Computer Graphics
- Draw Primitives in OpenGL

Junjie Cao @ DLUT

Spring 2016

http://jjcao.github.io/ComputerGraphics/

How to send vertex data to graphics card

- Different ways:
  - Immediate mode (glBegin / glVertex / glEnd etc.)
  - Display Lists
  - Vertex Arrays
  - **Vertex Buffer Objects (VBO) etc.**
- Immediate Mode is quite inefficient, so it’s been dropped in OpenGL ES and depreciated in OpenGL 3.0.
- Vertex Arrays or VBO are way more efficient, but generally not as straightforward to setup and use.

Immediate mode (glBegin / glVertex / glEnd etc.)

Deprecated!

OpenGL Geometric Drawing Primitives

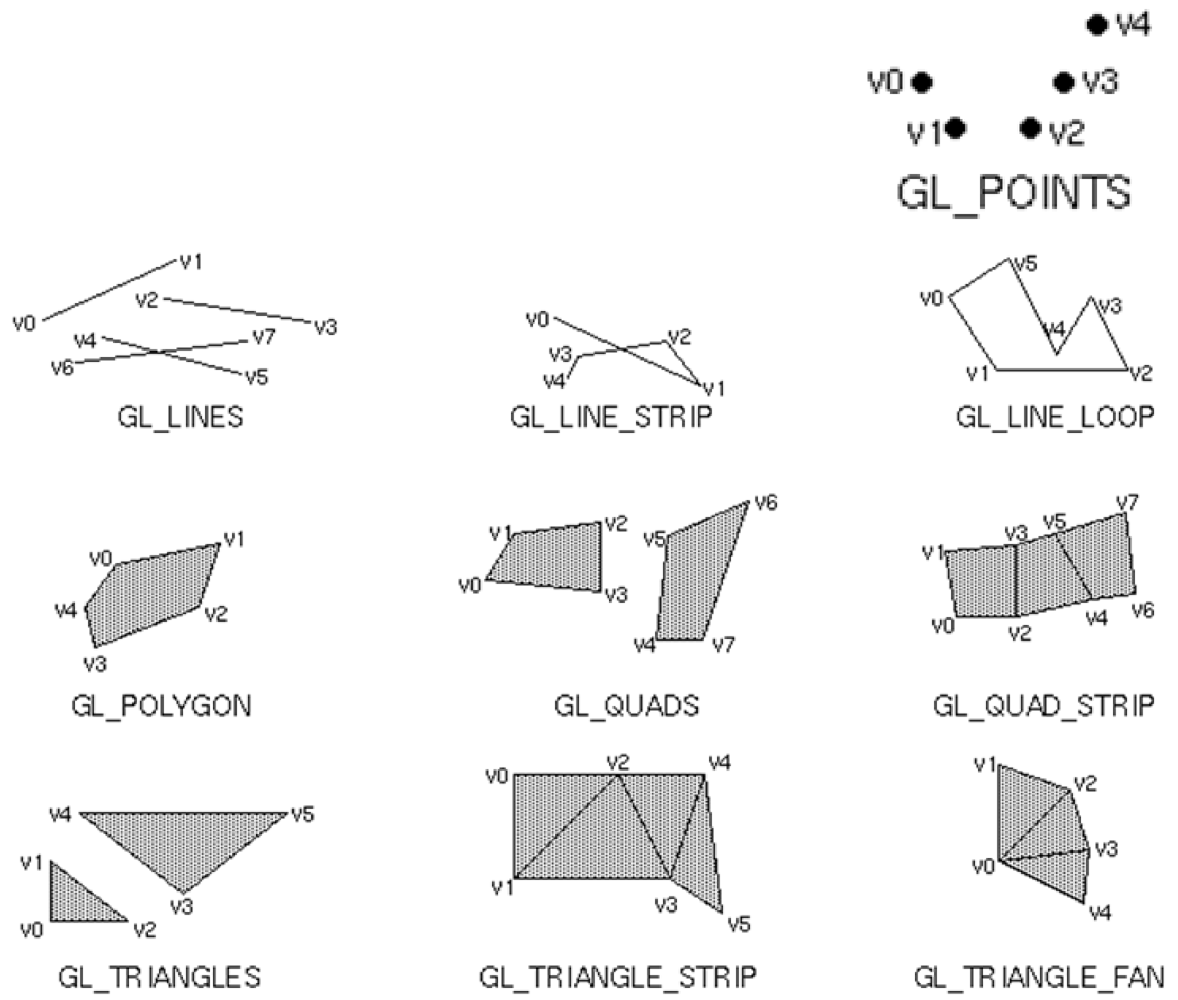
- OpenGL geometric primitives can create a set of points, a line, or a polygon from vertices

- OpenGL support **ten** primitives

- A drawing primitive must start with
  glBegin();
- And finish with
  glEnd();

- Between them the primitive

```
glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f( 0.5, 0.5);
        glVertex2f( 0.5, -0.5);
glEnd();
```



Vertex Buffer Objects (VBO)
more efficient, but not straightforward

Vertex Buffer Object (VBO)

- Motivation
  - Replacing the out-dated functions such as glBegin(), glEnd(), glVertex*(), glNormal*(), glTexCoord*, glColor*, etc to define the geometry
  - Provide per-vertex input to the GPU
  - Allowing significant increases in vertex throughput between CPU and GPU

  - A mechanism to provide generic vertex attributes to the shader, and store vertex data in video RAM
  - The programmer is free to define an arbitrary set of pervertex attributes to the vertex shader

Creating a VBO

- Step 1: Generate a new buffer object with **glGenBuffers()**
  - Create buffer objects and returns the identifiers of the buffer objects
  - void glGenBuffers(Glsizei n, Gluint* ids);
- Bind the buffer object with **glBindBuffer()**
  - Specify the target (i.e., what kind of buffer) to which the buffer object is
  - bound
  - target: GL_ARRAY_BUFFER, GL_ELEMENT_ARRAY_BUFFER, GL_PIXEL_PACK_BUFFER, GL_PIXEL_UNPACK_BUFFER
  - GL_ARRAY_BUFFER is to provide the vertex attributes, and GL_ELEMENT_ARRAY_BUFFER is to provide the triangle indices
- Copy the vertex data to the buffer object
  - **glBufferData(**Glenum target, Glsizei size, const void* data, Glenum usage)
  - Usage is the access pattern: STATIC_, STREAM_, DYNAMIC_{DRAW, COPY, READ}

Example

```
typedef struct{
float location[4];
float color[4];
} Vertex;

Vertex verts[6]; // triangle vertices
GLubyte tindices[6]; // triangle vertex indices
GLuint vboHandle[1]; // a VBO that contains interleaved positions and colors
GLuint indexVBO;
```

Example (cont'd)

```
void InitGeometry()
{
verts[0].location[0] = -0.5; verts[0].location[1] = -0.5; verts[0].location[2] = 0; verts[0].location[3] = 1;
verts[1].location[0] = -0.5; verts[1].location[1] = 0.5; verts[1].location[2] = 0; verts[1].location[3] = 1;
verts[2].location[0] = 0.5; verts[2].location[1] = 0.5; verts[2].location[2] = 0; verts[2].location[3] = 1;
verts[3].location[0] = 0.5; verts[3].location[1] = 0.5; verts[3].location[2] = 0; verts[3].location[3] = 1;
verts[4].location[0] = 0.5; verts[4].location[1] = -0.5; verts[4].location[2] = 0; verts[4].location[3] = 1;
verts[5].location[0] = -0.5; verts[5].location[1] = -0.5; verts[5].location[2] = 0; verts[5].location[3] = 1;

verts[0].color[0] = 1; verts[0].color[1] = 1; verts[0].color[2] = 0; verts[0].color[3] = 1;
verts[1].color[0] = 1; verts[1].color[1] = 1; verts[1].color[2] = 0; verts[1].color[3] = 1;
verts[2].color[0] = 1; verts[2].color[1] = 1; verts[2].color[2] = 0; verts[2].color[3] = 1;
verts[3].color[0] = 1; verts[3].color[1] = 0; verts[3].color[2] = 0; verts[3].color[3] = 1;
verts[4].color[0] = 1; verts[4].color[1] = 0; verts[4].color[2] = 0; verts[4].color[3] = 1;
verts[5].color[0] = 1; verts[5].color[1] = 0; verts[5].color[2] = 0; verts[5].color[3] = 1;

// create triangle vertex indices.
tindices[0] = 0; tindices[1] = 1; tindices[2] = 2;
tindices[3] = 3; tindices[4] = 4; tindices[5] = 5;
}
```

Example (cont'd)

```
void InitVBO(){
glGenBuffers(1, vboHandle); // create VBO handle for position & color
glBindBuffer(GL_ARRAY_BUFFER, vboHandle[0]); // bind the handle
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertex)*6, verts, GL_STATIC_DRAW); //
allocate space and copy the position data over
glBindBuffer(GL_ARRAY_BUFFER, 0); // clean up


glGenBuffers(1, &indexVBO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexVBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(GLubyte)*6, tindices,
GL_STATIC_DRAW); // load the index data
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,0); // clean up
// by now, we moved the position and color data over to the graphics card. There will be no redundant datacopy at drawing time
}
```

Draw VBOs

- Bind (like activate) the VBOs
  - The **vertex (attributes)** and **element (indices)** arrays for example
- capabilities to handle/use vertex attribute arrays on the client (CPU) side
  - By default, all client-side capabilities are disabled.
  - http://www.opengl.org/sdk/docs/man/xhtml/glEnableClientState.xml
- Specify the starting positions and strides of the vertex attributes in the VBO
  - glColorPointer(4, GL_FLOAT, sizeof(Vertex), (char*) NULL+ 16);
  - glVertexPointer(4,GL_FLOAT, sizeof(Vertex), (char*) NULL+ 0);
- Draw the geometry
  - glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (char*) NULL+0);
- Clean up
  - glDisableClientState(GL_VERTEX_ARRAY);
  - glDisableClientState(GL_COLOR_ARRAY);

Example (cont'd)

```
void display()
{
glClearColor(0,0,1,1); glClear(GL_COLOR_BUFFER_BIT);

glBindBuffer(GL_ARRAY_BUFFER, vboHandle[0]);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexVBO);

glEnableClientState(GL_VERTEX_ARRAY); // enable the vertex array on the client side
glEnableClientState(GL_COLOR_ARRAY); // enable the color array on the client side

glColorPointer(4, GL_FLOAT, sizeof(Vertex), (char*) NULL+ 16);
glVertexPointer(4,GL_FLOAT, sizeof(Vertex), (char*) NULL+ 0);

glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (char*) NULL+0);

glDisableClientState(GL_VERTEX_ARRAY); glDisableClientState(GL_COLOR_ARRAY);
glutSwapBuffers();
}
```

From Now On

- From now on, let’s not use the old OpenGL methods to specify vertex
- attributes!!
- That is, no more glBegin()/glEnd() whenever possible please!!

glGenBuffers v.s. glGenBuffersARB

根据OpenGL所支持VBO的情况，有三种方式执行渲染

1. 支持OpenGL 1.5，使用标准的VBO函数
   - glGenBuffers
2. 不支持OpenGL 1.5，但以ARB扩展的形式支持VBO
   - glGenBuffersARB
3. 不支持VBO，使用Vertex Array代替

1. glGenBuffers() is a core OpenGL function in OpenGL 1.5 and later; glGenBuffersARB() was an extension implementing the same functionality in earlier versions.

2. Unless you're developing for an ancient system, there's no longer any reason to use the ARB extension.