



# COMPUTAÇÃO GRÁFICA



LEI / LCC  
DEPARTAMENTO DE INFORMÁTICA  
UNIVERSIDADE DO MINHO

## Performance

*Vertex Buffer Objects*



## VBOs - Initialization

- VBO – Vertex Buffer Object
- Enable Buffer Functionality
  - Do once during initialization of the app

```
glEnableClientState(GL_VERTEX_ARRAY);
```



## Data Preparation

- Allocate and fill arrays with vertices and indices (optional)

```
// array for vertices
float *vertexB;
// fill arrays with vertex values

// array for indices
unsigned int *indices;
// fill arrays with indices
```



# Buffer Initialization

- Generate Vertex Buffer Objects

```
// buffers is a global variable
// n is the number of buffers - one buffer per array
GLuint buffers[n];
...
glGenBuffers(n, buffers);
```

- For Each Buffer *i*

- Set buffer active

```
glBindBuffer(GL_ARRAY_BUFFER, buffers[i]);
```

- Fill buffer

```
glBufferData(GL_ARRAY_BUFFER, arraySize, vertexB, GL_STATIC_DRAW);
```

→ in bytes



# Index Buffer Initialization

- Generate Index Buffer Object

```
// indexes is a global variable
GLuint indexes[1];
...
glGenBuffers(n, indexes);
```

- Set buffer active

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexes[0]);
```

- Fill buffer

```
glBufferData(GL_ELEMENT_ARRAY_BUFFER, arraySize, indices, GL_STATIC_DRAW);
```



## VBOs - Drawing

- Step 1: Set buffer active and define the semantics

```
glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);  
glVertexPointer(3, GL_FLOAT, 0, 0);
```

- Step 2 : Draw VBOs

- With index list

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexes[0]);  
glDrawElements(GL_TRIANGLES, count, GL_UNSIGNED_INT, NULL);
```

- Without index list

```
glDrawArrays(GL_TRIANGLES, first, count);
```

`first` – the starting index

`count` – the number of vertices (not triangles) to draw



# Frames per Second

- Global variables:

```
int timebase;  
float frames;
```

- Code to place in the initialization

```
timebase = glutGet(GLUT_ELAPSED_TIME);
```

– `glutGet(GLUT_ELAPSED_TIME)` returns the number of milliseconds since GLUT has been initialized

- Place the following code in the render scene function

...

```
frame++;  
time = glutGet(GLUT_ELAPSED_TIME);  
if (time - timebase > 1000) {  
    fps = frame*1000.0/(time-timebase);  
    timebase = time;  
    frame = 0;  
}
```

...

- Use function `glutSetWindowTitle(char *s)` to display the fps counter (`sprintf`)

Note: must register render function as idle function to count frames



# Practical Assignment

- Define vertex buffers for the cylinder (without indices)
- Initialization:
  - Create the arrays with the suitable dimension for the vertices of the cylinder
    - $\text{Number of vertices} = \underset{\text{top}}{\text{sides} \times 3} + \underset{\text{body}}{\text{sides} \times 6} + \underset{\text{bottom}}{\text{sides} \times 3}$
    - Each vertex takes three floats
  - Fill the vertex array with the appropriate values to draw the cylinder
  - Generate and enable the VBOs
  - Copy data to OpenGL buffer





## Practical Assignment

- Render:
  - Bind the array
  - Define the semantic for the vertex buffer
  - Use `glDrawArrays` to draw the cylinder
- Compute the FPS values obtained with and without VBOs for several cylinders and fill the grid below:

Mode\sides	4096	16384	65536	262144
Immediate mode				
VBO				



## OpenGL > 1.1 (Win & Linux)

- GLEW – library that facilitates access to OpenGL functionality post version 1.1 (Win)

```
#include <GL/glew.h> // before including glut.h
```

- In the main function (after GLUT's callback registry):

```
glewInit(); // after glutCreateWindow and before calling any OpenGL function
```



## Installing GLEW (Linux)

- Install GLEW
  - `sudo apt-get install libglew-dev`