March 4, 2025, Brno
Author: David Procházka

Vertex Buffer Objects

Graphic Application Development

MENDELU
Faculty
of Business
and Economics

Table of content

Vertex Buffer Objects

- So far, we stored vertices data into operating memory of a computer. VBOs’ allows us to store the data directly in a graphics card memory.
- Therefore, during the re-rendering, it is not necessary to upload them repeatedly.
- Usage of VBO is the only recommended method from OpenGL (3.0+).

Rendering using VBOs

There are five key operations with the VBOs:

1. Generating name for a buffer,
2. binding the buffer,
3. storing data in the buffer,
4. drawing the data in the buffer,
5. removing buffer from memory.

Generating the name

- `glGenBuffers(GLsizei n, GLuint *buffers)`.
- The parameter n determines how many parameters we want to generate.
- Generated names are store into `buffers` field.
- This field contains a sequence of (unsigned) integer names.
- The generating function guarantees that the names are unique.
- Following example shows generation of a single name:

```
GLuint bufferID;
glGenBuffers(1, &bufferID);
```

- Releasing of a buffer with given name can be done using function `glDeleteBuffers()`:

```
glDeleteBuffers(1, &bufferID);
```

Creating a buffer

- Let's create a buffer in the card memory:
- `void glBindBuffer(GLenum target, GLuint buffer).`
- `buffer` – name of a buffer created in the previous step.
- `target` – type of a buffer, it can contain:
  - `GL_ARRAY_BUFFER` – vertices,
  - `GL_ELEMENT_ARRAY_BUFFER` – indices on vertices,
  - `GL_PIXEL_PACK_BUFFER, GL_PIXEL_UNPACK_BUFFER` – rasters (not important right now).
- Example:

```
glBindBuffer(GL_ARRAY_BUFFER, bufferID)
```

Storing the vertices data

- `void glBufferData(GLenum target, GLsizeiptr size, const GLvoid data, GLenum usage)`
- `target` – (again) the buffer type,
- `size` – size of data stored into the buffer,
- `data` – pointer on a list with vertices,
- `usage` – how the buffer will be used.
  - Usage is described because of the performance optimization.
  - It will work every time, just not so efficiently.

  1. `GL_STREAM_DRAW, GL_STREAM_READ, GL_STREAM_COPY,`
  2. `GL_STATIC_DRAW, GL_STATIC_READ, GL_STATIC_COPY,`
  3. `GL_DYNAMIC_DRAW, GL_DYNAMIC_READ, GL_DYNAMIC_COPY.`

Explanation of the keywords in the buffer usage

How frequently is content accessed

Parameter	Meaning
`STREAM`	data will be once written, a few times read
`STATIC`	once written, frequently read
`DYNAMIC`	both written and read frequently

Who creates the buffer content

Parameter	Meaning
`DRAW`	filled by an app and used by the OpenGL for rendering
`READ`	filled by the OpenGL and read by an app
`COPY`	filled by the OpenGL and used by the OpenGL for rendering

Storing data into a buffer – example

- We have an array `trinagles` with vertices data:

```
glBufferData(
  GL_ARRAY_BUFFER,   // buffer typ
  sizeof(triangles), // amount of data stored
  triangles,         // where are the data
  GL_STATIC_DRAW);   // how it will be used
```

- If we call `glBufferData()` on a buffer already containing data, the previously stored data will be erased.
- If you attempt to store more data than is free memory on a card, `GL_OUT_OF_MEMORY` exception will be raised.

Storing data into a buffer – separate initialization

- We can initialize a buffer as empty and store data in a separate step.
- For the data storage after initialization, we can use command `void glBufferSubData(GLenum target, GLintptr offset, GLsizeiptr size, const GLvoid* data)`.
- The first param is the buffer type, the `offset` param determines from what position will be data written (0 = beginning of a buffer), the third is the data size, the last one is the data source.
- Example of re-writing the whole buffer:

1.
```
glBufferData(GL_ARRAY_BUFFER, sizeof(triangles),
NULL, GL_STATIC_DRAW);
```
2.
```
glBufferSubData(GL_ARRAY_BUFFER, 0,
sizeof(triangles), triangles);
```

Pointer definition

- The pointer definition is the same as it was when we used vertex arrays.
- The difference is in the last param – the name of the array is always zero
- In that case, OpenGL automatically reads data from active VBO.
- Example: `glVertexPointer(2, GL_INT, 0, 0);` or `glVertexAttribPointer(m_colAttr, 2, GL_INT, GL_FALSE, 0, 0);` when we use shaders.
- Therefore, it is necessary to define a pointer after binding of appropriate buffer

Table of content

VBO initialization

```
GLuint vertexID; // buffer ids'
GLuint colorID;
...


// name generation
glGenBuffers(1, &vertexID);
glGenBuffers(1, &colorID);
...
```

Vertices

```
GLint triangles[] = {
  10, 10,
  320, 470,
  630, 10,
};

GLfloat colors[] = {
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0
};
```

Creating VBOs’

```
glBindBuffer(GL_ARRAY_BUFFER, vertexID);

glBufferData(GL_ARRAY_BUFFER, sizeof(triangles),
      triangles, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, colorID);
glBufferData(GL_ARRAY_BUFFER, sizeof(colors),
      colors, GL_STATIC_DRAW);
```

Activation and rendering without shaders

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);

glBindBuffer(GL_ARRAY_BUFFER, vertexID);
glVertexPointer(2, GL_INT, 0, 0);
glBindBuffer(GL_ARRAY_BUFFER, colorID);
glColorPointer(3, GL_FLOAT, 0, 0);

glDrawArrays(GL_TRIANGLES, 0, 3);

glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```

Activation and rendering using shaders

```
glEnableVertexAttribArray(m_positionAttribute);
glEnableVertexAttribArray(m_colorAttribute);

glBindBuffer(GL_ARRAY_BUFFER, m_vertexBufferId);
glVertexAttribPointer(m_posAttr, 2, GL_FLOAT,
                      GL_FALSE, 0, 0);
glBindBuffer(GL_ARRAY_BUFFER, m_colorBufferId);
glVertexAttribPointer(m_colAttr, 3, GL_FLOAT,
                      GL_FALSE, 0, 0);

glDrawArrays(GL_TRIANGLES, 0, 3);

glDisableVertexAttribArray(m_positionAttribute);
glDisableVertexAttribArray(m_colorAttribute);
```

Table of content

Summary

- VBOs' are the only recommended method for data storage from OpenGL 3.0.
- Data are stored directly in the graphics card memory.
- Following operations must be always implemented:
    1. generating name for a buffer,
    2. binding the buffer,
    3. storing data in the buffer,
    4. drawing the data in the buffer,
    5. removing buffer from memory.