**11494 Interactive and GPU Computing** Lab. 5 - 29/04/2015

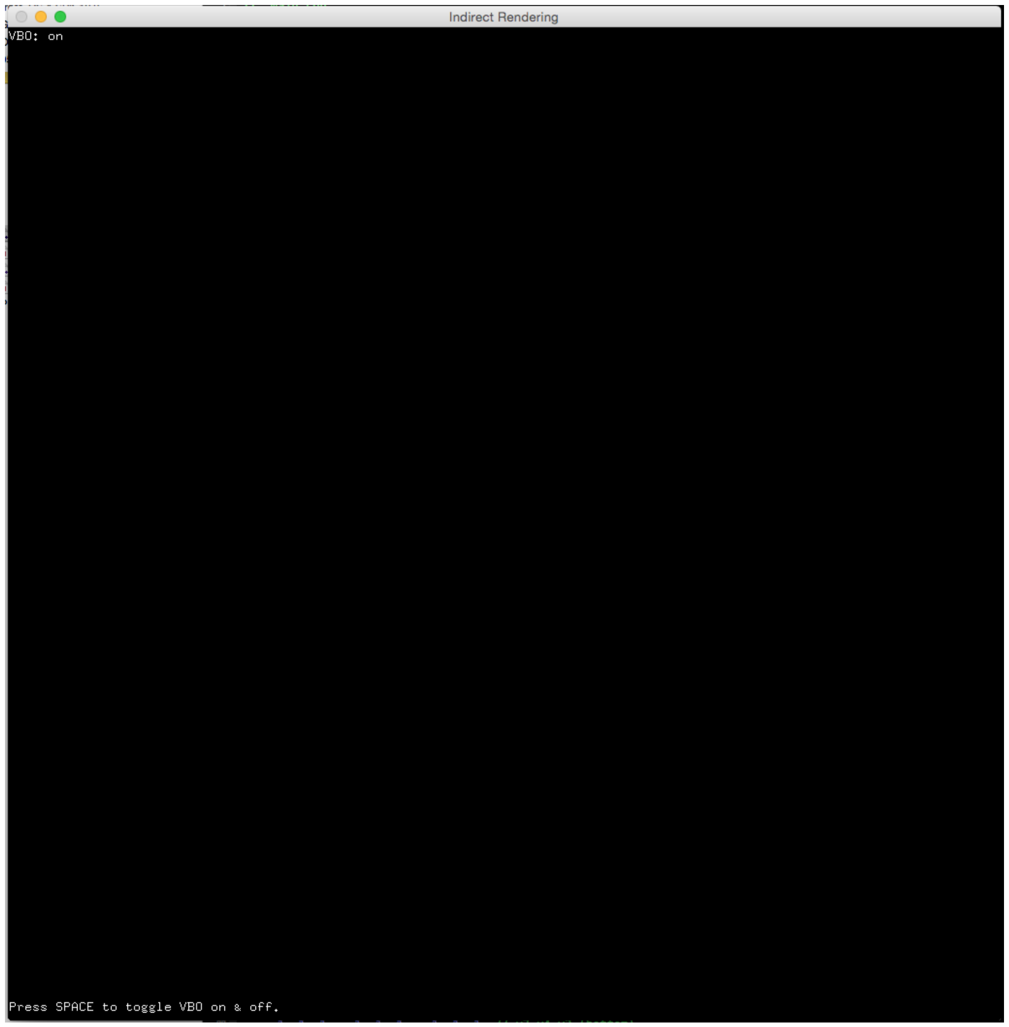
# Indirect Rendering

*Supervisor: Abel Gomes* *Scribe: Orlando Pereira*

The goal of this assignment is to understand how productivity can be achieved using the power of GPU computing.

## 1 Exercises: Indirect Rendering

1. Download the source code for this class, and analyze it carefully. Build and run it. You should have an interface like the following:



```
VBO: on
```

```
Press SPACE to toggle VBO on & off.
```

**Figure 1:** Initial rendering interface.

2. Using direct rendering (glBegin and glEnd instructions) render the cube defined by the *vertices*, *normals* and *colors* arrays. Note that you can use your mouse to interact with the cube. Pressing the *d* key will cycle through several rendering options.

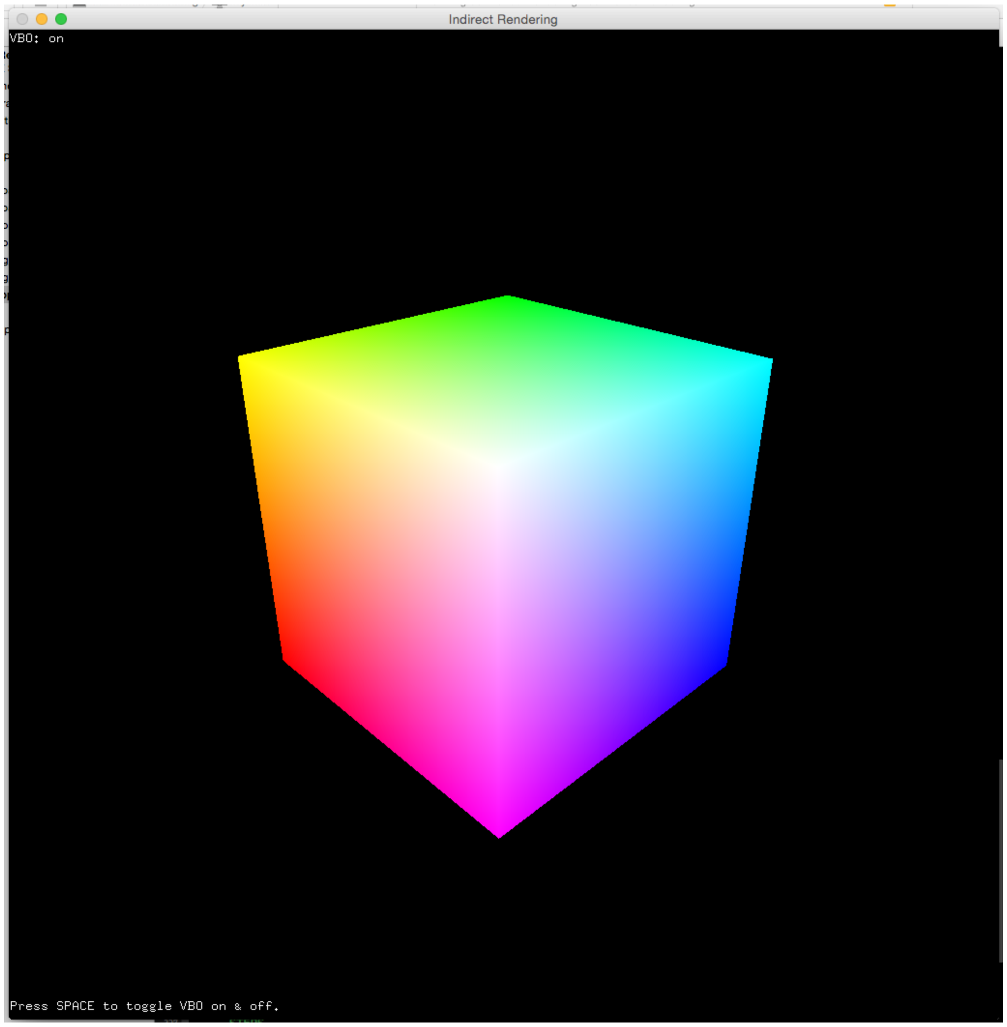


**Figure 2:** Cube rendered using direct rendering.

3. Render the cube using direct rendering through *Vertex Arrays Objects* (VAO). A VAO is an OpenGL object that stores all of the state needed to supply vertex data (vertex, normals, colors, and textures).

   Before progressing any further you are advised to carefully read the following links: *Client-Side Vertex Array Objects* `https://www.opengl.org/wiki/Client-Side_Vertex_Arrays`, and *Vertex Specifications* `https://www.opengl.org/wiki/Vertex_Specification#Vertex_Array_Object`.

   The major steps to achieve VAO rendering are the following:

   (a) Enable the client state;

   (b) Specify the pointers to each array;

   (c) Draw the elements;
   (d) Disable the client state.

   Is there any difference in performance?

4. Create your *Vertex Buffer Object* (VBO). A VBO is an OpenGL feature that provides methods for uploading vertex data (position, normal vector, color, etc.) to the video device for indirect rendering.

   Before progressing any further you are advised to carefully read the following link: *Vertex Buffer Objects Examples* `https://www.opengl.org/wiki/VBO_-_just_examples`.

   The major steps are the following:

   (a) Generate a VBO. Note that, you need to delete it when program exits;;
   (b) Put the vertex, normal and color arrays in the same buffer object;
   (c) Copy data with 3 calls of glBufferSubDataARB, one for vertex coordinates, one for normals vectors, and one for colors.;
   (d) The target flag is *GL_ARRAY_BUFFER_ARB*, and usage flag is *GL_STATIC_DRAW_ARB*. All VBO creation code should be placed inside the following conditional instruction.

```
if(vboSupported) {
    // VBO creation code here
}
```

5. Configure the VBO rendering routine. The major steps to achieve VBO rendering are the following:

   (a) Bind your VBO with an ID and set the buffer offsets of the bound VBOs (When a buffer object is bound with its ID, all pointers in gl*Pointer() are treated as offset instead of real pointer;
   (b) Enable the client state;
   (c) Specify vertex, normals, and color index arrays with their offsets;
   (d) Draw the elements;
   (e) Disable the client state;
   (f) Unbind the VBO. Once bound with 0, all pointers in gl*Pointer() behave as real pointer, so, normal vertex array operations are re-activated.
   Is there any difference in performance?

## 2 Exercises: External 3D Models

1. Import an external .obj file and render it using direct rendering (glBegin and glEnd instructions). The main operations you need to do are the following:

   (a) Initialization:

```
char *newPathToModel = "media/teapot.obj";
objLoader *objData = new objLoader();
```

   (b) Loading the external model:

```
objData->load(newPathToModel);
```

(c) Rendering the model:

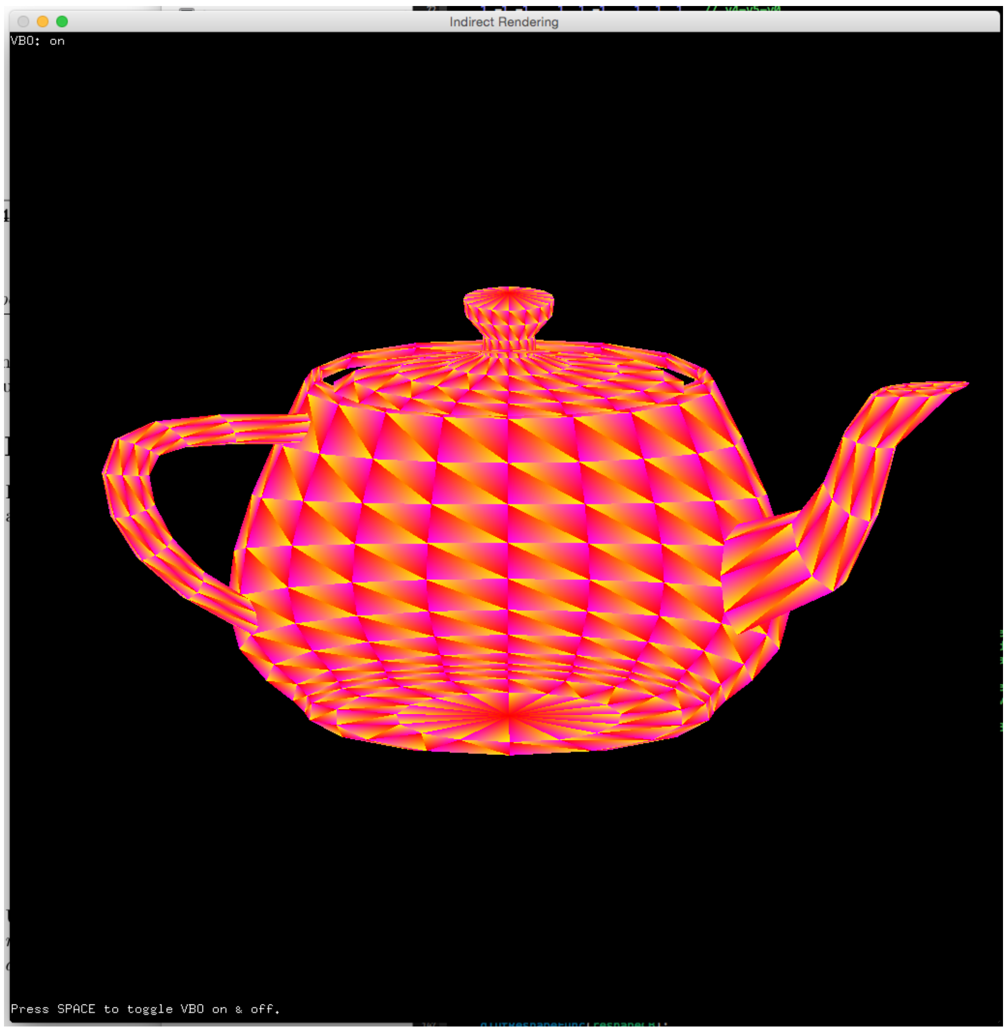
```
for (int i = 0; i < objData->faceCount; i++) {
        glBegin(GL_TRIANGLES);
        obj_face *o = objData->faceList[i];

        glColor3f(1, 0, 0);
        glVertex3f(objData->vertexList[o->vertex_index[0]]->e[0],
            objData->vertexList[o->vertex_index[0]]->e[1],
            objData->vertexList[o->vertex_index[0]]->e[2]);
        glNormal3f(objData->normalList[o->vertex_index[0]]->e[0],
            objData->normalList[o->vertex_index[0]]->e[1],
            objData->normalList[o->vertex_index[0]]->e[2]);

        glColor3f(1, 0, 1);
        glVertex3f(objData->vertexList[o->vertex_index[1]]->e[0],
            objData->vertexList[o->vertex_index[1]]->e[1],
            objData->vertexList[o->vertex_index[1]]->e[2]);
        glNormal3f(objData->normalList[o->vertex_index[1]]->e[0],
            objData->normalList[o->vertex_index[1]]->e[1],
            objData->normalList[o->vertex_index[1]]->e[2]);

        glColor3f(1, 1, 0);
        glVertex3f(objData->vertexList[o->vertex_index[2]]->e[0],
            objData->vertexList[o->vertex_index[2]]->e[1],
            objData->vertexList[o->vertex_index[2]]->e[2]);
        glNormal3f(objData->normalList[o->vertex_index[2]]->e[0],
            objData->normalList[o->vertex_index[2]]->e[1],
            objData->normalList[o->vertex_index[2]]->e[2]);
        glEnd();
    }
```

You should obtain something similar to:


Indirect Rendering

VBO: on

Press SPACE to toggle VBO on & off.

**Figure 3:** Teapot.

2. Modify your rendering step to render your model using VAO.
3. Modify your rendering step to render your model using VBO.
4. Using the *Computer Graphics Data* link `http://graphics.cs.williams.edu/data/meshes.xml` download the Chinese Dragon model and render it using direct and indirect rendering (VBO). Is there any difference in performance? Why?

   You should obtain something similar to:

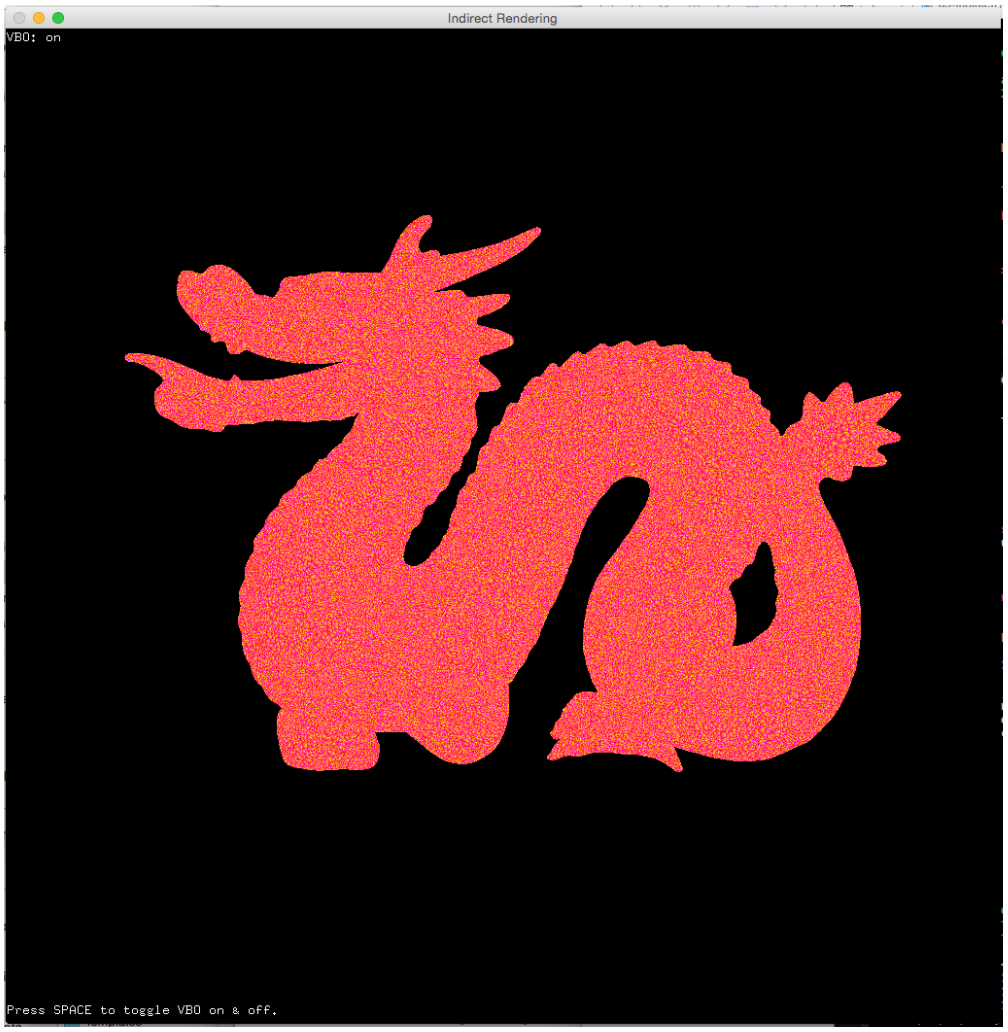


**Figure 4:** Teapot.

## References

[1] Client-Side Vertex Array Objects `https://www.opengl.org/wiki/Client-Side_Vertex_Arrays`, last access on 29/04/2015.

[2] Vertex Specification `https://www.opengl.org/wiki/Vertex_Specification#Vertex_Array_Object`, last access on 29/04/2015.

[3] Vertex Buffer Objects Examples `https://www.opengl.org/wiki/VBO_-_just_examples`, last access on 29/04/2015.

[4] Computer Graphics Data `http://graphics.cs.williams.edu/data/meshes.xml`, last access on 29/04/2015.