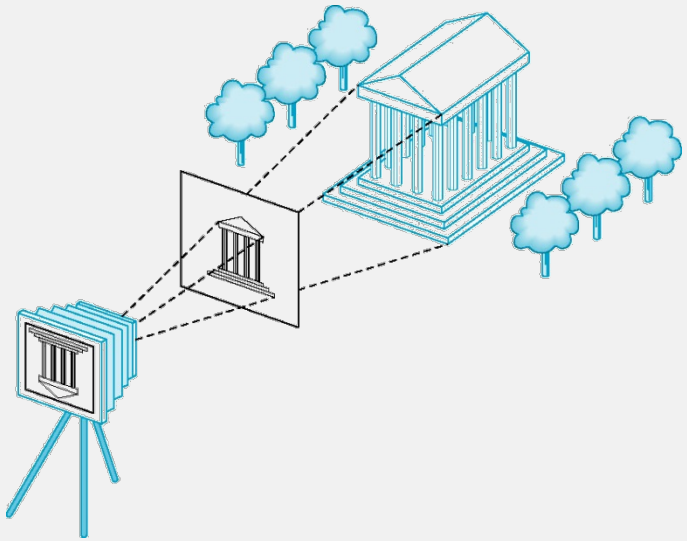
컴퓨터그래픽스

김준호
Visual Computing Lab.
국민대학교 소프트웨어학부

Synthetic Objects

Elements of Image Formation

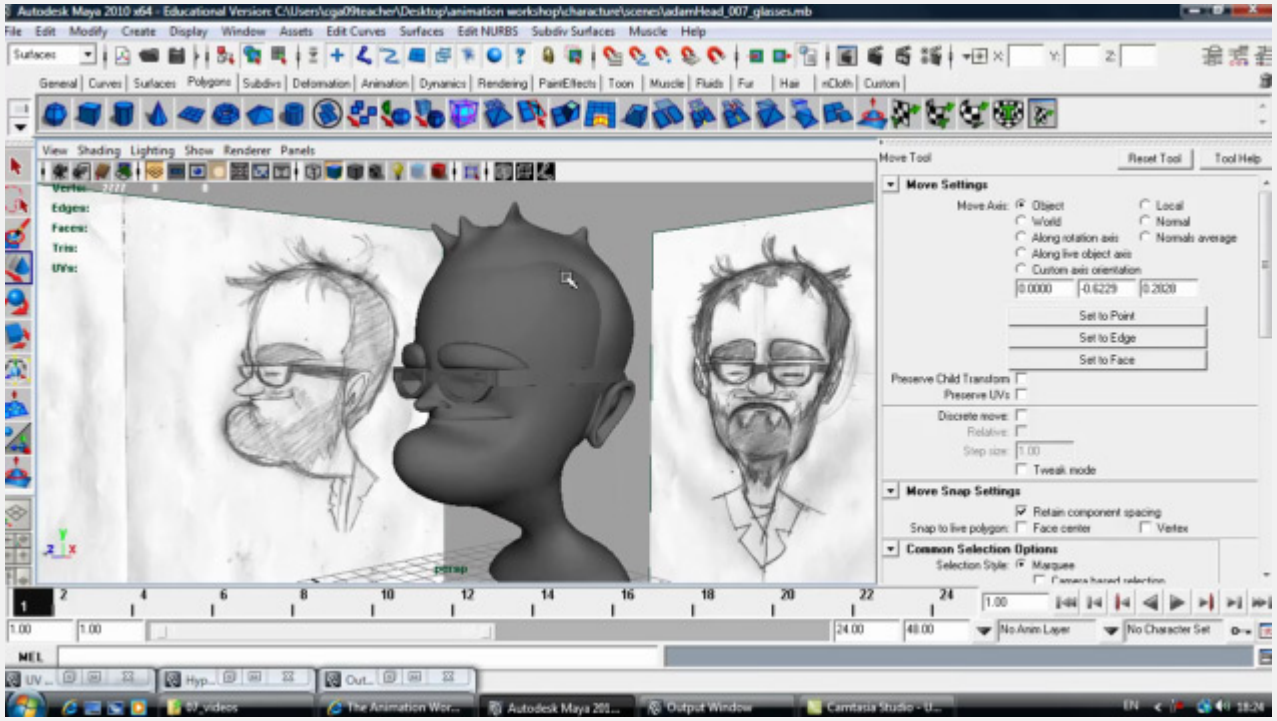
- Viewer (or camera)
  - Synthetic camera
- Objects
  - Synthetic objects
- Light source(s)
  - Synthetic lights
- Attributes
  - Material, surface normal for reflection model (i.e., light-material interaction)



Synthetic image formation
in Computer Graphics

Modeling of Synthetic Object

- 3D artists generate the modeling data of synthetic objects
  - 3D modeling tools: Maya, 3D studio Max, etc.

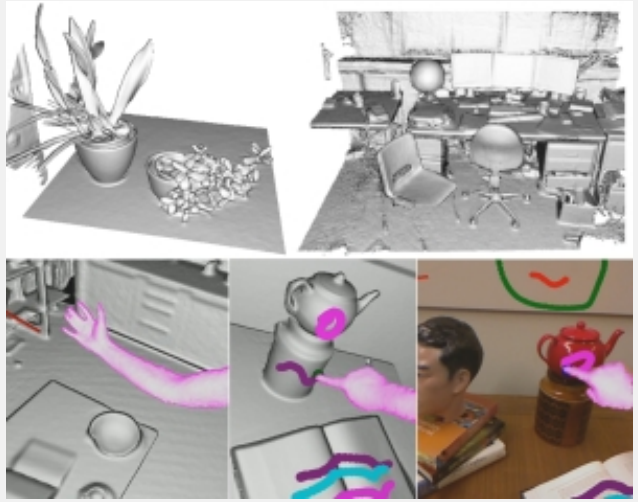


http://3dexport.com/3dtuts/3d-tutorials/facial-modelling-in-maya-tutorial-part-7-of-8/

Modeling of Synthetic Object

- 3D scanners capture the modeling data of real-world objects

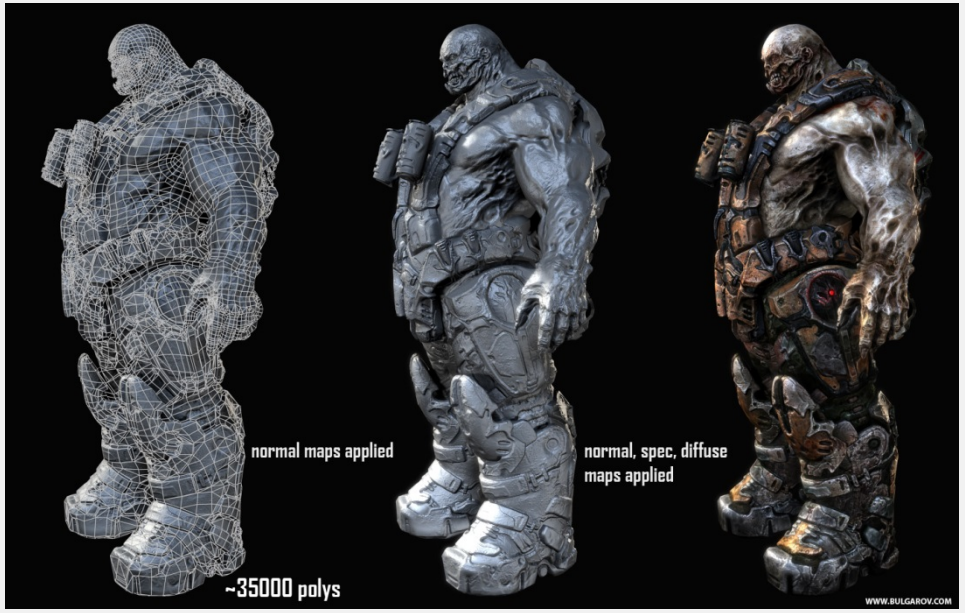
http://news.thomasnet.com/fullstory/3D-Scanners-capture-images-at-rate-of-15-surfaces-sec-828949



[KinectFusion 2011]

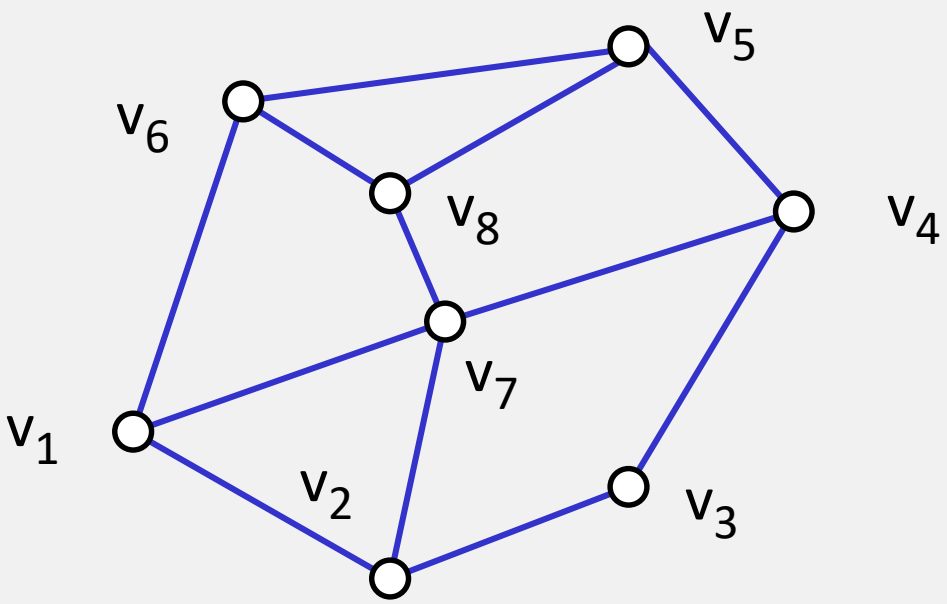
Modeling of Synthetic Object

- Data for synthetic object
  - 3D model
    - Vertices: 3D position, normal, color, texture coord., for each vertex
    - Faces: polygon-vertex indices, for each face
  - Texture image



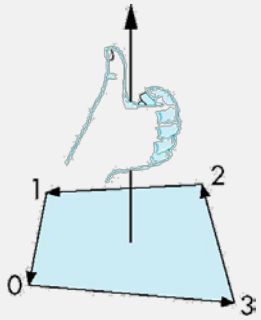
Simple Example

- Simple polygon model
  - 8 vertices, 5 faces

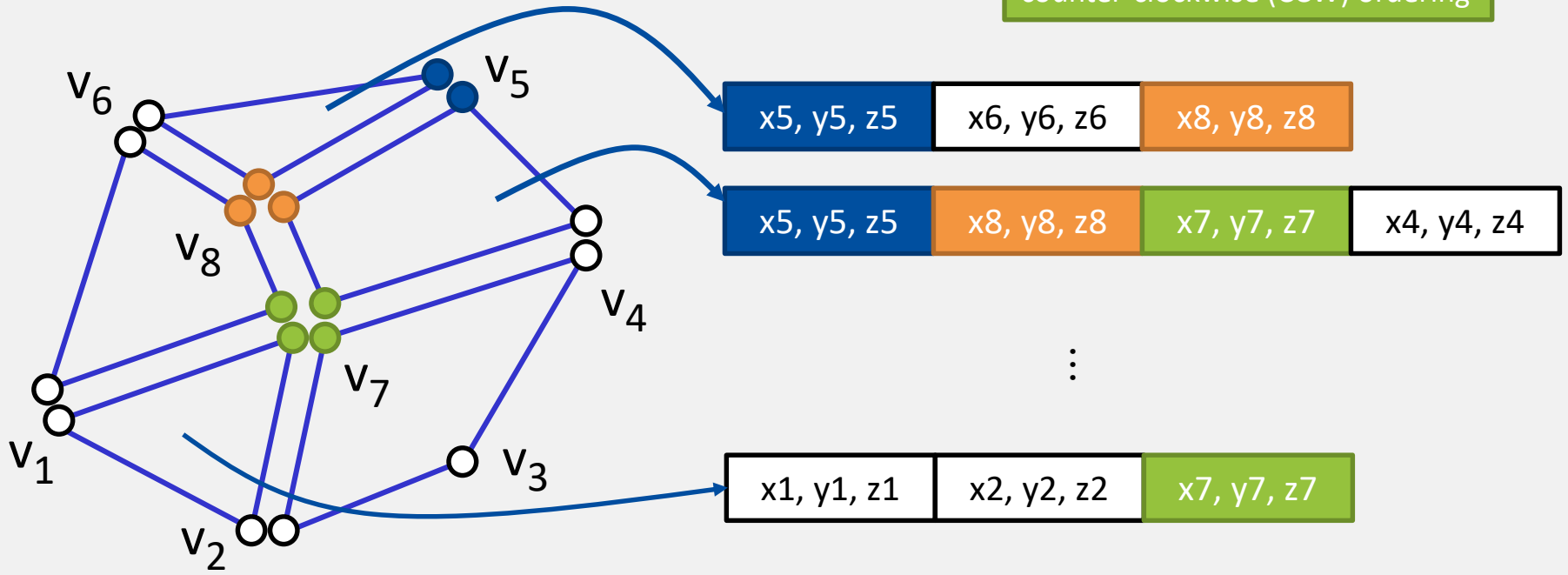


Simple Example – Polygon Soup

- Simple polygon model
  - 8 vertices, 5 faces
- Polygon soup
  - Duplication of vertex information

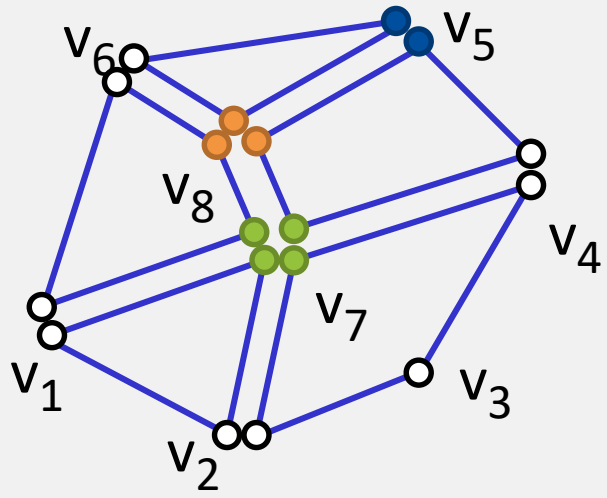


Outward facing:
counter-clockwise (CCW) ordering



Simple Example – Polygon Soup

- Polygon data transmission
  - 2 triangles
  - 3 quads



Vertex Attribute: 3D Position
Primitive type: TRIANGLES
# of primitives: 2

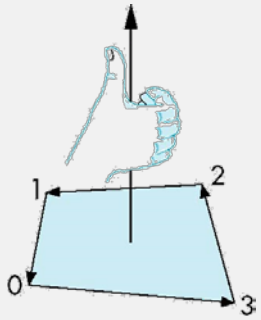
x5, y5, z5	x6, y6, z6	x8, y8, z8	x1, y1, z1	x2, y2, z2	x7, y7, z7

Vertex Attribute: 3D Position
Primitive type: QUADS
# of primitives: 3

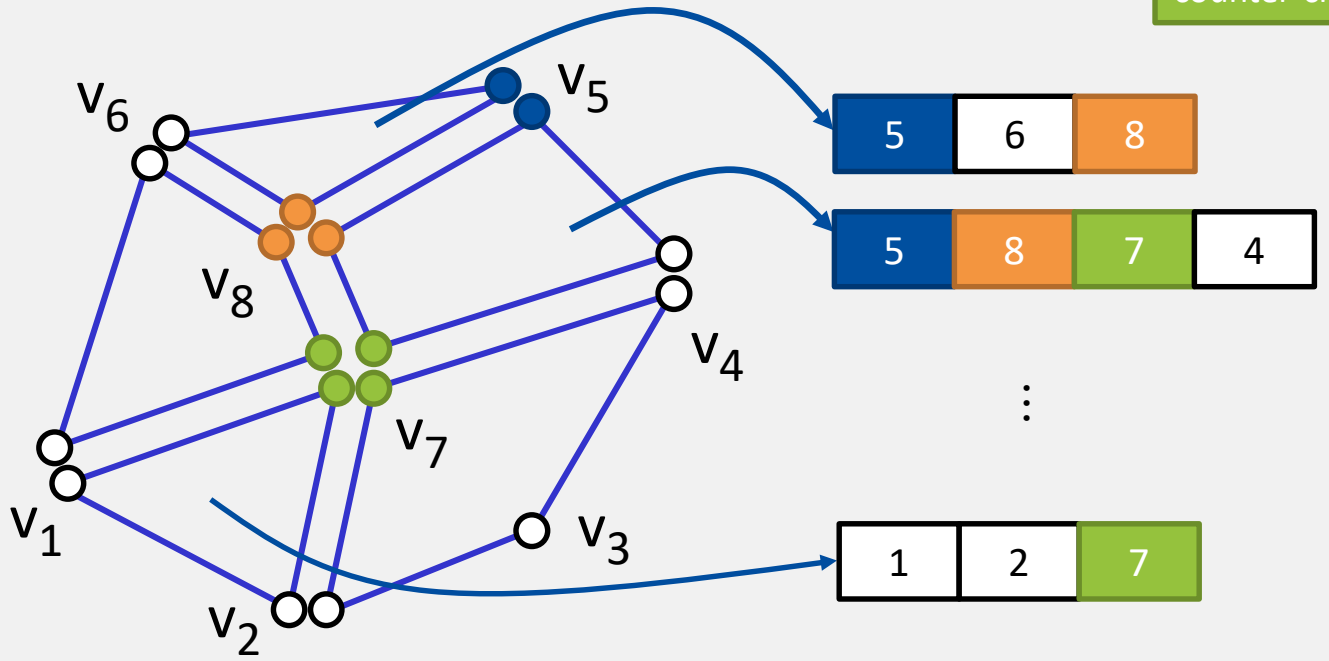
x5, y5, z5	x8, y8, z8	x7, y7, z7	x4, y4, z4	x7, y7, z7	x2, y2, z2
x3, y3, z3	x4, y4, z4	x7, y7, z7	x8, y8, z8	x6, y6, z6	x1, y1, z1

Simple Example – Vertex List & Polygons

- Simple polygon model
  - 8 vertices, 5 faces
- Vertex list & polygons
  - Duplication of vertex indices



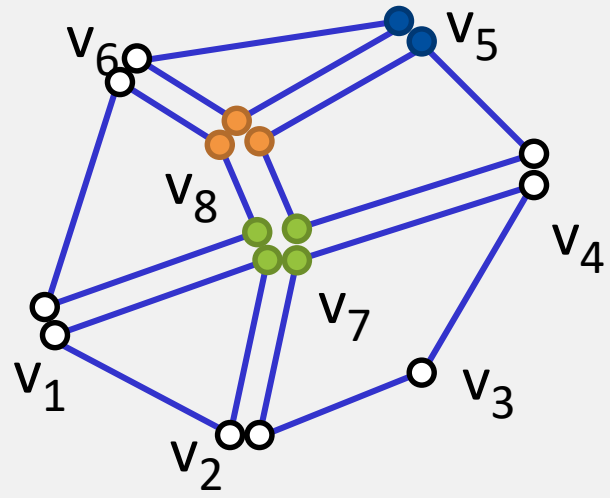
Outward facing:
counter-clockwise (CCW) ordering



$x_1\ y_1\ z_1$
$x_2\ y_2\ z_2$
$x_3\ y_3\ z_3$
$x_4\ y_4\ z_4$
$x_5\ y_5\ z_5.$
$x_6\ y_6\ z_6$
$x_7\ y_7\ z_7$
$x_8\ y_8\ z_8$

Simple Example – Vertex List & Polygons

- Polygon data transmission
  - 2 triangles
  - 3 quads



Vertex Attribute: 3D Position

x1, y1, z1	x2, y2, z2	x3, y3, z3	x4, y4, z4	x5, y5, z5	x6, y6, z6
x7, y7, z7	x8, y8, z8				

Polygon-Vertex Indices
Primitive type: TRIANGLES
# of Primitives: 2

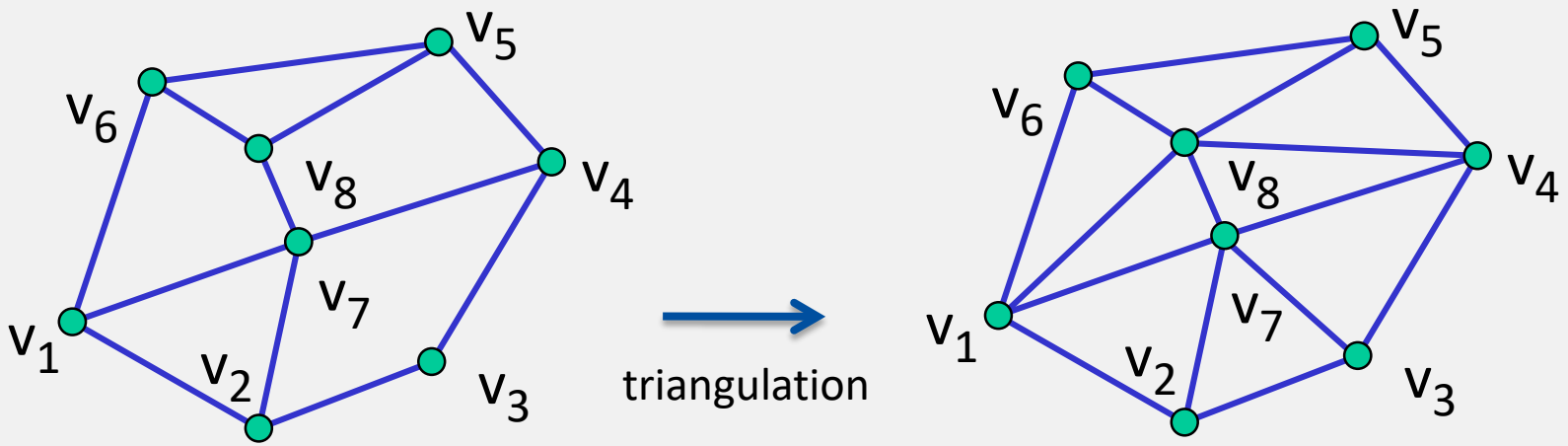
5	6	8	1	2	7

Polygon-Vertex indices
Primitive type: QUADS
# of Primitive: 3

5	8	7	4	7	2
3	4	7	8	6	1

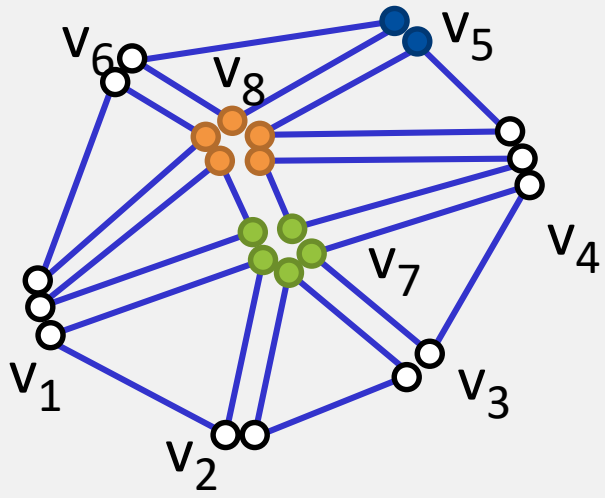
Triangle Meshes

- Triangle mesh: every polygon primitive is a triangle
- OpenGL v.s. OpenGL ES
  - OpenGL supports GL_TRIANGLES, GL_QUADS, GL_POYLGON for polygon primitives
  - OpenGL ES supports GL_TRIANGLES, ~~GL_QUADS~~, ~~GL_POYLGON~~ for polygon primitives
- Benefit?



Example of Triangle Mesh – Polygon Soup

- Triangle data transmission
  - 8 triangles
- Advantage
  - Simple data structure & simple function I/O

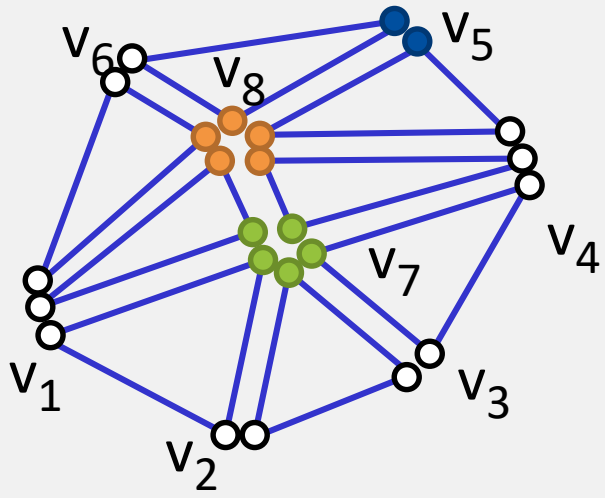


Vertex Attribute: 3D Position
Primitive type: TRIANGLES
# of Primitive: 8

x5, y5, z5	x6, y6, z6	x8, y8, z8	x6, y6, z6	x1, y1, z1	x8, y8, z8
x1, y1, z1	x7, y7, z7	x8, y8, z8	x7, y7, z7	x4, y4, z4	x8, y8, z8
x4, y4, z4	x5, y5, z5	x8, y8, z8	x1, y1, z1	x2, y2, z2	x7, y7, z7
x2, y2, z2	x3, y3, z3	x7, y7, z7	x3, y3, z3	x4, y4, z4	x7, y7, z7

Example of Triangle Mesh – Vertex List & Polygons

- Triangle data transmission
  - 8 triangles
- Advantage
  - Simple data structure & simple function I/O



Vertex Attribute: 3D Position

x1, y1, z1	x2, y2, z2	x3, y3, z3	x4, y4, z4	x5, y5, z5	x6, y6, z6
x7, y7, z7	x8, y8, z8				

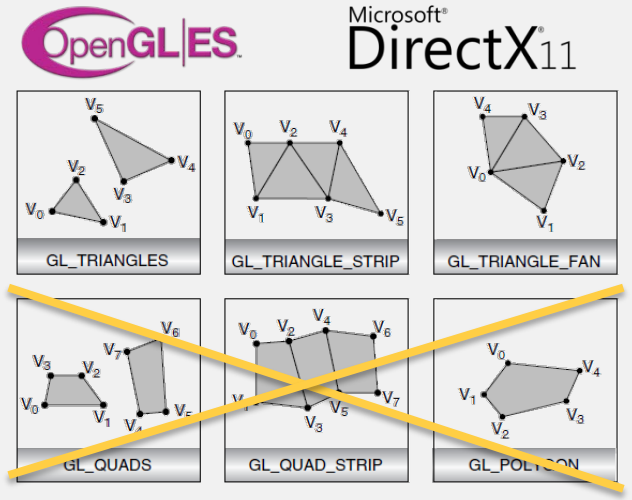
Polygon-Vertex Indices
Primitive type: TRIANGLES
# of Primitives: 2

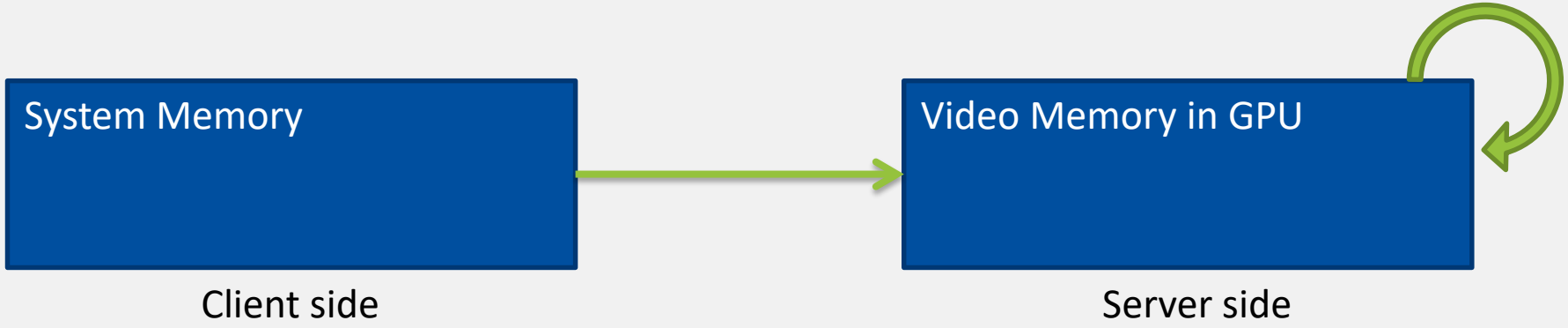
5	6	8	6	1	8	1	7	8	7	4	8
4	5	8	1	2	7	2	3	7	3	4	7

More Advantages of Triangle Meshes

- We can utilize block-based transmission
  - High speed
  - Suitable to embedded systems

- OpenGL ES, DirectX support triangle meshes only





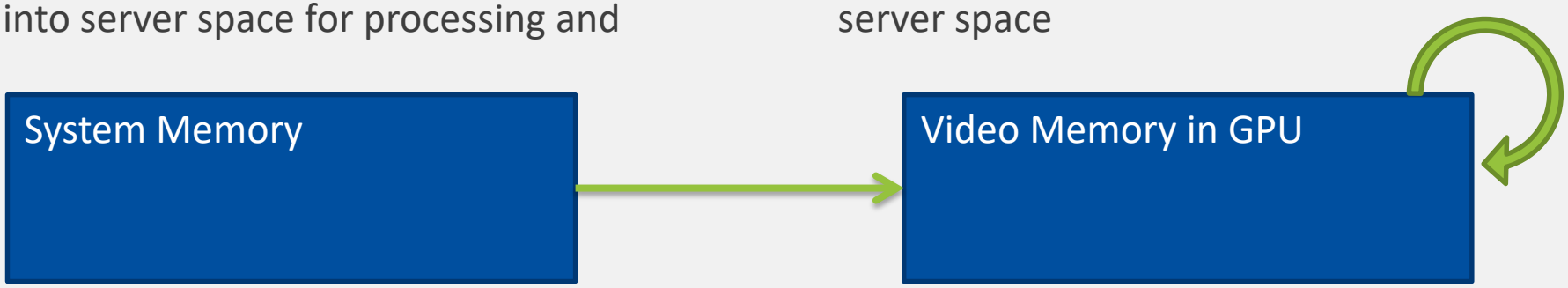
Modern OpenGL Rendering Architectures

Vertex Arrays

- OpenGL transfers vertex data using the client space array pointers into server space for processing and rendering

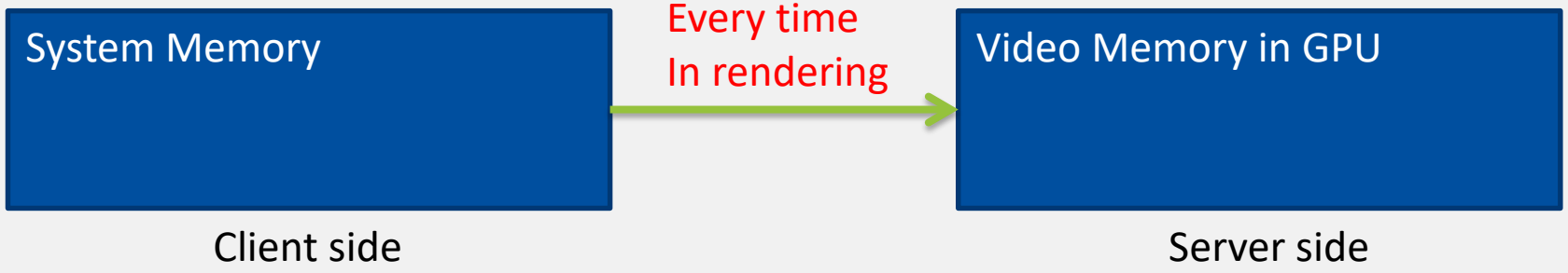
Vertex Buffer Objects (VBOs)

- Vertex buffer objects allow storing of vertex arrays in server space

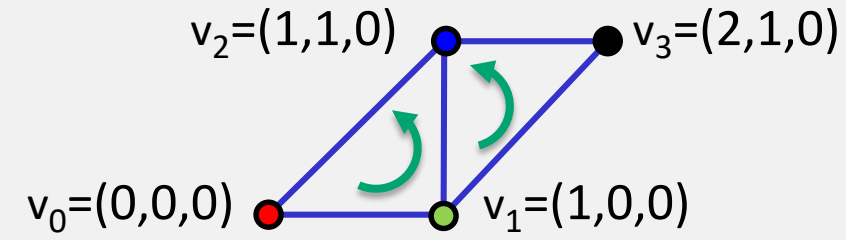


Modern OpenGL codes – Vertex Arrays

- Vertex arrays are stored in client space
  - Still need to transfer vertex data into server space, repeatedly
- Steps to use Vertex Arrays
  1. Enable Arrays
     - glEnableVertexAttribArray()
  2. Specify Data
     - glVertexAttribPointer()
  3. Render with glDrawArrays() or glDrawElements()



Modern OpenGL codes – Vertex Arrays



Polygon Soup

1. Enable Arrays
2. Specify Data (polygon soup)
3. Render with glDrawArrays()

```
GLfloat position[] = {0,0,0,    1,0,0,    1,1,0,   1,0,0,   2,1,0,   1,1,0};
GLfloat color[]    = {1,0,0,1,  0,1,0,1,  0,0,1,1, 0,1,0,1, 0,0,0,1, 0,0,1,1};

// ...
GLint loc_a_position = glGetAttribLocation(program, "a_position");
GLint loc_a_color    = glGetAttribLocation(program, "a_color");

// ...
glEnableVertexAttribArray (loc_a_position);
glVertexAttribPointer(loc_a_position, 3, GL_FLOAT, GL_FALSE, 0, position);
glEnableVertexAttribArray (loc_a_color);
glVertexAttribPointer(loc_a_color, 4, GL_FLOAT, GL_FALSE, 0, color);

glDrawArrays(GL_TRIANGLES, 0, 6);

glDiableVertexAttribArray(loc_a_position);
glDiableVertexAttribArray(loc_a_color);
```

Vertex List & Polygons

1. Enable Array
2. Specify Data (vertex list & polygons)
3. Render with glDrawElements()

```
GLfloat position[] = {0,0,0,    1,0,0,    1,1,0,   1,0,0,   2,1,0,   1,1,0};
GLfloat color[]    = {1,0,0,1,  0,1,0,1,  0,0,1,1, 0,1,0,1, 0,0,0,1, 0,0,1,1};
GLubyte indices[]  = {0, 1, 2, 1, 3, 2};

// ...
GLint loc_a_position = glGetAttribLocation(program, "a_position");
GLint loc_a_color    = glGetAttribLocation(program, "a_color");

glEnableVertexAttribArray (loc_a_position);
glVertexAttribPointer(loc_a_position, 3, GL_FLOAT, GL_FALSE, 0, position);
glEnableVertexAttribArray (loc_a_color);
glVertexAttribPointer(loc_a_color, 4, GL_FLOAT, GL_FALSE, 0, color);

glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, indices);

glDiableVertexAttribArray(loc_a_position);
glDiableVertexAttribArray(loc_a_color);
```

Modern OpenGL codes – Vertex Arrays

- glEnableVertexAttribArray() / glDisableVertexAttribArray()
  - Enable or disable client-side capability of the arrays, with each storing a different type of data

```
// Enable or disable client-side capability of the arrays

void glEnableVertexAttribArray(GLuint index);
void glDisableVertexAttribArray(GLuint index);


// The parameter index specifies the index of generic vertex attributes to be enabled or disabled\

// If enabled, the values in the generic vertex attribute array will be accessed and used for
// rendering when calls are made to vertex array commands such as glDrawArrays(), or glDrawElements()
```

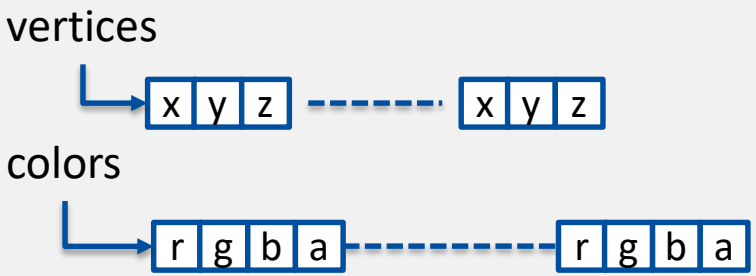
Modern OpenGL codes – Vertex Arrays

- glVertexAttribPointer()
  - Define an array of generic vertex attribute data

```
// Define an array of generic vertex attribute data
void glVertexAttribPointer(GLuint index, GLint size, GLenum type, GLboolean normalized, GLsizei stride, const GLvoid* pointer);

// index:            It specifies the index of the generic vertex attribute to be modified.
//
// size:             Must be 1, 2, 3, or 4.
//                   It specifies the number of components per generic vertex attribute
//
// type:             GL_FLOAT, GL_BYTE, GL_SHORT, GL_FIXED.
//                   It specifies the data type of each component in the array
//
// normalized:       GL_TRUE, when fixed-point data should be normalized
//                   GL_FALSE, when they can be accessed directly as fixed-point values
//
// stride:           0, in general.
//                   It specifies the byte offset between data for vertex index I and vertex index (I+1)
//
// pointer:          It specifies an offset of the first component of the first generic vertex attributes
```

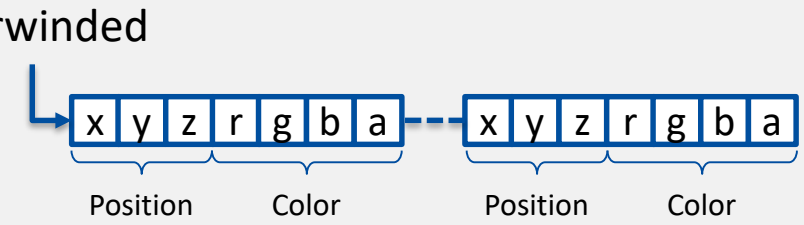
Modern OpenGL codes – Vertex Arrays

Separate Arrays




```
GLfloat position[] = {0,0,0,    1,0,0,    1,1,0,   1,0,0,   2,1,0,   1,1,0};
GLfloat color[]    = {1,0,0,1,  0,1,0,1,  0,0,1,1, 0,1,0,1, 0,0,0,1, 0,0,1,1};

// ...
GLint loc_a_position = glGetAttribLocation(program, "a_position");
GLint loc_a_color    = glGetAttribLocation(program, "a_color");

glEnableVertexAttribArray (loc_a_position);
glVertexAttribPointer(loc_a_position, 3, GL_FLOAT, GL_FALSE, 0, position);
glEnableVertexAttribArray (loc_a_color);
glVertexAttribPointer(loc_a_color, 4, GL_FLOAT, GL_FALSE, 0, color);

glDrawArrays(...);

glDiableVertexAttribArray(loc_a_position);
glDiableVertexAttribArray(loc_a_color);
```

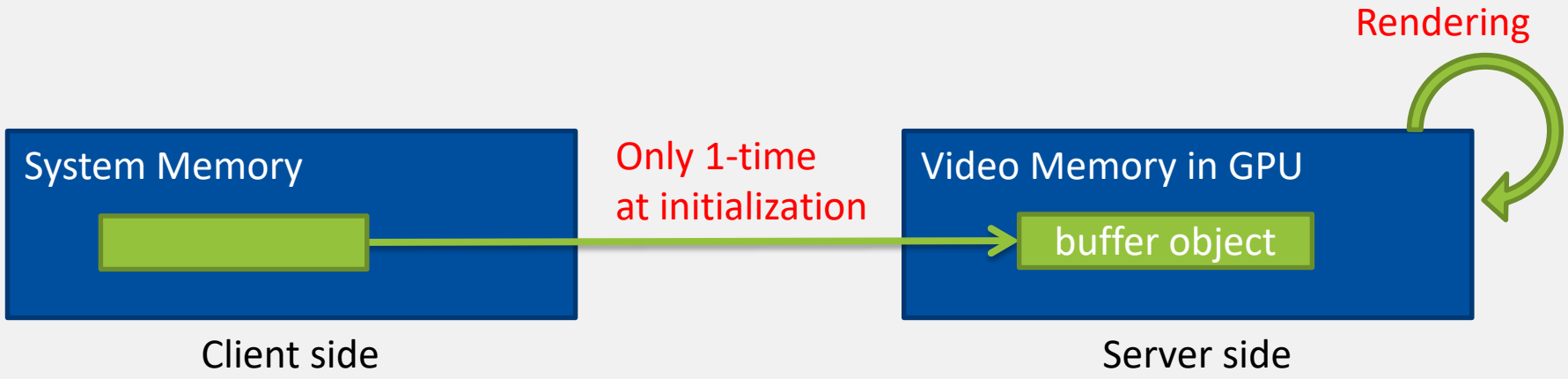
Interleaved Arrays



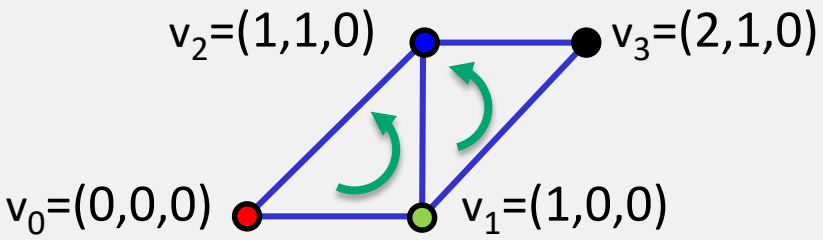

```
GLfloat interwinded[] = {0,0,0, 1,0,0,1,   1,0,0, 0,1,0,1,   1,1,0, 0,0,1,1,
                         1,0,0, 0,1,0,1,   2,1,0, 0,0,0,1,   1,1,0, 0,0,1,1};

// ...
GLint loc_a_interwinded = glGetAttribLocation(program, "a_interwinded");


glEnableVertexAttribArray(loc_a_interwinded);
glVertexAttribPointer(loc_a_interwinded, 3, GL_FLOAT, GL_FALSE,
                      7*sizeof(float), &interwinded[0]);
glVertexAttribPointer(loc_a_interwinded, 4, GL_FLOAT, GL_FALSE,
                      7*sizeof(float), &interwinded[3]);

glDrawArrays(…);

glDiableVertexAttribArray(loc_a_interwinded);
```

Modern OpenGL codes – Vertex Buffer Objects (VBOs)

- Vertex Buffer Objects (VBOs) allow storing of vertex arrays in *server* space
- Steps to use VBOs
  1. Generate buffer object identifiers
  2. Bind a buffer object, specifying for vertex data or indices
  3. Request storage, optionally initialize
  4. Specify data including offsets into buffer object
  5. Bind buffer object to be used in rendering
  6. Render using vertex array techniques (e.g., glDrawElements)



Modern OpenGL codes – Vertex Buffer Objects (VBOs)

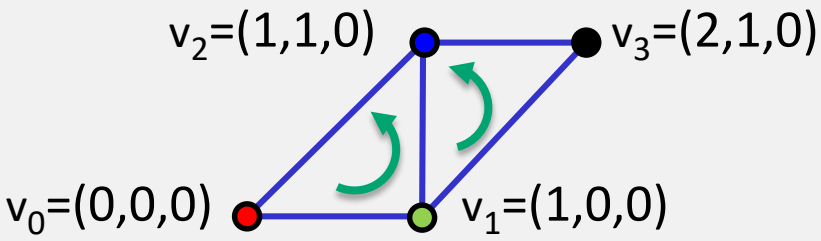


- Initialization
  1. Generate buffer object identifiers
  2. Bind a buffer object, specifying for vertex data or indices
  3. Request storage, optionally initialize
  4. Specify data including offsets into buffer object VBOs generation
  5. Bind buffer object to be used in rendering
  6. Render using vertex array techniques (e.g., glDrawElements)

- Modern OpenGL Codes (C/C++)

```
// buffers in client space
GLfloat vertices[] = {1, 0, 0, 0, 1, 0, -1, 0, 0};
GLubyte indices[] = {0, 1, 2, 1, 3, 2};

// buffer IDs in server space
GLuint verticesBuffer;
GLuint indicesBuffer;

// create a vertex buffer & trasfer vertices data from client space to server space
glGenBuffers(1,  &verticesBuffer);
glBindBuffer(GL_ARRAY_BUFFER, verticesBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);

// create an index buffer & trasfer vertices data from client space to server space
glGenBuffers(1, &indicesBuffer);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indicesBuffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_DYNAMIC_DRAW);
glGenBuffers(GL_ELEMENT_ARRAY_BUFFER, 0);
```

Modern OpenGL codes – Vertex Buffer Objects (VBOs)



- Rendering
  1. Generate buffer object identifiers
  2. Bind a buffer object, specifying for vertex data or indices
  3. Request storage, optionally initialize
  4. Specify data including offsets into buffer object VBOs generation
  5. Bind buffer object to be used in rendering
  6. Render using vertex array techniques (e.g., glDrawElements)

- Modern OpenGL Codes (C/C++)

```
// buffers in client space
// GLfloat vertices[] = {1, 0, 0, 0, 1, 0, -1, 0, 0};
// GLubyte indices[] = {0, 1, 2, 1, 3, 2};

// buffer IDs in server space
// GLuint verticesBuffer;
// GLuint indicesBuffer;


// specifying vertex data
glBindBuffer(GL_ARRAY_BUFFER, verticesBuffer);
glVertexPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));

// specifying index data
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indicesBuffer);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);

// reset buffers
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

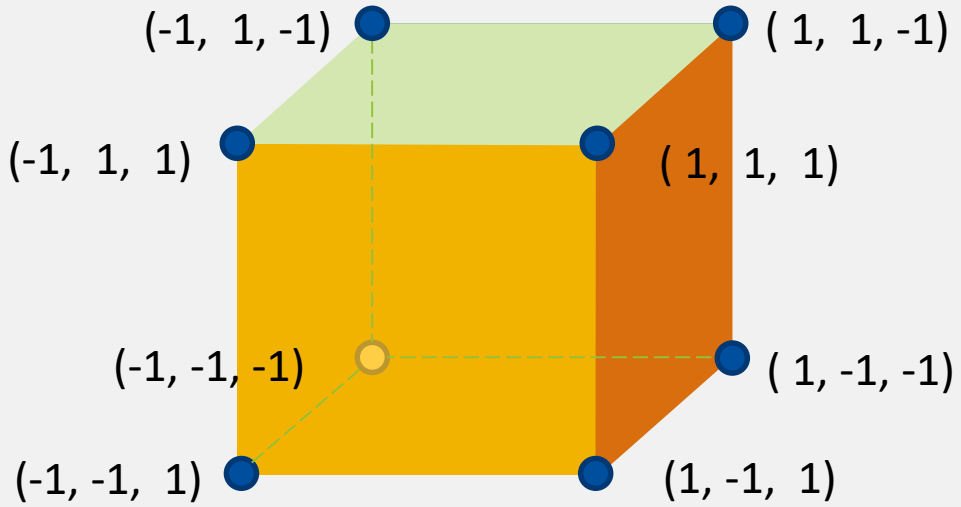
- Modeling a cube
  - Use Vertex Arrays
    - Use DrawArrays()
    - Use DrawElements()
  - Use Vertex Buffer Objects
- Quiz

Programming Practice

Synthetic Objects

Programming Practice

- Modeling a cube
  - The six rectangles should have different colors
  - Use glDrawArrays()
- Quiz
  - Use glDrawElements()



Programming Practice

- Modeling a cube
- Draw a cube using 3 different ways
  - glDrawArrays
  - glDrawElements
  - Vertex Buffer Objects (VBOs)