1 (Practice 6) Vertex Buffer Objects (VBOs)

In the previous practice we have seen how to draw a mesh with data on main memory (arrays of vertices, normals and triangle index allocated with malloc()). With large applications simulating in realtime worlds and hundreds of characters this approach would be unreasonable. You will have to upload thousands of megabytes per seconds to the GPU in order to draw your scene. Because the bus between GPU and CPU has a limited bandwidth this may result in serious slowdowns.

OpenGL offers a solution to explicitly upload data on GPU and modify it on the fly. This way data can be uploaded once and reused later without any memory transfer costs. This mechanism is called buffer objects. Buffer objects will allow you to store mesh informations, textures or other buffers on GPU and gives means to partially update them.

The practice focus on vertex buffer objects (VBOs) which are used to store mesh data. Although OpenGL function name can change between the buffer object types (frame/pixel/vertex/... buffer objects) the logic stays pretty much the same:

- First you tell OpenGL to create the buffer object.
- Before using any function that modify the buffer you have to tell openGL which buffer you are using by binding it.
- You allocate and copy data to the buffer object as you would with standard arrays but using dedicated openGL functions.

A little example:

```
{
    GLfloat vertices[nb_vertices * 3];
    GLuint  index   [nb_faces    * 3];
    // Init arrays of normals, vertices and index
    ...
    // Creates three VBOs and store their identifier in vboID
    GLuint vboId[2];
    glGenBuffers(2, vboId);

    // Bind the first vbo to work on it
    glBindBuffer(GL_ARRAY_BUFFER, vboId[0]);
    // Allocate memory on GPU and copy data on CPU from the array 'vertices'
    glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat)*nb_vertices*3, vertices,
GL_STATIC_DRAW);

    // Allocate and initialise memory of the second vbo storing triangle index
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboId[1]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(GLuint)*nb_faces*3, GL_STATIC_DRAW);

    /*
      Drawing with VBOs
    */

    glEnableClientState (GL_VERTEX_ARRAY);
```

```
    // Tells OpenGl which VBO to use for the vertices when drawing
    glBindBuffer(GL_ARRAY_BUFFER, vboId[0]);
    // Instead of giving a pointer to the main memory we use NULL to use the VBO
    glVertexPointer(3, GL_FLOAT, 0, NULL/* <- We use VBO instead of an array*/);

    // Drawing with VBO of triangle index :
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboId[1]);
    glDrawElements(GL_TRIANGLES, nb_faces*3, GL_UNSIGNED_INT, NULL);

    glDisableClientState (GL_VERTEX_ARRAY);
    // Binding the VBO of identifier 0 tells OpenGL to unbind the VBOs
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    // Deleting the VBOs from OpenGL memory
    glDeleteBuffers(2, vboId);
}
```

One thing to be aware of: VBO of type GL_ARRAY_BUFFER are VBO used to store vertex attributes (position, color, normals, tangents etc.)

VBO of type GL_ELEMENT_ARRAY_BUFFER is used to store face index (triangle index, quad index, line index etc.)

The enum field GL_STATIC_DRAW is an hint to OpenGL to tell how you plane to use the VBO. In this case GL_STATIC_DRAW tells that you will upload once the data and draw it many times. They are other hints, for instance to tell the mesh will be changed many time. You can look at the documentation.

- ✔ Use VBOs instead of arrays to draw the mesh. You will need to add some field to store the VBOs ids in the mesh structure.