Computer Graphics

spring, 2013

Chapter 6
Vertex Attributes, Vertex Arrays, and Buffer Objects

Generic Vertex Attributes

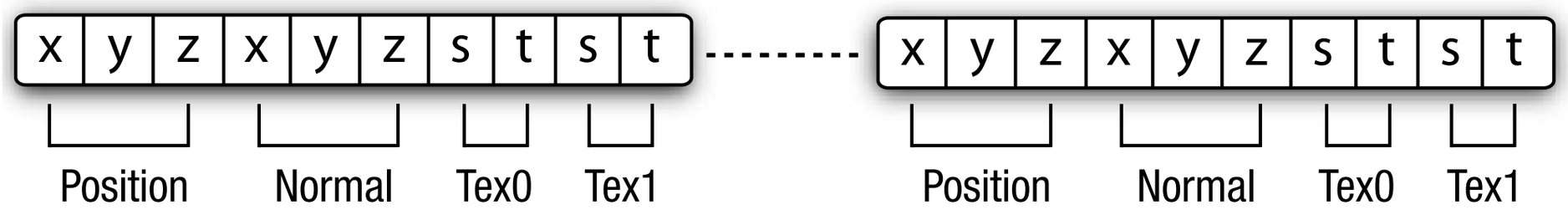
- OpenGL ES 1.1
  - predefined
    - position, normal, color, texcoords
- OpenGL ES 2
  - user-defined (generic)
  - >= 8 (GL_MAX_VERTEX_ATTRIBS)

Constant Vertex Attribute

- Same for all vertices of a primitive
- Set by glVertexAttrib*
- Default values are 0.0 for y&z, 1.0 for w
- Supports float only
- Difference from uniforms? --> Uniforms are constant for all primitives in a rendering frame

Vertex Arrays

- Buffers stored in the “client space” --> copied to graphics mem when glDrawArrays or glDrawElements is called
- glVertexAttribPointer
- Two methods
  - All attribs in a single buffer -- “array of structures”
  - Each attrib in a separate buffer -- “structure of arrays”



```
#define VERTEX_POS_SIZE           3   // x, y and z
#define VERTEX_NORMAL_SIZE        3   // x, y and z
#define VERTEX_TEXCOORD0_SIZE     2   // s and t
#define VERTEX_TEXCOORD1_SIZE     2   // s and t

#define VERTEX_POS_INDX           0
#define VERTEX_NORMAL_INDX        1
#define VERTEX_TEXCOORD0_INDX     2
#define VERTEX_TEXCOORD1_INDX     3

// the following 4 defines are used to determine location of various
// attributes if vertex data is are stored as an array of structures
#define VERTEX_POS_OFFSET         0
#define VERTEX_NORMAL_OFFSET      3
#define VERTEX_TEXCOORD0_OFFSET   6
#define VERTEX_TEXCOORD1_OFFSET   8

#define VERTEX_ATTRIB_SIZE   VERTEX_POS_SIZE + \
                             VERTEX_NORMAL_SIZE + \
                             VERTEX_TEXCOORD0_SIZE + \
                             VERTEX_TEXCOORD1_SIZE

float *p  = malloc(numVertices * VERTEX_ATTRIB_SIZE
                   * sizeof(float));

// position is vertex attribute 0
glVertexAttribPointer(VERTEX_POS_INDX, VERTEX_POS_SIZE,
                      GL_FLOAT, GL_FALSE,
                      VERTEX_ATTRIB_SIZE * sizeof(float),
                      p);
 // normal is vertex attribute 1
 glVertexAttribPointer(VERTEX_NORMAL_INDX, VERTEX_NORMAL_SIZE,
                       GL_FLOAT, GL_FALSE,
                       VERTEX_ATTRIB_SIZE * sizeof(float),
                       (p +  VERTEX_NORMAL_OFFSET));

 // texture coordinate 0 is vertex attribute 2
 glVertexAttribPointer(VERTEX_TEXCOORD0_INDX, VERTEX_TEXCOORD0_SIZE,
                       GL_FLOAT, GL_FALSE,
                       VERTEX_ATTRIB_SIZE * sizeof(float),
                       (p +  VERTEX_TEXCOORD0_OFFSET));

 // texture coordinate 1 is vertex attribute 3
 glVertexAttribPointer(VERTEX_TEXCOORD1_INDX, VERTEX_TEXCOORD1_SIZE,
                       GL_FLOAT, GL_FALSE,
                       VERTEX_ATTRIB_SIZE * sizeof(float),
                       (p + VERTEX_TEXCOORD1_OFFSET));
```

```
float *position  = malloc(numVertices * VERTEX_POS_SIZE *
                          sizeof(float));
float *normal    = malloc(numVertices * VERTEX_NORMAL_SIZE *
                          sizeof(float));
float *texcoord0 = malloc(numVertices * VERTEX_TEXCOORD0_SIZE *
                          sizeof(float));
float *texcoord1 = malloc(numVertices * VERTEX_TEXCOORD1_SIZE *
                          sizeof(float));

// position is vertex attribute 0
glVertexAttribPointer(VERTEX_POS_INDX, VERTEX_POS_SIZE,
                      GL_FLOAT, GL_FALSE,
                      VERTEX_POS_SIZE * sizeof(float), position);

// normal is vertex attribute 1
glVertexAttribPointer(VERTEX_NORMAL_INDX, VERTEX_NORMAL_SIZE,
                      GL_FLOAT, GL_FALSE,
                      VERTEX_NORMAL_SIZE * sizeof(float), normal);

// texture coordinate 0 is vertex attribute 2
glVertexAttribPointer(VERTEX_TEXCOORD0_INDX, VERTEX_TEXCOORD0_SIZE,
                      GL_FLOAT, GL_FALSE, VERTEX_TEXCOORD0_SIZE *
                      sizeof(float), texcoord0);
// texture coordinate 1 is vertex attribute 3
glVertexAttribPointer(VERTEX_TEXCOORD1_INDX, VERTEX_TEXCOORD1_SIZE,
                      GL_FLOAT, GL_FALSE,
                      VERTEX_TEXCOORD1_SIZE * sizeof(float),
                      texcoord1);
```

Performance

- Accessing data -- AoS
  - Efficient sequential accessing
- Updating one attribs -- SoA
  - Efficient bulk update

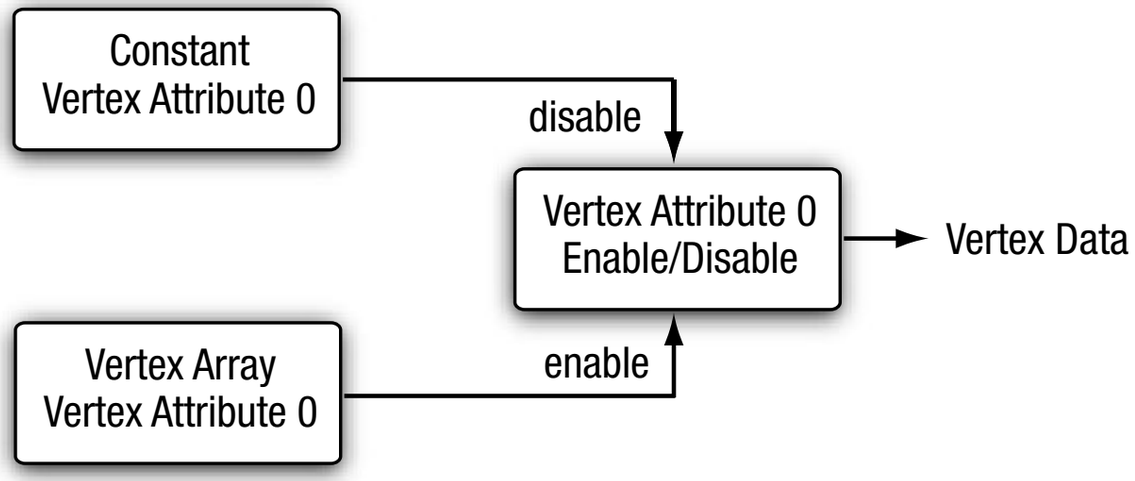
Data Format

- Smaller memory footprint --> lower mem. bandwidth --> better performance
- Guideline
  - GL_HALF_FLOAT_OES for texcoords, normals, binormals, tangent vectors, etc. (+position if possible)
  - GL_UNSIGNED_BYTE for colors
  - GL_FLOAT / GL_FIXED for positions

Attributes Conversion

- Vertex attribs are INTERNALLY stored as a single precision floating-point number
- Converted if not a float
- Normalized flag controls the conversion
  - false: converted directly
  - true: mapped to
    - [-1.0,1.0] for GL_BYTE, GL_SHORT, GL_FIXED
    - [0.0,1.0] for GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT

Constant Attrib vs. Vertex Array



- glEnable(Disable)VertexAttribArray

```
GLbyte vertexShaderSrc[] =
      "attribute vec4 a_position;    \n"
      "attribute vec4 a_color;       \n"
      "varying vec4   v_color;       \n"
      "void main()                   \n"
      "{                             \n"
      "    v_color = a_color;        \n"
      "    gl_Position = a_position; \n"
      "}";

GLbyte fragmentShaderSrc[] =
      "varying vec4 v_color;         \n"
      "void main()                   \n"
      "{                             \n"
      "    gl_FragColor = v_color;   \n"
      "}";

GLfloat   color[4] = { 1.0f, 0.0f, 0.0f, 1.0f };
GLfloat   vertexPos[3 * 3];  // 3 vertices, with (x,y,z) per-vertex
GLuint    shaderObject[2];
GLuint    programObject;

shaderObject[0] = LoadShader(vertexShaderSrc, GL_VERTEX_SHADER);
shaderObject[1] = LoadShader(fragmentShaderSrc, GL_FRAGMENT_SHADER);

programObject = glCreateProgram();
glAttachShader(programObject, shaderObject[0]);
glAttachShader(programObject, shaderObject[1]);

glVertexAttrib4fv(0, color);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, vertexPos);
glEnableVertexAttribArray(1);

glBindAttribLocation(programObject, 0, "a_color");
glBindAttribLocation(programObject, 1, "a_position");

glLinkProgram(programObject);
glUseProgram(programObject);

glDrawArrays(GL_TRIANGLES, 0, 3);
```

Attribs in a Vertex Shader

- Declared with “attribute” qualifier
- Arrays & structures are not allowed
- Attribs are not packed
  - float, vec2, vec3 --> one vec4 attrib
  - mat2, mat3, mat4 --> 2,3,4 vec4 attribs
- Each component stored internally as a 32-bit single precision FP value
- Care needed not to waste space
- Read-only
- Not allocated if not used
- # & list of active attribs can be queried

Binding Vertex Attribs

- Two ways
  - By app manually
    - glBindAttribLocation
  - By GLES2
    - Index assigned if not specified by glBindAttribLocation
    - glGetAttribLocation

Vertex Buffer Objects

- Vertex attribs & element indices can be stored in the graphics memory using VBOs --> best performance
- Two types
  - array buffer objects
    - Specified by GL_ARRAY_BUFFER
    - To store vertex attribs
  - element array buffer objects
    - Specified by GL_ELEMENT_ARRAY_BUFFER
    - To store primitive indices

Creating & Binding VBOs

```
void   initVertexBufferObjects(vertex_t *vertexBuffer,
                               GLushort *indices,
                               GLuint numVertices, GLuint numIndices
                               GLuint *vboIds)
{
    glGenBuffers(2, vboIds);

    glBindBuffer(GL_ARRAY_BUFFER, vboIds[0]);
    glBufferData(GL_ARRAY_BUFFER, numVertices * sizeof(vertex_t),
                 vertexBuffer, GL_STATIC_DRAW);

    // bind buffer object for element indices
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboIds[1]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
                 numIndices * sizeof(GLushort),indices,
                 GL_STATIC_DRAW);
}
```

Creating & Binding VBOs (cont'd)

- glGenBuffers
- glBindBuffer -- A VBO is allocated when first bound
- States
  - GL_BUFFER_SIZE
  - GL_BUFFER_USAGE -- hint for usage
    - GL_STATIC_DRAW -- specified once / used many times
    - GL_DYNAMIC_DRAW -- repeatedly / many times
    - GL_STREAM_DRAW -- once / few time
- Reading buffer data not supported

Allocation & Initialization

- glBufferData
- glBufferSubData

```
#define VERTEX_POS_SIZE         3   // x, y and z
#define VERTEX_NORMAL_SIZE      3   // x, y and z
#define VERTEX_TEXCOORD0_SIZE   2   // s and t

#define VERTEX_POS_INDX         0
#define VERTEX_NORMAL_INDX      1
#define VERTEX_TEXCOORD0_INDX   2

//
// vertices   - pointer to a buffer that contains vertex attribute
//              data
// vtxStride  - stride of attribute data / vertex in bytes
// numIndices - number of indices that make up primitive
//              drawn as triangles
// indices    - pointer to element index buffer.
//
void   drawPrimitiveWithoutVBOs(GLfloat *vertices, GLint vtxStride,
                                GLint numIndices, GLushort *indices)
{
    GLfloat   *vtxBuf = vertices;

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

    glEnableVertexAttribArray(VERTEX_POS_INDX);
    glEnableVertexAttribArray(VERTEX_NORMAL_INDX);
    glEnableVertexAttribArray{VERTEX_TEXCOORD0_INDX);

    glVertexAttribPointer(VERTEX_POS_INDX, VERTEX_POS_SIZE,
                          GL_FLOAT, GL_FALSE, vtxStride, vtxBuf);
    vtxBuf += VERTEX_POS_SIZE;
    glVertexAttribPointer(VERTEX_NORMAL_INDX, VERTEX_NORMAL_SIZE,
                          GL_FLOAT, GL_FALSE, vtxStride, vtxBuf);
    vtxBuf += VERTEX_NORMAL_SIZE;
    glVertexAttribPointer(VERTEX_TEXCOORD0_INDX,
                          VERTEX_TEXCOORD0_SIZE, GL_FLOAT,
                          GL_FALSE, vtxStride, vtxBuf);

    glBindAttribLocation(program, VERTEX_POS_INDX, "v_position");
    glBindAttribLocation(program, VERTEX_NORMAL_INDX, "v_normal");
    glBindAttribLocation(program, VERTEX_TEXCOORD0_INDX,
                         "v_texcoord");

    glDrawElements(GL_TRIANGLES, numIndices, GL_UNSIGNED_SHORT,
                   indices);
}
```

array of structures

```
void   drawPrimitiveWithVBOs(GLint numVertices, GLfloat *vtxBuf,
                             GLint vtxStride, GLint numIndices,
                             GLushort *indices)
{
    GLuint   offset = 0;
    GLuint   vboIds[2];

    // vboIds[0] - used to store vertex attribute data
    // vboIds[1] - used to store element indices
    glGenBuffers(2, vboIds);

    glBindBuffer(GL_ARRAY_BUFFER, vboIds[0]);
    glBufferData(GL_ARRAY_BUFFER, vtxStride * numVertices,
                 vtxBuf, GL_STATIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboIds[1]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
                 sizeof(GLushort) * numIndices,
                 indices, GL_STATIC_DRAW);

    glEnableVertexAttribArray(VERTEX_POS_INDX);
    glEnableVertexAttribArray(VERTEX_NORMAL_INDX);
    glEnableVertexAttribArray{VERTEX_TEXCOORD0_INDX);

    glVertexAttribPointer(VERTEX_POS_INDX, VERTEX_POS_SIZE,
                          GL_FLOAT, GL_FALSE, vtxStride,
                          (const void*)offset);

    offset += VERTEX_POS_SIZE * sizeof(GLfloat);
    glVertexAttribPointer(VERTEX_NORMAL_INDX, VERTEX_NORMAL_SIZE,
                          GL_FLOAT, GL_FALSE, vtxStride,
                          (const void*)offset);

    offset += VERTEX_NORMAL_SIZE * sizeof(GLfloat);
    glVertexAttribPointer(VERTEX_TEXCOORD0_INDX,
                          VERTEX_TEXCOORD0_SIZE,
                          GL_FLOAT, GL_FALSE, vtxStride,
                          (const void*)offset);
    glBindAttribLocation(program, VERTEX_POS_INDX, "v_position");
    glBindAttribLocation(program, VERTEX_NORMAL_INDX, "v_normal");
    glBindAttribLocation(program, VERTEX_TEXCOORD0_INDX,
                         "v_texcoord");

    glDrawElements(GL_TRIANGLES, numIndices, GL_UNSIGNED_SHORT, 0);

    glDeleteBuffers(2, vboIds);
}
```

structure of arrays

```
#define VERTEX_POS_SIZE         3   // x, y and z
#define VERTEX_NORMAL_SIZE      3   // x, y and z
#define VERTEX_TEXCOORD0_SIZE   2   // s and t

#define VERTEX_POS_INDX         0
#define VERTEX_NORMAL_INDX      1
#define VERTEX_TEXCOORD0_INDX   2

//
// numVertices - number of vertices
// vtxBuf - an array of pointers describing attribute data
// vtxStrides - an array of stride values for each attribute
// numIndices - number of element indices of primitive
// indices - actual element index buffer
//
void   drawPrimitiveWithVBOs(GLint numVertices,
                             GLfloat **vtxBuf, GLint *vtxStrides,
                             GLint numIndices, GLushort *indices)
{
    GLuint   vboIds[4];

    // vboIds[0] - used to store vertex position
    // vboIds[1] - used to store vertex normal
    // vboIds[2] - used to store vertex texture coordinate 0
```

```
// vboIds[3] - used to store element indices
glGenBuffers(4, vboIds);

glBindBuffer(GL_ARRAY_BUFFER, vboIds[0]);
glBufferData(GL_ARRAY_BUFFER, vtxStrides[0] * numVertices,
             vtxBuf[0], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, vboIds[1]);
glBufferData(GL_ARRAY_BUFFER, vtxStrides[1] * numVertices,
             vtxBuf[1], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, vboIds[2]);
glBufferData(GL_ARRAY_BUFFER, vtxStrides[2] * numVertices,
             vtxBuf[2], GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboIds[3]);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,
             sizeof(GLushort) * numIndices,
             indices, GL_STATIC_DRAW);

glBindBuffer(GL_ARRAY_BUFFER, vboIds[0]);
glEnableVertexAttribArray(VERTEX_POS_INDX);
glBindBuffer(GL_ARRAY_BUFFER, vboIds[1]);
glEnableVertexAttribArray(VERTEX_NORMAL_INDX);
glBindBuffer(GL_ARRAY_BUFFER, vboIds[2]);
glEnableVertexAttribArray{VERTEX_TEXCOORD0_INDX);

glVertexAttribPointer(VERTEX_POS_INDX, VERTEX_POS_SIZE,
                      GL_FLOAT, GL_FALSE, vtxStrides[0], 0);
glVertexAttribPointer(VERTEX_NORMAL_INDX, VERTEX_NORMAL_SIZE,
                      GL_FLOAT, GL_FALSE, vtxStrides[1], 0);
glVertexAttribPointer(VERTEX_TEXCOORD0_INDX,
                      VERTEX_TEXCOORD0_SIZE,
                      GL_FLOAT, GL_FALSE, vtxStrides[2], 0);

glBindAttribLocation(program, VERTEX_POS_INDX, "v_position");
glBindAttribLocation(program, VERTEX_NORMAL_INDX, "v_normal");
glBindAttribLocation(program, VERTEX_TEXCOORD0_INDX,
                     "v_texcoord");

glDrawElements(GL_TRIANGLES, numIndices, GL_UNSIGNED_SHORT, 0);

glDeleteBuffers(4, vboIds)
}
```

Release

- glDeleteBuffers

Mapping Buffer Objects

- glMapBufferOES
- glUnmapBufferOES
  - Should be only used if the whole buffer is updated
  - Expensive compared to glBufferData
  - Doesn’t seem to be supported by all implementations --> Do not use