Lab 2
Vertex Buffer Object

CSE 4431/5331.03M

Advanced Topics in 3D Computer Graphics

TA: Margarita Vinnikov

mvinni@cse.yorku.ca

Vertex Buffer Arrays (VBOs)

- Goals of any 3d application is speed.
- You should always limit the amount of polygons actually rendered
  - by sorting, culling, or level-of-detail algorithms.
  - when all else fails
    - Vertex Arrays are one good way to do that,
    - Vertex Buffer Objects
      - works just like vertex arrays, except that it loads the data into the graphics card's high-performance memory, significantly lowering rendering time
      - The extension being relatively new, not all cards will support it
        - Either use ARB
        - Glew library

Setup Glew and FreeImage

- Download
  - Glew from http://glew.sourceforge.net/
  - Free Image from http://freeimage.sourceforge.net/download.html
- Open your project folder and manually create a new folder named *libraries*
- Copy *GLEW* and *FreeImage* this folder.
- In *VS* now, select *Properties*.
  - From the *Configuration Properties-> C/C++ –> General-> Additional Include Directories*
    - Click on *Additional Include Directories* and select *Edit*.
    - A new menu will pop that will allow you to insert the locations of the header files, press the *New Line* button and browse for the location of *GLEW*, click on the library folder and select *include*.
    - Repeat for *FreeImage*.
  - Now, select *Linker –> General –> Additional Library Directories*, chose *Edit* and add, like before, the location of the binary libraries for *GLEW* and *FreeImage*.
  - Select *Linker –> Input –> Additional Dependencies*, chose *Edit* and add the names of the three binary libraries, one per lines:
    - *glew32.lib*,
    - *FreeImage.lib*
    - Windows *opengl32.lib*.
- You could press the *F7* key or select *BUILD –> Build Solution*.
  - If you've correctly executed the above steps the build should be successful.
- Copy *FreeImage.dll* and *glew32.dll* in the *Debug* folder from your project folder.
- From http://solarianprogrammer.com/2013/05/10/opengl-101-windows-osx-linux-getting-started

Glew (usage)

```
#include <GL/glew.h>
#include <GLUT/glut.h> // not glew should be first
```

In main add the following:

```
glutInit(&argc, argv);
glutCreateWindow("GLEW Test");
GLenum err = glewInit();
if (GLEW_OK != err)
{
  /* Problem: glewInit failed, something is seriously wrong. */
     fprintf(stderr, "Error: %s\n",
     glewGetErrorString(err));
  ...
}
fprintf(stdout, "Status: Using GLEW %s\n",
glewGetString(GLEW_VERSION));
```

- Memory buffer containing geometry data that is managed by the driver
- Providing the benefits of vertex array, while avoiding downsides of their implementations

Vertex array

- Pro: reduces the number of function calls
- Con: redundant usage of the shared vertices
- functions are in the CPU and the data in the arrays must be re-sent to the server each time when it is referenced.

Vertex buffer object

- can be read and updated by mapping the buffer into client's memory space
- Allows vertex array data to be stored in high-performance graphics memory on the server side
- Creates "buffer objects" for vertex attributes
- Provides same access functions to reference the arrays, which are used in vertex arrays,
- such as
  - glVertexPointer(), glNormalPointer(), glTexCoordPointer(), etc.

Declarations

- Same as before

```
//Vertices of a triangle
   float data[] = {1.0, 0.0, 1.0,
                   0.0, 0.0, -1.0,
                   -1.0, 0.0, 1.0};
```

- Plus VBO reference
- `GLuint triangleVBO;`

Creating VBO

- Creating a VBO requires 3 steps (usually in init() after glew initialization.
  1. Generate a new buffer object with **glGenBuffers**().
  2. Bind the buffer object with **glBindBuffer**().
  3. Copy vertex data to the buffer object with **glBufferData**().

Example

Initialization:

```
//Create a new VBO and use the variable id to store the VBO id
  glGenBuffers(1, &triangleVBO);
//Make the new VBO active
  glBindBuffer(GL_ARRAY_BUFFER, triangleVBO);
//Upload vertex data to the video device
  glBufferData(GL_ARRAY_BUFFER, sizeof(data), data, GL_STATIC_DRAW);
```

Clearing the memory:

```
// it is safe to delete after copying data to VBO
  delete [] vertices;
// deactivate VBO active
  glBindBuffer(GL_ARRAY_BUFFER, 0);
// delete VBO when program terminated
  glDeleteBuffers(1, &vboId);
```

glGenBuffers()

```
void
glGenBuffers( GLsizei n,
                  GLuint* ids)
```

- Creates buffer objects and returns the identifiers of the buffer objects.
- 2 parameters:
  - number of buffer objects to create
  - the address of a GLuint variable or array to store a single ID or multiple IDs.

glDeleteBuffers()

- ```
  void
  glDeleteBuffers(GLsizei n,
                  const GLuint* ids)
  ```
- Can delete
  - a single VBO
  - multiple VBOs
- After a buffer object is deleted, its contents will be lost.
  ```

glBindBuffer()

```
void glBindBuffer(GLenum target, GLuint id)
```

- Connects the buffer object with the corresponding ID
- VBO initializes the buffer with a zero-sized memory buffer and set the initial VBO states, such as usage and access properties.

- Takes 2 parameters:
  - Target - tells VBO whether this buffer object will store vertex array data or index array data:
    - GL_ARRAY_BUFFER
      - Any vertex attributes, such as vertex coordinates, texture coordinates, normals and color component arrays
    - GL_ELEMENT_ARRAY_BUFFER
      - Index array which is used for glDraw[Range]Elements()
  - ID – the Id generated by glGenBuffers()

glBufferData()

```
void glBufferData(GLenum target, GLsizei size, const
  void* data, GLenum usage)
```

- Copys the data into the buffer object
- Takes two parameters
  - target would be GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER.
  - Size is the number of bytes of data to transfer.
  - Pointer to the array of source data.
    - If data is NULL pointer, then VBO reserves only memory space with the given data size.
  - Usage flag hints for VBO to provide how the buffer object is going to be used:
    - static, dynamic or stream, and read, copy or draw.

- VBO specifies 9 enumerated values for usage flags;
  - GL_STATIC_DRAW - GL_STATIC_READ - GL_STATIC_COPY
  - GL_DYNAMIC_DRAW - GL_DYNAMIC_READ - GL_DYNAMIC_COPY
  - GL_STREAM_DRAW - GL_STREAM_READ - GL_STREAM_COPY

glBufferSubData()

```
void glBufferSubData(GLenum target, GLint
  offset, GLsizei size, void* data)
```

- Like glBufferData(), glBufferSubData() is used to copy data into VBO, but it only replaces a range of data into *the existing buffer*, starting from the given offset.

Example (Drawing and Updating)

```
// bind VBOs for vertex array and index array
glBindBuffer(GL_ARRAY_BUFFER, vboId);// for vertex coordinates
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboId1);// for indices (Don't
  forget to set it appropriately before hand)
// do same as vertex array except pointer
glEnableClientState(GL_VERTEX_ARRAY);
// activate vertex coords array
glVertexPointer(3, GL_FLOAT, 0, 0);// last param is offset, not ptr
// draw
//Actually draw the triangle, giving the number of vertices provided
glDrawArrays(GL_TRIANGLES, 0, sizeof(data) / sizeof(float) / 3);
glDrawElements(…);

glDisableClientState(GL_VERTEX_ARRAY);
// deactivate vertex array - bind with 0, so, switch back to normal
  pointer operation
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

Updating VBO

- To update VBO
  - Copy new data into the bound VBO with
    - glBufferData() or glBufferSubDataARB()
    - should have a valid vertex array all the time
  - Map the buffer object into client's memory, and the client can update data with the pointer to the mapped buffer.

glMapBufferARB()

```
void* glMapBufferGLenum( target,
  GLenum access)
```

- Maps the buffer object into client's memory.
- If succesful
  - glMapBuffer() returns the pointer to the buffer.
  - Otherwise it returns NULL.
- Takes two parameters
  - *target* is mentioned earlier at glBindBuffer()
  - *access* flag specifies what to do with the mapped data: read, write or both.
    - GL_READ_ONLY_ARB
    - GL_WRITE_ONLY_ARB
    - GL_READ_WRITE_ARB

glUnmapBuffer()

```
GLboolean glUnmapBuffer(GLenum target)
```

- Unmappes the buffer object from the client's memory.
- returns
  - GL_TRUE if success
  - GL_FALSE if the contents of VBO was corrupted while the buffer was mapped.

Example

```
// bind then map the VBO
glBindBufferARB(GL_ARRAY_BUFFER_ARB, vboId);
float* ptr =
  (float*)glMapBufferARB(GL_ARRAY_BUFFER_ARB,GL_WRITE_ONLY_ARB);
// if the pointer is valid(mapped), update VBO
if(ptr)
{
   // modify buffer data
   updateMyVBO(ptr, ...);
  // unmap it after use
   glUnmapBufferARB(GL_ARRAY_BUFFER_ARB);
}
 // you can draw the updated VBO ...
```

First task

VBO

- Note the current Frame Rate for your application
- Modify your code for vertex array to use VBO
- Use glew or ARB extensions (see complementary instruction)
- Note the new Frame Rate for your application

Textures and VBO

Declaration

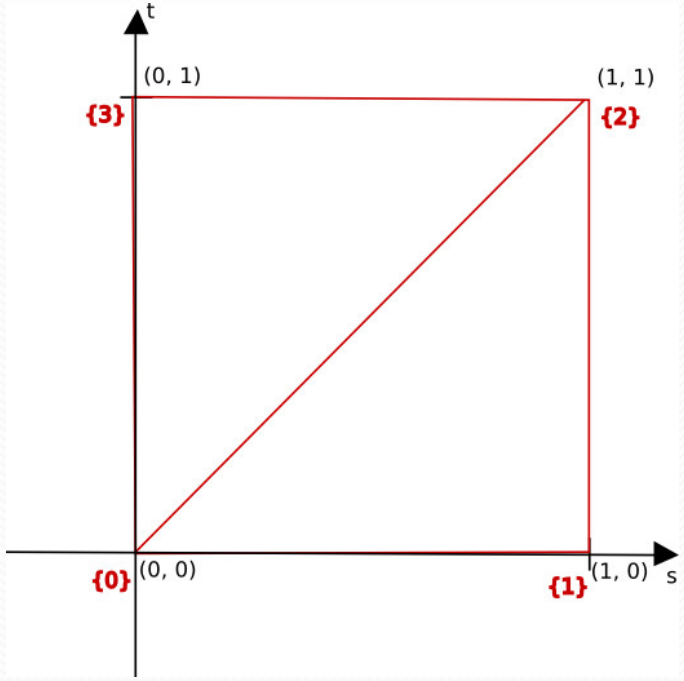
```
GLfloat texture_coord[6] = { 0.0, 0.0,
                             0.0, 1.0,
                             1.0, 1.0};
```

We could store the *texture_coord* in the same *VBO* that stores the *vertex_position* array:

```
glBufferSubData(GL_ARRAY_BUFFER,
sizeof(vertices_position), sizeof(texture_coord),
texture_coord);
```

Also use

```
glTexCoordPointer(2,GL_FLOAT,0,NULL);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
```



Next Task

- Add texture to your figure
- Use FreeImage and provided texture wrapper
- Make sure you use VBO

- Bonus: Add multiple texture, toggle textures, blend textures.

Preparation for next week

Light – on and off

- Add ambient light to the scene
- Add material to your figures
- Allow to toggle the light on and off with the 'L' key

Bonus:

- Rotate light source
- Make fancy light – different colors from different directions

Look for help:

http://www.videotutorialsrock.com/opengl_tutorial/lighting/text.php

http://sjbaker.org/steve/omniv/opengl_lighting.html

Links

http://www.songho.ca/opengl/gl_vbo.html

http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_buffer_object.txt

http://en.wikipedia.org/wiki/Vertex_Buffer_Object

- Write you name and student number in comments at the top of your code
- Submit your entire project code as follows:

  submit 4431 lab2 filename(s)