컴퓨터그래픽스

김준호

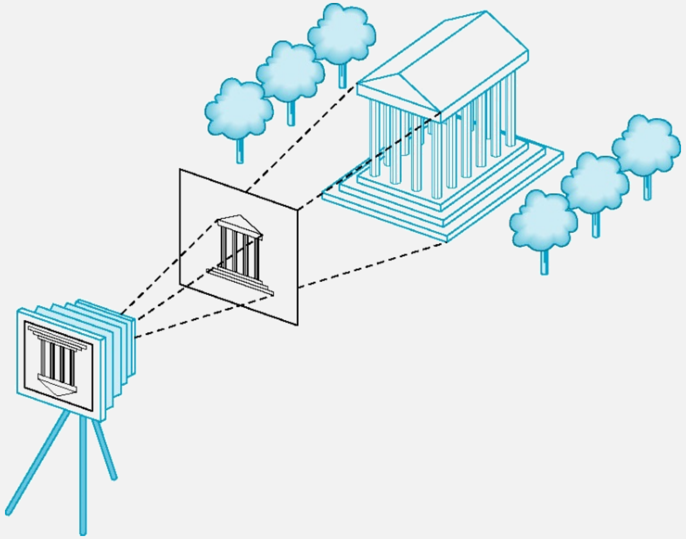
Visual Computing Lab.

국민대학교 소프트웨어학부

Synthetic Objects

Elements of Image Formation

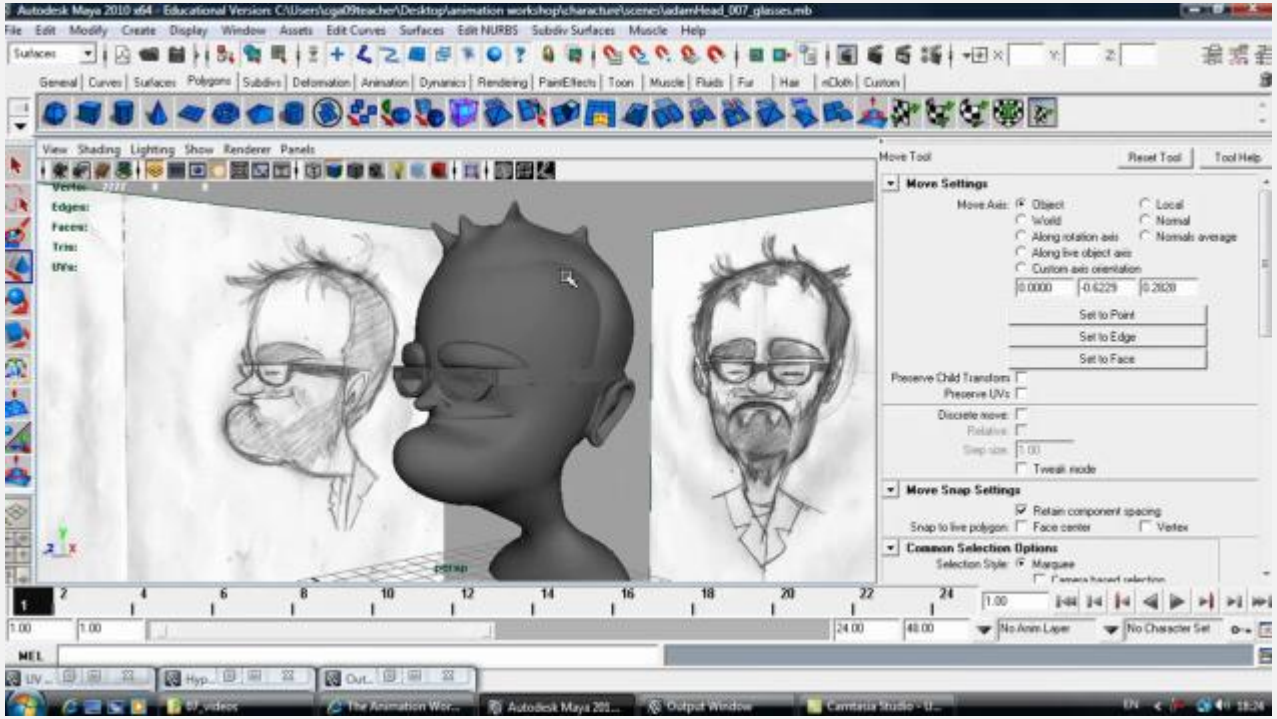
- Viewer (or camera)
  - Synthetic camera
- Objects
  - Synthetic objects
- Light source(s)
  - Synthetic lights
- Attributes
  - Material, surface normal
    for reflection model
    (i.e., light-material interaction)



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.



Autodesk Maya 2010 x64 - Educational Version: C:\Users\cga09teacher\Desktop\animation workshop\character\scenes\adamHead_007_glasses.mb
File Edit Modify Create Display Window Assets Edit Curves Surfaces Edit NURBS Subdiv Surfaces Muscle Help
Surfaces
General Curves Surfaces Polygons Subdivs Deformation Animation Dynamics Rendering PaintEffects Toon Muscle Fluids Fur Hair nCloth Custom
View Shading Lighting Show Renderer Panels
Verts:
Edges:
Faces:
Tris:
UVs:
persp
MEL

Move Tool Reset Tool Tool Help

▾ Move Settings

Move Axis: ● Object ○ Local
○ World ○ Normal
○ Along rotation axis ○ Normals average
○ Along live object axis
○ Custom axis orientation
0.0000 -0.6229 0.2828

Set to Point
Set to Edge
Set to Face

Preserve Child Transform □
Preserve UVs □

Discrete move: □
Relative: □
Step size: 1.00
□ Tweak mode

▾ Move Snap Settings

☑ Retain component spacing
Snap to live polygon: □ Face center □ Vertex

▾ Common Selection Options

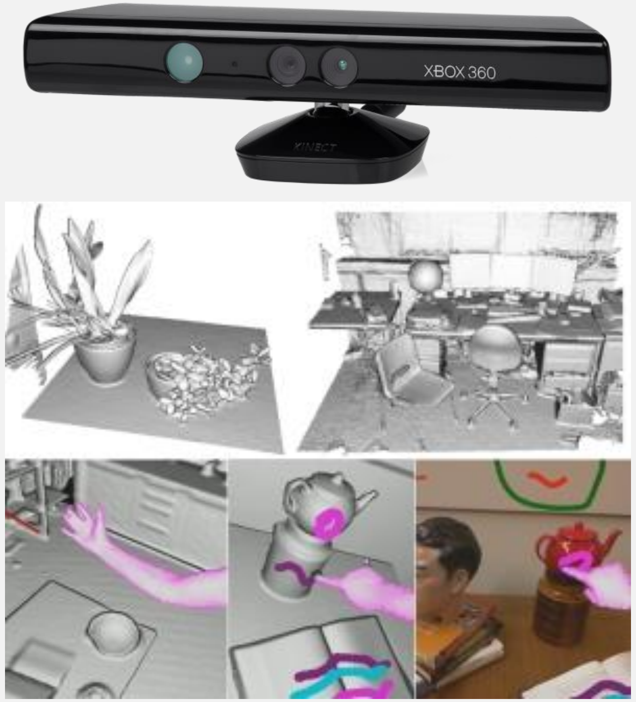
Selection Style: ● Marquee
□ Camera based selection

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

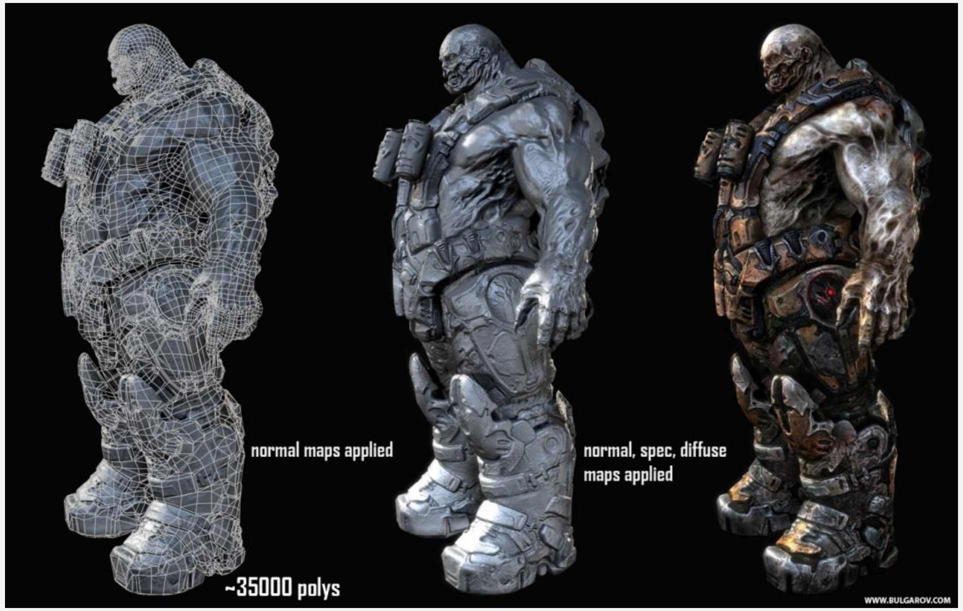
http://news.thomasnet.com/fullstory/3D-Scanners-capture-images-at-rate-of-15-surfaces-sec-828949

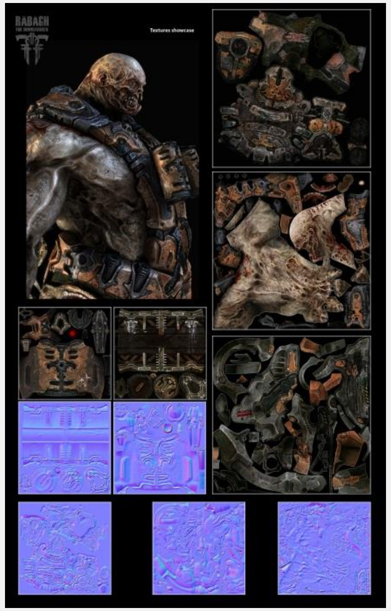


[KinectFusion 2011]

Modeling of Synthetic Object

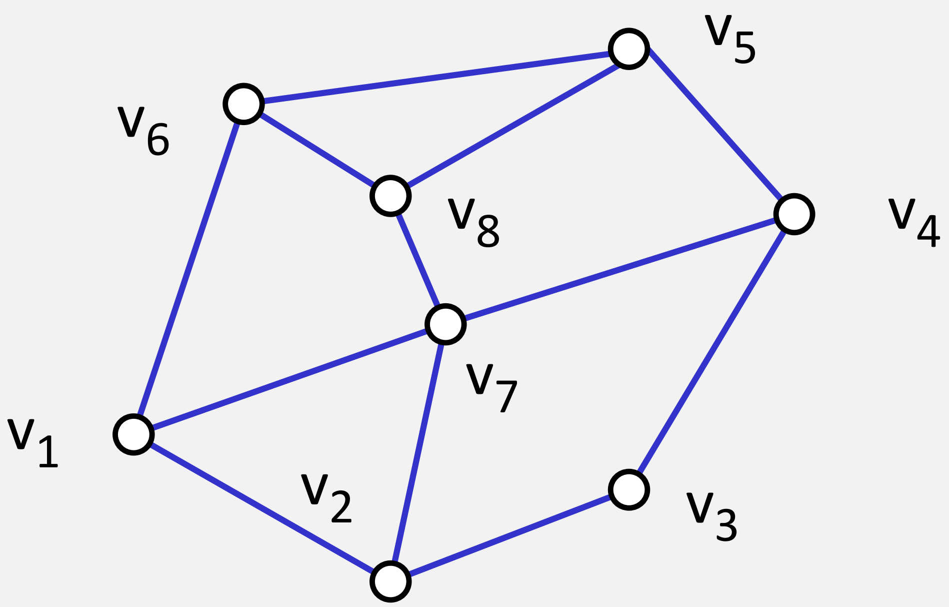
- Data for synthetic object
  - 3D model
    - Vertices: 3D position, normal, color, texture coord., for each vertex
    - Faces: polygon-vertex indices, for each face
  - Texture image





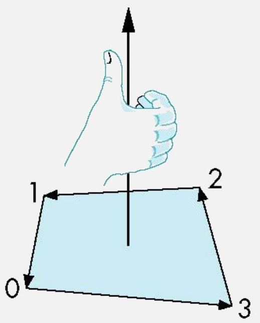
Simple Example

- Simple polygon model
  - 8 vertices, 5 faces

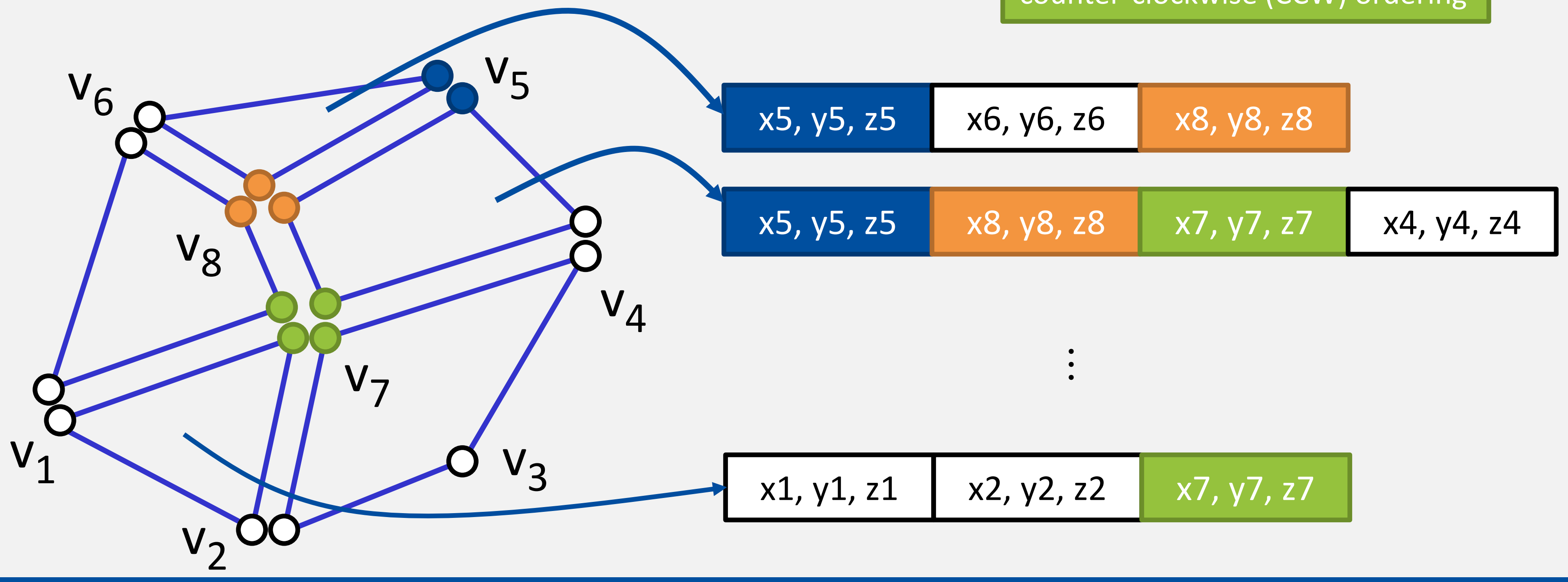


Simple Example – Polygon Soup

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



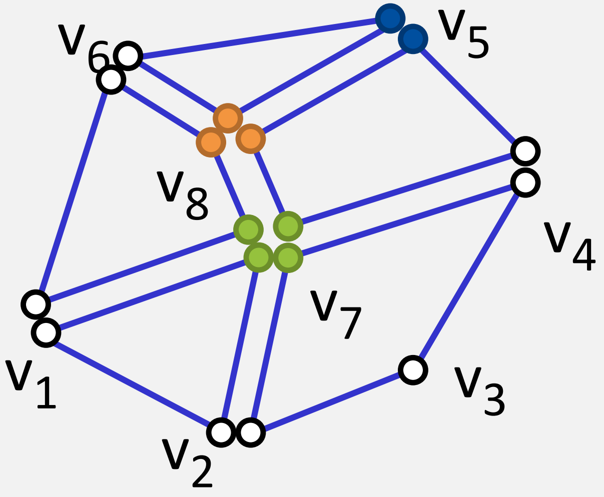
Outward facing:
counter-clockwise (CCW) ordering



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

Simple Example – Polygon Soup

- Polygon data transmission
  - 2 triangles
  - 3 quads



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

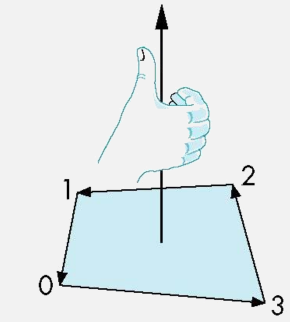
x5, y5, z5	x6, y6, z6	x8, y8, z8	x1, y1, z1	x2, y2, z2	x7, y7, z7

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

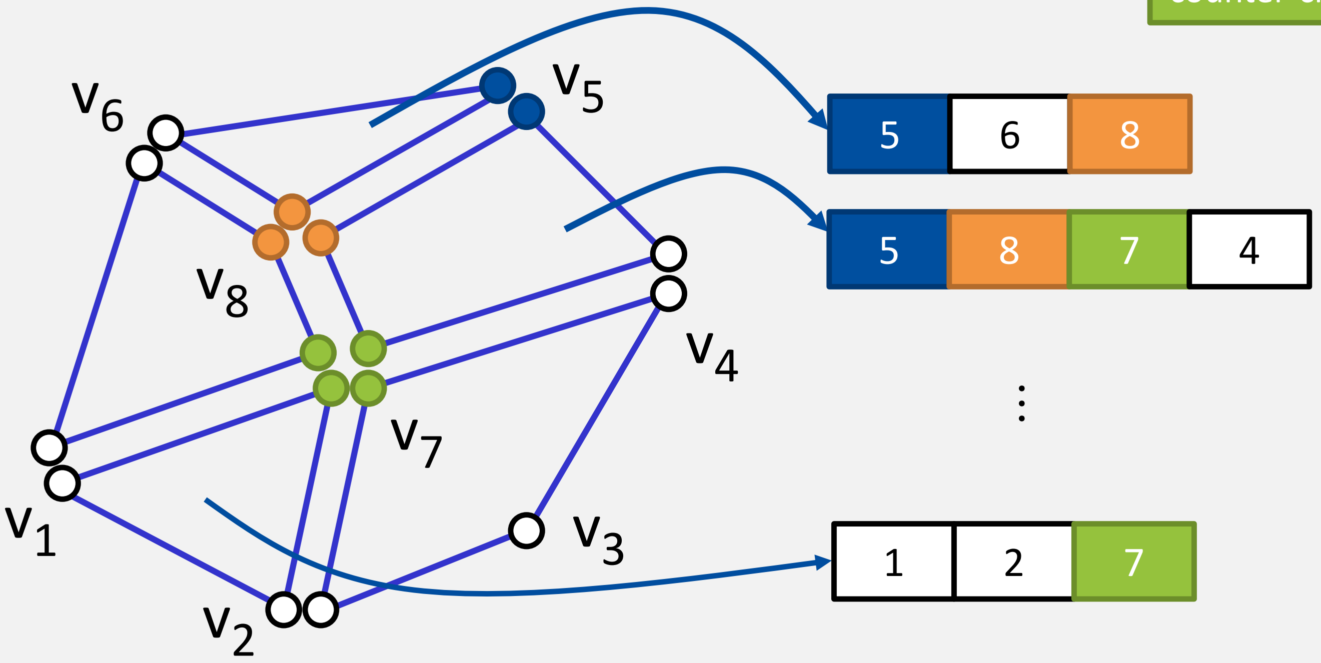
x5, y5, z5	x8, y8, z8	x7, y7, z7	x4, y4, z4	x7, y7, z7	x2, y2, z2
x3, y3, z3	x4, y4, z4	x7, y7, z7	x8, y8, z8	x6, y6, z6	x1, y1, z1

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

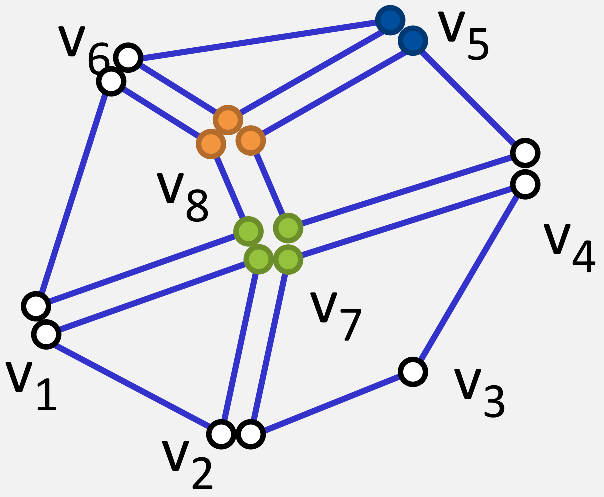


5	6	8	
5	8	7	4
⋮			
1	2	7	

$x_1\ y_1\ z_1$
$x_2\ y_2\ z_2$
$x_3\ y_3\ z_3$
$x_4\ y_4\ z_4$
$x_5\ y_5\ z_{5.}$
$x_6\ y_6\ z_6$
$x_7\ y_7\ z_7$
$x_8\ y_8\ z_8$

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

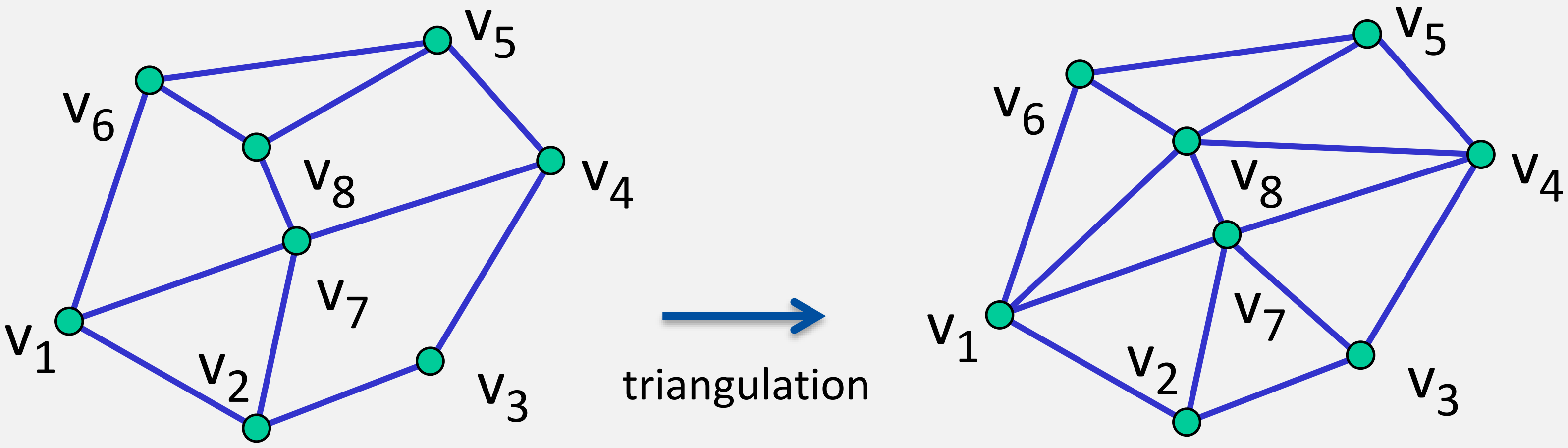
5	6	8	1	2	7

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

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

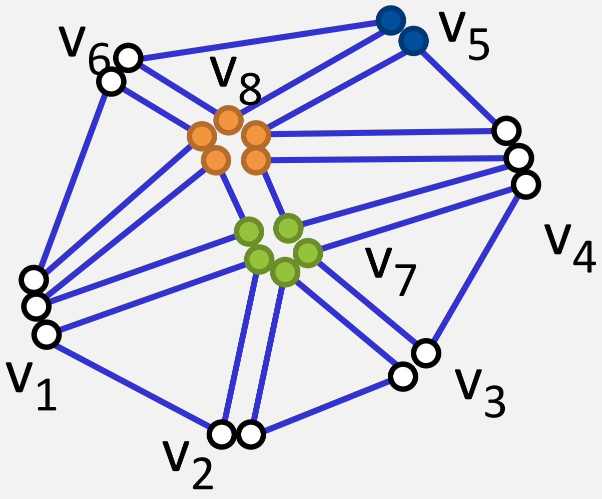
Triangle Meshes

- 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 – Triangle Soup

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

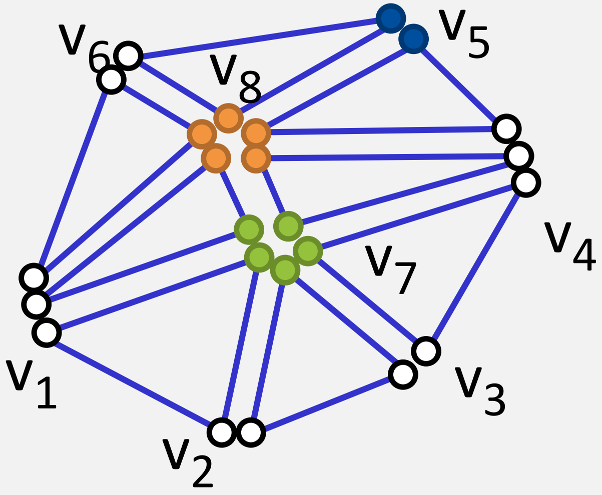


Vertex Attribute: 3D Position
Primitive type: TRIANGLES
# of Primitive: 8

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 & Triangles

- 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				

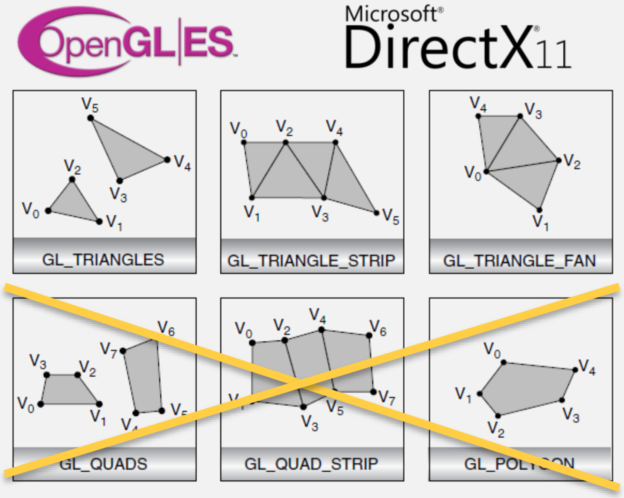
Polygon-Vertex Indices
Primitive type: TRIANGLES
# of Primitives: 2

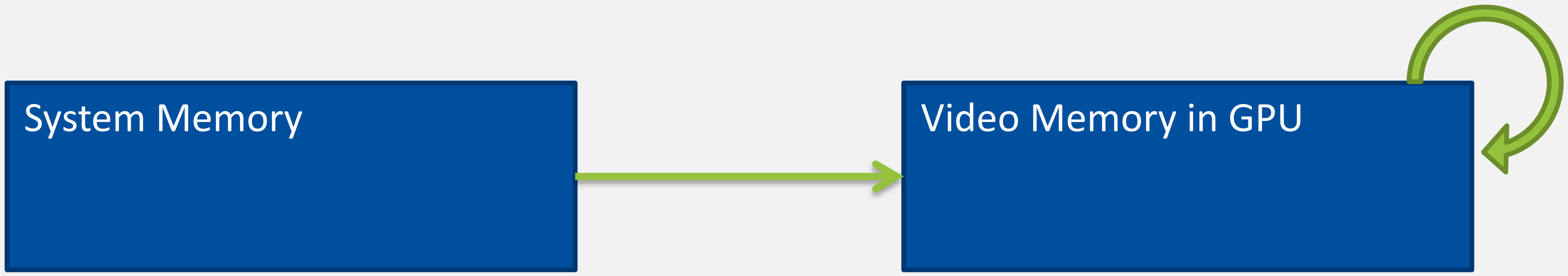
5	6	8	6	1	8	1	7	8	7	4	8
4	5	8	1	2	7	2	3	7	3	4	7

More Advantages of Triangle Meshes

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

- OpenGL ES, DirectX support triangle meshes only





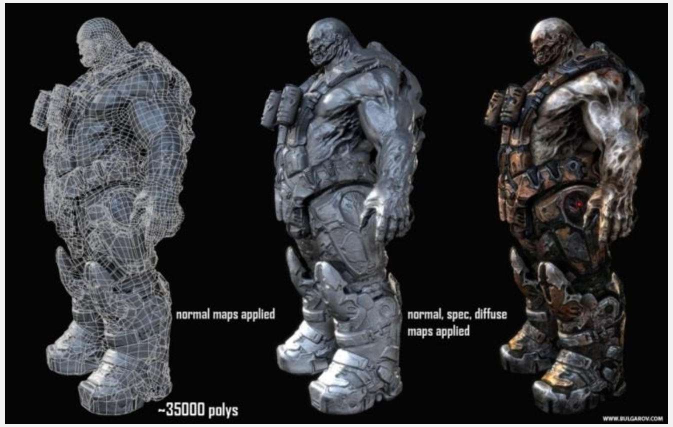
System Memory

Video Memory in GPU

Client side

Server side

Modern OpenGL Rendering Architectures

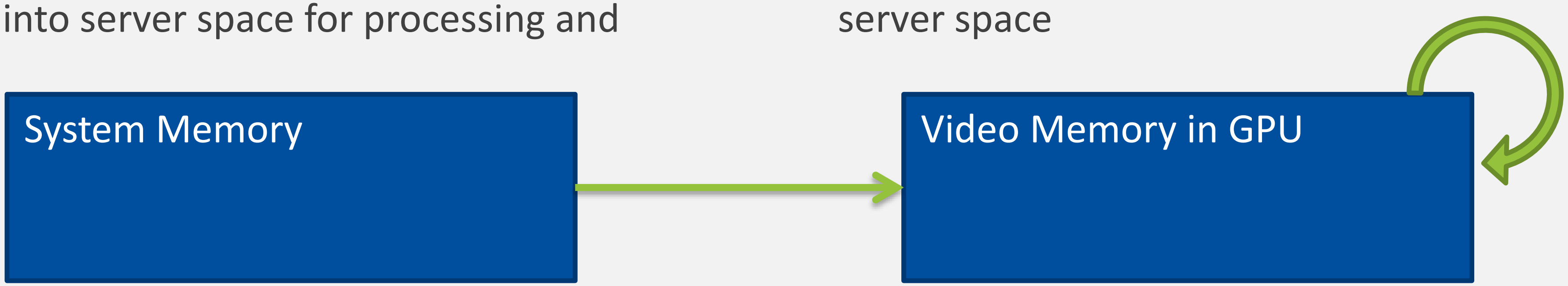


Vertex Arrays

- 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



System Memory

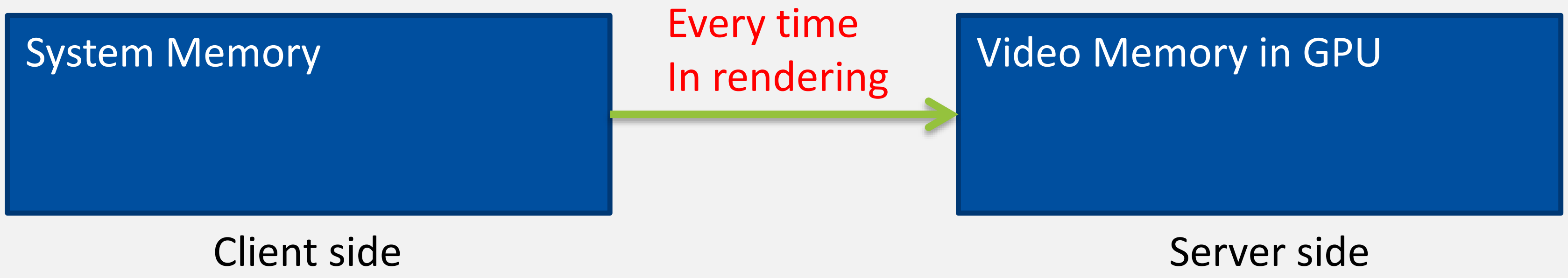
Video Memory in GPU

Client side

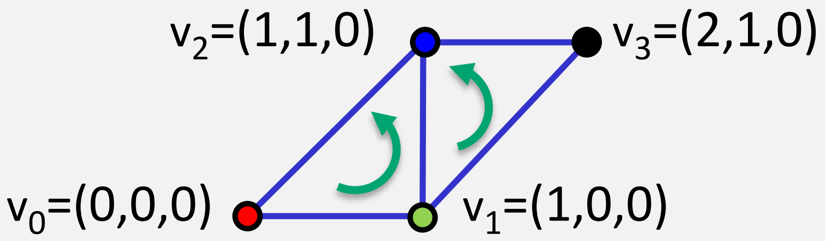
Server side

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



Triangle 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,  0,1,0,  0,0,1, 0,1,0, 0,0,0, 0,0,1 };

// ...
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, 3, GL_FLOAT, GL_FALSE, 0, color);

glDrawArrays(GL_TRIANGLES, 0, 6);

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

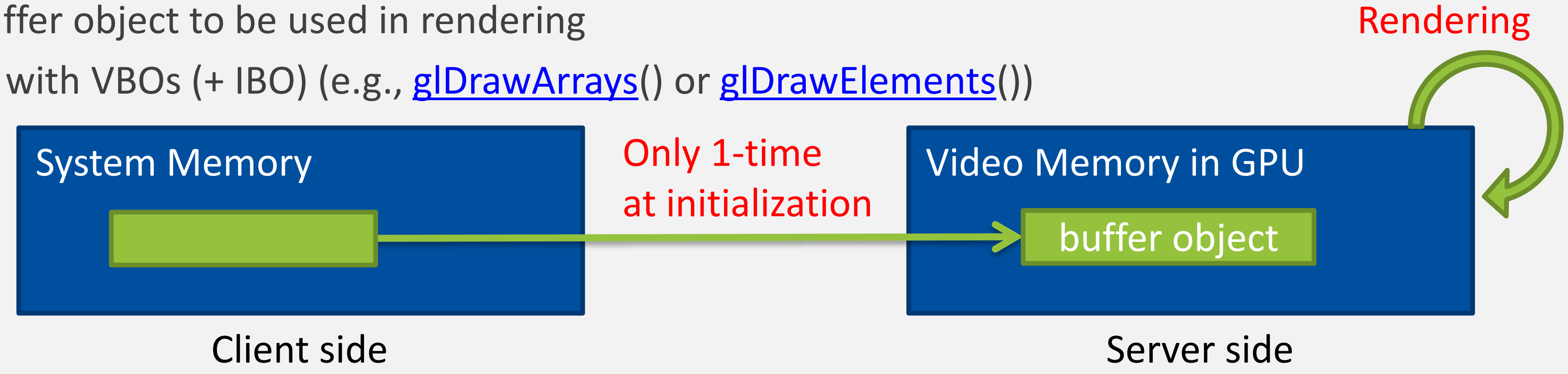
Vertex List & Triangles

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

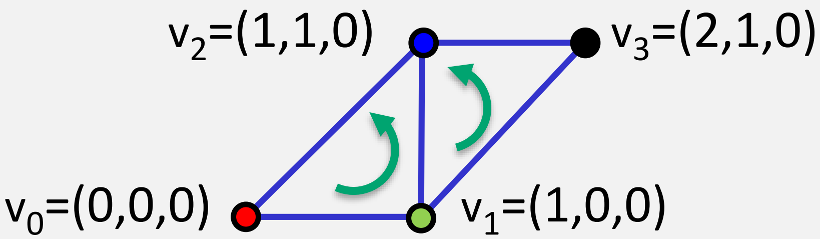
```
GLfloat position[] = { 0,0,0,  1,0,0,  1,1,0, 2,1,0 };
GLfloat color[]    = { 1,0,0,  0,1,0,  0,0,1, 0,0,0 };
GLubyte indices[]  = { 0, 1, 2, 1, 3, 2 };

// ...
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, 3, 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/Index Buffer Objects (VBOs+IBO)

- Buffer Objects (BOs) allow storing of arrays in *server* space
  - E.g.) Vertex Buffer Objects (VBOs), Index Buffer Object (IBO)
- Steps to use BOs
  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. Enable vertex attribute array
  6. Bind buffer object to be used in rendering
  7. Render with VBOs (+ IBO) (e.g., glDrawArrays() or glDrawElements())



Modern OpenGL codes – Vertex/Index Buffer Objects (VBOs+IBO)



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

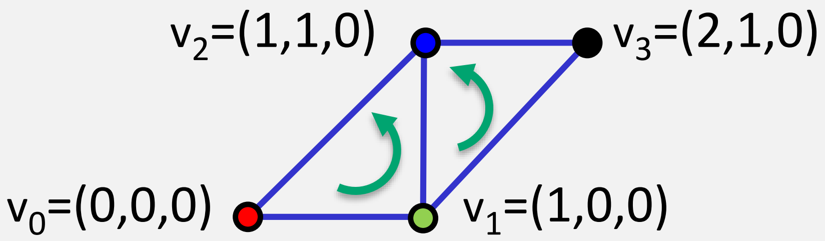
Triangle Soup (Init VBOs)

```
// buffers in client space
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,  0,1,0,  0,0,1, 0,1,0, 0,0,0, 0,0,1 };

// buffer IDs in server space
GLuint position_buffer;
GLuint color_buffer;

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

// create an index buffer & trasfer vertices data from client space to server space
glGenBuffers(1, &color_buffer);
glBindBuffer(GL_ARRAY_BUFFER, color_buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(color), color, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Vertex List & Triangles (Init VBOs + IBO)

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

// buffer IDs in server space
GLuint position_buffer;
GLuint color_buffer;
GLuint index_buffer;

// reate a VBOs & trasfer vertices data from client space to server space
// * SAME as the case of Triangle Soup on the left
// ...

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

Modern OpenGL codes – Vertex/Index Buffer Objects (VBOs+IBO)



Rendering

5. Enable vertex attribute array
6. Bind buffer object to be used in rendering
7. Render using VBOs (+IBO) (e.g., glDrawArrays() or glDrawElements())

Triangle Soup (Render w/ VBOs)

```
// buffers in client space
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,  0,1,0,  0,0,1, 0,1,0, 0,0,0, 0,0,1 };


// specifying vertex data
glEnableVertexAttribArray(GL_ARRAY_BUFFER, loc_a_position);
glBindBuffer(GL_ARRAY_BUFFER, position_buffer);
glVertexAttribPointer(loc_a_position, 3, GL_FLOAT, GL_FALSE, 0, (void*) 0);

// specifying color data
glEnableVertexAttribArray(GL_ARRAY_BUFFER, loc_a_color);
glBindBuffer(GL_ARRAY_BUFFER, color_buffer);
glVertexAttribPointer(loc_a_color, 3, GL_FLOAT, GL_FALSE, 0, (void*) 0);

glDrawArrays(GL_TRIANGLES, 0, 6);

// reset buffers
glDisableVertexAttribArray(GL_ARRAY_BUFFER, loc_a_position);
glDisableVertexAttribArray(GL_ARRAY_BUFFER, loc_a_color);
```

Vertex List & Triangles (Render w/ VBOs + IBO)

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

// specifying VBOs for shader attributes
// * SAME as the case of Triangle Soup on the left
// ...





// specifying triangle-vertex index data
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, index_buffer);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (void*) 0);

// reset buffers
glDisableVertexAttribArray(GL_ARRAY_BUFFER, loc_a_position);
glDisableVertexAttribArray(GL_ARRAY_BUFFER, loc_a_color);
```

Modern OpenGL codes – Vertex/Index Buffer Objects (VBOs+IBO)

- 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/Index Buffer Objects (VBOs+IBO)

- 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/Index Buffer Objects (VBOs+IBO)

- glGenBuffers()
  - Generate buffer object names

```
// Define an array of generic vertex attribute data
void glGenBuffers(GLsizei n, GLuint* buffers);

// n:                 Specifies the number of buffer object names to be generated
//
// buffers:           Specifies an array in which the generated buffer object names are stored
//
```

Modern OpenGL codes – Vertex/Index Buffer Objects (VBOs+IBO)

- glBindBuffer()
  - Bind a named buffer object

```
// Define an array of generic vertex attribute data
void glBindBuffer(GLenum target, GLuint buffer);

// target:          Specifies the target to which the buffer object is bound
//                  GL_ARRAY_BUFFER, GL_ELEMENT_ARRAY_BUFFER, GL_TEXTURE_BUFFER, GL_UNIFORM_BUFFER, ...
//
// buffer:          Specifies the name of a buffer object
//
```

Modern OpenGL codes – Vertex/Index Buffer Objects (VBOs+IBO)

- glBufferData()
  - Bind a named buffer object

```
// Define an array of generic vertex attribute data
void glBufferData(GLenum target, GLsizeiptr size, const void* data, GLenum usage);

// target:            Specifies the target to which the buffer object is bound
//                    GL_ARRAY_BUFFER, GL_ELEMENT_ARRAY_BUFFER, GL_TEXTURE_BUFFER, GL_UNIFORM_BUFFER, ...
//
// size:              Specifies the size in bytes of the buffer object’s new data store
//
// data:              Specifies a pointer to data that will be copied into the data store for initialization,
//                    or NULL if no data is to be copied.
//
// usage:             Specifies the expected usage pattern of the data store. The symbolic constant must be
//                    GL_STREAM_DRAW, GL_STREAM_READ, GL_STREAM_COPY,
//                    GL_STATIC_DRAW, GL_STATIC_READ, GL_STATIC_COPY,
//                    GL_DYNAMIC_DRAW, GL_DYNAMIC_READ, GL_DYNAMIC_COPY,
//
```

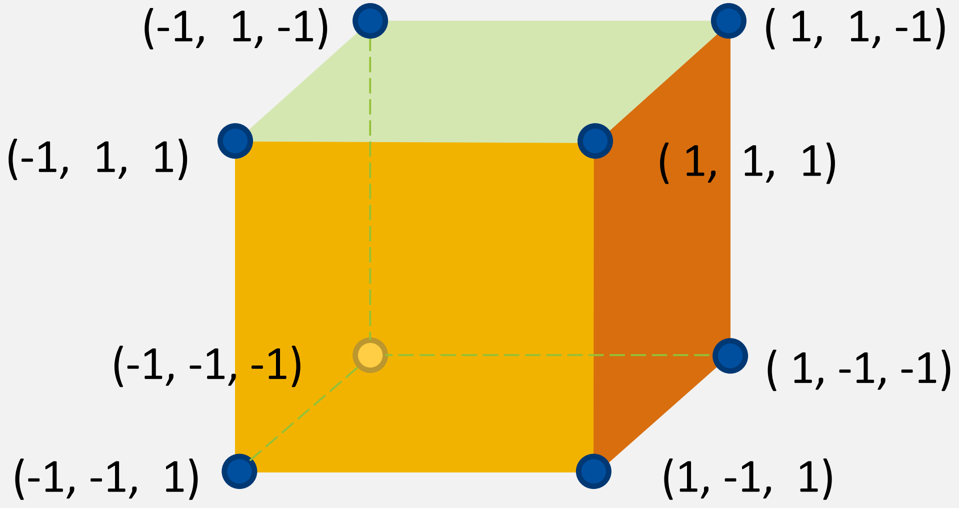
- Modeling a cube
  - Use Vertex Arrays
    - Use DrawArrays()
    - Use DrawElements()
  - Use Vertex Buffer Objects
- Quiz

Programming Practice

Synthetic Objects

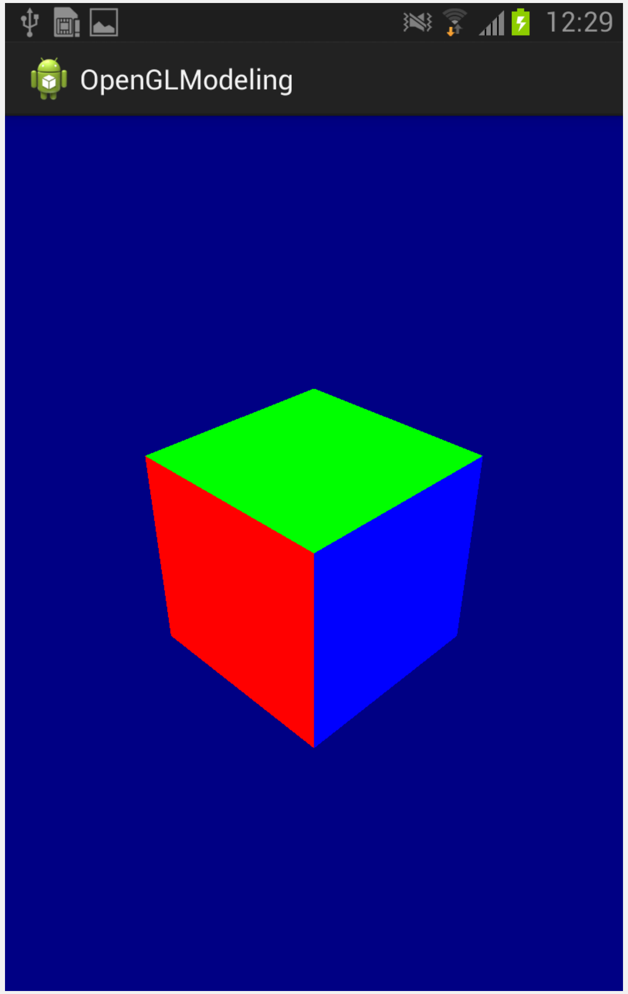
Programming Practice

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



Programming Practice

- Modeling a cube
- Draw a cube using 2 different ways
  - glDrawArrays (triangle soup)
  - glDrawElements (vertex list & triangles)



Q&A