LAB 3 (PART 2)

Rendering a quad using Vertex Buffer Object - VBO and Vertex Array Objects - VAO

* All functions are runed from the graphic card, and every time when the object is redrawn - all data are sended to the Graphic processor over and over again. When it comes to simple objects, like those we are drawing, there is no problem, but when we talk about thousands and millions of triangles - things get more complicated.
* The easier way is, to create a buffer in the graphics card and save the data there only once and than read them, where it is needed.

Vertex Buffer Object - VBO and Vertex Array Objects - VAO

* **Vertex Array Objects** - are the modern way of storing and rendering a models in OpenGL. A vertex array objects are the objects in which we can store data about 3D models. They has a lot of slots in which we can store data. This slots are named attributes lists, and can contain different sets of data information like, vertex, textures, lights etc. And there are stored in attribute lists like as vertex buffer objects (VBO).
* Every row of this slots has a unique identificator, so we can access it at any time using its ID.
* **Vertex Buffer Objects** - are just a data, an looks like an array of numbers. This data can be anyting, it coud be positions, collors etc. And each VBO can be put into separate attribute list inn the Vertex Array Objects.
* Every single 3D model is made out of the number of triangles, and each of these triangles has three vertices (three points in three dimensional space). Each point has a 3D coordinate an X, Y and Z. The list of the coordinates of all vertices, define our object in the 3D space. This sort of data we will put into Vertex Buffer Object. And then store into an attribute list of Vertex Array Objects.
* **Create** a new class called **RawModel** - this class going to represent a 3D models stored in memory. There is two things we need to know about a model once it’s stored in memory: the ID of the VAO attribute, and how many vertices are in that model.

**public** **class** RawModel {

**private** **int** vaoID;

**private** **int** vetexCount;

//constructor

**public** RawModel(**int** vaoID, **int** vertexCount) {

**this**.vaoID = vaoID;

**this**.vetexCount = vertexCount;

}

**public** **int** getVaoID() {

**return** vaoID;

}

**public** **int** getVetexCount() {

**return** vetexCount;

}

}

* Create a new class **Loader in package renderEngine**- this class is going to deal with loading 3d models into the memory, by storing positional data about the model in a VAO. Create a method that will take all positions of the models vertices, load this data into a VAO an then return information about the VAO as a RawModel object.

**private** List<Integer> vaos = **new** ArrayList<Integer>();

**private** List<Integer> vbos = **new** ArrayList<Integer>();

**public** RawModel loadToVAO(**float**[] positions) {

**int** vaoID = createVAO();

storeDataInAttributeList(0, positions);

unbindVAO();

**return** **new** RawModel (vaoID, positions.length/3);

}

1. Create method createVAO() to create new empty VAO, that return the unique identifier of this VAO ( vaoID):

**private** **int** createVAO() {

**int** vaoID = GL30.*glGenVertexArrays*();//create new empty VAO

vaos.add(vaoID);//add ID to the list of ID’s

GL30.*glBindVertexArray*(vaoID);//activate VAO by binding it

**return** vaoID;//return unique ID of the created VAO

}

1. Create a method, to story data in the one of the attribute list of the VAO:

**private** **void** storeDataInAttributeList (**int** attributeNumber, **float**[] data) {

**int** vboID = GL15.*glGenBuffers*();//create an empty vbo buffer, to put the data

vbos.add(vboID);//store vbo ID in the vbos list

GL15.*glBindBuffer*(GL15.***GL\_ARRAY\_BUFFER***, vboID);//bind the vbo buffer and specify the type

FloatBuffer buffer = storeDataInFloatBuffer(data);//story data into VBO

GL15.*glBufferData*(GL15.***GL\_ARRAY\_BUFFER***, buffer, GL15.***GL\_STATIC\_DRAW***);

//create the pointer to the buffer with parameters: how many are the attributes(attributeNumber),how many point has every vertex - 3,type float,is there are normalized - false,the distance until the data - 0,begun from - 0.//

GL20.*glVertexAttribPointer*(attributeNumber, 3, GL11.***GL\_FLOAT***, **false**, 0,0);

GL15.*glBindBuffer*(GL15.***GL\_ARRAY\_BUFFER***, 0);//clear buffer

}

**private** FloatBuffer storeDataInFloatBuffer(**float**[] data) {

FloatBuffer buffer =BufferUtils.*createFloatBuffer*(data.length);

buffer.put(data);

buffer.flip();

**return** buffer;

}

**private** **void** unbindVAO() {

GL30.*glBindVertexArray*(0);//fill with 0, to clear

}

* Clear all lists.

**public** **void** cleanUp() {

**for**(**int** vao:vaos) {

GL30.*glDeleteVertexArrays*(vao);

}

**for**(**int** vbo:vbos) {

GL15.*glDeleteBuffers*(vbo);

}

<https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glBindBuffer.xhtml>

* Create a class called **Render** - this class takes the model from the memory and render it.

**package** renderEngine;

**import** org.lwjgl.opengl.GL11;

**import** org.lwjgl.opengl.GL20;

**import** org.lwjgl.opengl.GL30;

**public** **class** Render {

//prepare OpenGL to render every fraim, clearing the buffer from the //last frame

**public** **void** prepare() {

GL11.*glClear*(GL11.***GL\_COLOR\_BUFFER\_BIT***);

GL11.*glClearColor*(0, 0, 0, 1);

}

//receive the RawModel

**public** **void** render(RawModel model) {

//bind Vertex Array with the model

GL30.*glBindVertexArray*(model.getVaoID());

//activate the attribute list

GL20.*glEnableVertexAttribArray*(0);

//Draw, triangles,start from - 0, vertex count - from model.

GL11.*glDrawArrays*(GL11.***GL\_TRIANGLES***, 0, model.getVertexCount());

//disable attribute list

GL20.*glDisableVertexAttribArray*(0);

//unbind Buffer array

GL30.*glBindVertexArray*(0);

}

}

* Main function:

**package** engineTester;

**import** org.lwjgl.opengl.Display;

**import** renderEngine.DisplayManager;

**import** renderEngine.Loader;

**import** renderEngine.RawModel;

**import** renderEngine.Render;

**public** **class** MainGameLoop {

**public** **static** **void** main(String[] args) {

DisplayManager.*createDisplay*();

//create instances of loader and render classes

Loader loader = **new** Loader();

Render render = **new** Render();

**float**[] vertices = {

//1 triangle

-0.5f, 0.5f, 0f,

-0.5f, -0.5f, 0f,

0.5f, -0.5f, 0f,

//2 triangle

0.5f, -0.5f, 0f,

0.5f,0.5f,0f,

-0.5f,0.5f,0f

};

RawModel model = loader.loadToVAO(vertices);

**while**(!Display.*isCloseRequested*()) {

render.prepare();

render.render(model);

DisplayManager.*updateDisplay*();

}

loader.cleanUp(); // do not forget clear!!

DisplayManager.*closeDisplay*();

}

}