OpenGL Rendering Modes

Outline

- The Immediate Mode
- Vertex Arrays
- Vertex Buffer Objects

The Immediate Mode

- Vertices data and attributes (the normals, the color...) are sent to the GPU one by one
- One function call for each vertex and for each attribute of a vertex
- Example:

```
glBegin(GL_QUADS);
glNormal3f(1, 0, 0);
glColor3f(1, 0, 1);
glVertex3f(0, 0, 0);
… //set other vertices and their attributes
glEnd();
```

The Immediate Mode

- Pros:
  - Easy to read (if the mesh is small)
- Cons:
  - Very slow when displaying a large mesh that way
  - Deprecated since OpenGL 3.0

Vertex Arrays

- Vertices data and attributes (the normals, the color, the texture coordinates...) are packed into one or several arrays and then sent to the GPU
- Usually one array per vertex attribute, but we can pack several attributes in a single array
- Enabling a vertex array or an attribute array:
  - glEnableClientState(GLenum array_type);
    array_type: GL_VERTEX_ARRAY, GL_NORMAL_ARRAY, GL_COLOR_ARRAY, GL_TEXTURE_COORD_ARRAY or GL_INDEX_ARRAY
  - Tells OpenGL to read the specified attributes or the vertex coordinates or the indices from an array
- Disabling an array:
  - glDisableClientState(GLenum array_type);

Vertex Arrays

- Setting the array that contains the vertices data:
  - glVertexPointer(GLint size, GLenum type, GLsizei stride, const Glvoid* pointer);
    size: the number of coordinates per vertex (2 in 2D, 3 in 3D)
    type: the data type of each coordinate (usually GL_FLOAT)
    stride: the offset between two consecutive vertices (0 if the array contains only the vertices coordinates)
    pointer: a pointer to the 1st element of the array
- Setting the array of other vertex attributes:
  - glNormalPointer(GLenum type, GLsizei stride, const Glvoid* pointer);
  - glTexCoordPointer(GLint size, GLenum type, GLsizei stride, const Glvoid* pointer);
  - glColorPointer(GLint size, GLenum type, GLsizei stride, const Glvoid* pointer);
  - glIndexPointer(GLenum type, GLsizei stride, const Glvoid* pointer);

Vertex Arrays

- Drawing the whole mesh stored in the arrays:
  - glDrawArrays(GLenum mode, GLint first, GLsizei count);
    mode: the type of the primitives to render (GL_TRIANGLES, GL_QUADS, …)
    first: the starting index
    count: the number of indices to be rendered
    - Draw the mesh using the specified arrays
    - Needs at least a call to glVertexPointer(...) and a call to glIndexPointer(...) prior to the call to glDrawArrays(...)
  - glDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid* indices);
    - Same as above, but the index array is specified directly in the function call rather than with glIndexPointer(...)
    - We still a call to needglVertexPointer(...) prior to the the call to glDrawElements(...)

Vertex Arrays

- Pros:
  - Only one function call to draw a whole mesh
  - Possibility to merge vertex coordinates using indices (cf. lesson on meshes)
  - Much faster than immediate mode (only one memory transfer CPU-GPU for each array at each call of glDrawArrays(...))

Vertex Buffer Objects

- When using vertex arrays, the vertices coordinates and attributes are transferred to the GPU each time we call glDrawArrays(...) or glDrawElements(...)
- This memory transfer is useless if vertices data is the same as the previous call...
- A buffer object is an array stored on the GPU memory
- Buffer objects can store the vertex coordinates and attributes of a mesh once and for all

Vertex Buffer Objects

- Buffer objects kinda work like texture objects in OpenGL

- Creating a buffer object:
  - glGenBuffers(GLsizei n, GLuint* buffer_ids);
    generates n new buffer IDs that are stored in buffer_ids

- Setting the current buffer object to work on:
  - glBindBuffer(GLenum target, GLuint buffer_id);
    target: specifies the data type of the buffer object:
    GL_ARRAY_BUFFER for float values (ex: vertex coordinates)
    GL_ELEMENT_ARRAY_BUFFER for integer values (ex: indices)
    buffer_id: the ID of a previously generated buffer object

Vertex Buffer Objects

- Allocating and initializing a buffer object:
  - glBufferData(GLenum target, GLsizei size, const GLvoid* data, GLenum usage);

    target: either GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER

    size: the size of the buffer data, in bytes

    data: pointer to the input array

    usage: the expected usage of the buffer object:
    STREAM_DRAW: the buffer will be set once and used a few times
    STATIC_DRAW: the buffer will be set once and used many times
    DYNAMIC_DRAW: the buffer will be modified repeatedly and used many times

Vertex Buffer Objects

- Telling OpenGL to use a buffer object rather than an array:
  - Just set the value of the array pointers to zero after binding to the right buffer object

- Example:

```
glBindBuffer(GL_ARRAY_BUFFER, my_vbo);
glVertexPointer(3, GL_FLOAT, 0, 0);
```

Tutorial

- In this tutorial we will draw a regular polygon using vertex arrays and VBOs
- A regular polygon of N edges is composed of N triangles that share a common edge (the center of the polygon)
- Compute the vertices coordinates of a regular polygon centered at (0,0) and store them in an array. Don't forget the center vertex. You should store (N+1)*2 coordinates
- Compute the index array of the triangles that form the polygon. You should store N*3 indices
- Draw that polygon in OpenGL using vertex arrays
- Draw that polygon in OpenGL using vertex buffer objects
- Create another attribute arrays (color, or texcoord) and use it to modify the attributes of the vertices