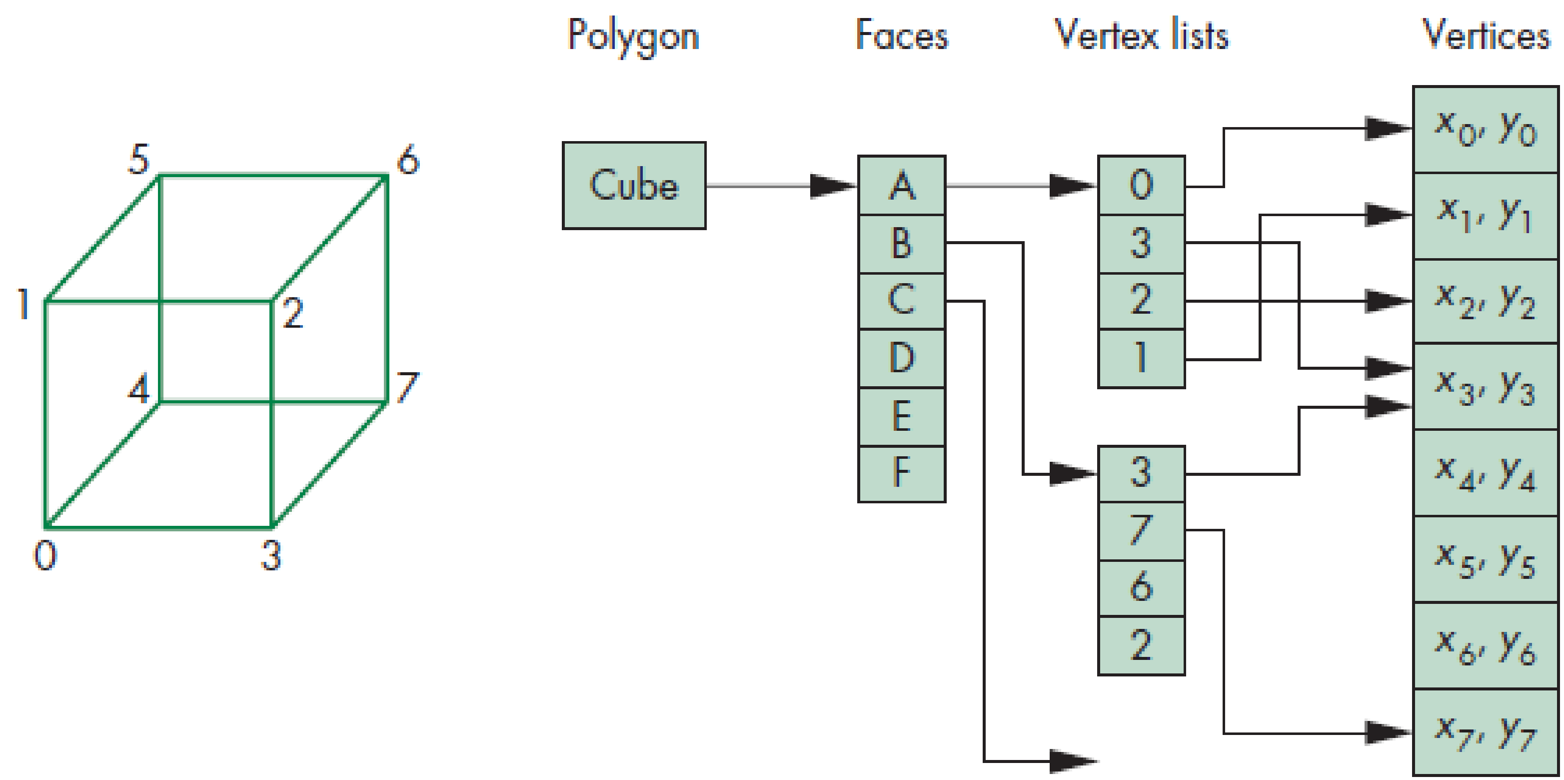
```
var faces = new Array(6);

for (var i = 0; i < faces.length; ++i) {
  faces[i] = new Array(4);
}
```

  - Or define 6×4 vertices

```
var faces = new Array(24);
```

Modeling A Colored Cube

- Data Structures for Object Representation



Vertex-list representation of a cube.

Modeling A Colored Cube

- Data Structures for Object Representation
  - One of the advantages of this structure is that each geometric location appears only once, instead of being repeated each time it is used for a face.

Modeling A Colored Cube

- The Colored Cube

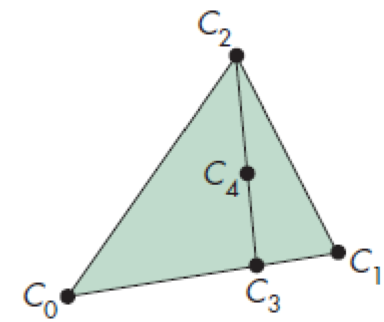
```
function colorCube()
{
  quad(1, 0, 3, 2);
  quad(2, 3, 7, 6);
  quad(3, 0, 4, 7);
  quad(6, 5, 1, 2);
  quad(4, 5, 6, 7);
  quad(5, 4, 0, 1);
}
```

```
var vertexColors = [
  [ 0.0, 0.0, 0.0, 1.0 ],  // black
  [ 1.0, 0.0, 0.0, 1.0 ],  // red
  [ 1.0, 1.0, 0.0, 1.0 ],  // yellow
  [ 0.0, 1.0, 0.0, 1.0 ],  // green
  [ 0.0, 0.0, 1.0, 1.0 ],  // blue
  [ 1.0, 0.0, 1.0, 1.0 ],  // magenta
  [ 1.0, 1.0, 1.0, 1.0 ],  // white
  [ 0.0, 1.0, 1.0, 1.0 ]   // cyan
];
```

```
function quad(a, b, c, d)
{
  var indices = [ a, b, c, a, c, d ];

  for (var i = 0; i < indices.length; ++i) {
    points.push(vertices[indices[i]]);
    colors.push(vertexColors[indices[i]]);
  }
}
```

Modeling A Colored Cube

- Color Interpolation
  - Assign colors to points inside
  - Based on barycentric coordinate representation of triangles



Interpolation using barycentric coordinates.

  - C_0, C_1, and C_2 are colors of the three vertices
  - $C_{01}(\alpha)$ are colors along the edge between vertices 0 and 1 $\quad C_{01}(\alpha) = (1-\alpha)\, C_0 + \alpha C_1 \quad 0 \le \alpha \le 1$
  - For a given α, we obtain color C_3. We can generate colors for the line connecting C_2 and C_3.

Modeling A Colored Cube

- Color Interpolation
  - $C_{32}(\beta) = (1 - \beta)\, C_3 + \beta C_2 \quad 0 \leq \beta \leq 1$
  - For a given β, we obtain color C_4.
  - As the barycentric coordinates α and β range from 0 and 1, we can get interpolated colors for all the interior points and thus a color for each fragment generated by the rasterizer.

Modeling A Colored Cube

- Displaying the Cube

Scale the data to get a smaller cube

Vertex shader (homogeneous coord.)

```
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;

void main()
{
  fColor = vColor;
  gl_Position = 0.5 * vPosition;
}
```

Fragment shader

```
varying vec4 fColor;

void main()
{
  gl_FragColor = fColor;
}
```

Modeling A Colored Cube

- Drawing with Elements
  - 12 triangles

```
var indices = [
  1, 0, 3,
  3, 2, 1,
  2, 3, 7,
  7, 6, 2,
  3, 0, 4,
  4, 7, 3,
  6, 5, 1,
  1, 2, 6,
  4, 5, 6,
  6, 7, 4,
  5, 4, 0,
  0, 1, 5
];
```

```
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertexColors), gl.STATIC_DRAW);

var iBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, iBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(indices),
              gl.STATIC_DRAW);

gl.drawElements(gl.TRIANGLES, numVertices, gl.UNSIGNED_BYTE, 0);
```