February 20, 2025, Brno
Author: David Procházka

Geometric Primitives

Graphic Application Development

MENDELU
Faculty
of Business
and Economics

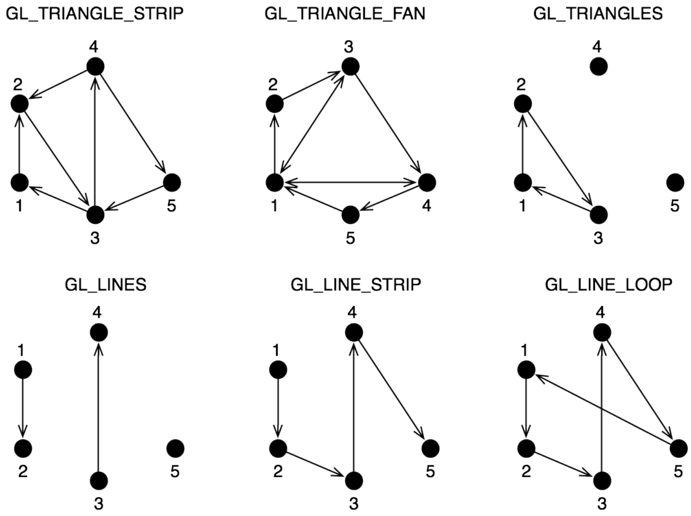
Table of content

Geometric primitives

All objects are stored as sets of vertices. The final shape is just representation of these vertices.
Basic primitives: `GL_POINTS`, `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`, `GL_TRIANGLES`, `GL_TRIANGLE_FAN`, `GL_TRIANGLE_STRIP`, `GL_QUADS`, `GL_POLYGON`, `GL_QUAD_STRIP`[1].

[1] from OpenGL 3.0, these primitives are deprecated

Geometric primitives



Further geometric primitives

- Further, there are primitives with adjacent vertices:
    - `LINES_ADJACENCY`,
    - `LINE_STRIP_ADJACENCY`,
    - `TRIANGLES_ADJACENCY`,
    - `TRIANGLE_STRIP_ADJACENCY`.
- These primitives are used by shaders. (Will be explained later.)

Table of content

Vertex Buffers

- *Vertex buffers* are an alternative to definition of the vertices by separate commands (which is insanely ineffective).
- We must implement following steps:
  1. Enabling of the array(s) – beside the vertex array, we can define color array, texture array etc.
  2. Setting pointers on the data in the arrays
  3. Drawing of the geometric primitives
  4. Disabling of the array(s)

Enabling and disabling of an array

Array is enable by following command:

- `glEnableClientState(GL_..._ARRAY)` a
- `glDisableClientState(GL_..._ARRAY)`.

Following arrays are available:

- `GL_VERTEX_ARRAY` – coordinates,
- `GL_COLOR_ARRAY` – colors,
- `GL_SECONDARY_COLOR_ARRAY` – secondary colors,
- `GL_INDEX_ARRAY` – deprecated,
- `GL_NORMAL_ARRAY` – normals for lighting,
- `GL_FOG_COORDINATE_ARRAY` – fog,
- `GL_TEXTURE_COORD_ARRAY` – texture coordinates,
- `GL_EDGE_FLAG_ARRAY` – is edge visible?

Pointer for reading

- Let us assume that the triangle is given by vertices with two `GLint` coordinates (x,y). The array will be: $\{x_1, y_1, x_2, y_2, x_3, y_3\}$, where x_a, y_a are `GLint` values.
- In C++, we can define it as:

```
GLint vertices[] = {10, 10, 100, 300, 200, 10};
```

- Further, we must specify the array structure:
  - `glVertexPointer()` (coordinate definition),
  - `glColorPointer()` (color definition).

Vertex pointers

Color pointer command

```
glVertexPointer(GLint size, GLenum type, GLsizei
stride, const GLvoid *pointers)
```

- `size` number of coords (2, 3, 4),
- `type` type of coords (GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE),
- `stride` distance between coords (explained later)
- `pointers` name of the variable with the coords.

Examples:

- `glVertexPointer(2, GL_INT, 0, vertices);`
- `glVertexPointer(2, GL_FLOAT, 5*sizeof(GL_FLOAT), &triangle[0]);`

Color pointers

Color pointer command

```
glColorPointer(GLint size, GLenum type, GLsizei
stride, cont GLvoid *pointers)
```

- Parameters are similar to vertex pointer.
- `size` is number of channels (3 or 4 – with alpha),
- `type` type of color channels (`GL_BYTE`, `GL_UNSIGNED_BYTE`, `GL_UNSIGNED_SHORT`, `GL_UNSIGNED_INT`).

Independent arrays with values

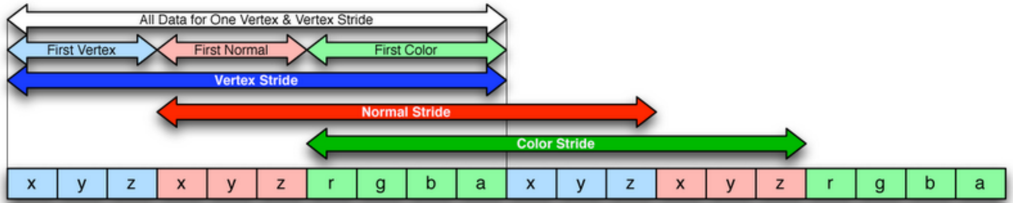
```
glClear(GL_COLOR_BUFFER_BIT);
GLint vertices[] = // three tuples of coords
  {10, 10, 100, 300, 200, 10};
GLfloat colors[] = // three triples of coords
  {1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0};
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glVertexPointer(2, GL_INT, 0, vertices);
glColorPointer(3, GL_FLOAT, 0, colors);
// drawing
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
glFlush();
```

Table of content

Interleaved arrays – *stride* and *pointer*

- `Stride` parameter is zero in case of independent arrays.
- It has non-zero value in case of interleaved arrays where in a single arrays are different values (e.g. coords nad colors).
- The value is a distace between first bytes of the values of the same meaning (e.g. two colors).
- Example: Both coords and colors are `GL_FLOAT` values. The stride will be `5*sizeof(GL_FLOAT)` (2 coords + 3 color channels).
- The last parameter is `pointer` where the values begin.
- Example: in case of array $x_1, y_1, R_1, G_1, B_1, \ldots$ the `pointer` will be `&data[0]` for coords and `&data[2] for colors`.

Interleaved arrays


All Data for One Vertex & Vertex Stride
First Vertex
First Normal
First Color
Vertex Stride
Normal Stride
Color Stride
x y z x y z r g b a x y z x y z r g b a

Interleaved arrays

```
glClear (GL_COLOR_BUFFER_BIT);
static GLfloat triangle[] = {
    10.0,  10.0, 1.0, 0.0, 0.0, // 2 coords, 3 colors
   100.0, 300.0, 0.0, 0.0, 1.0,
   200.0,  10.0, 0.0, 1.0, 0.0};
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glVertexPointer(2, GL_FLOAT, 5*sizeof(GL_FLOAT), &triangle[0]);
glColorPointer(3, GL_FLOAT, 5*sizeof(GL_FLOAT), &triangle[2]);
// drawing
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
glFlush();
```

Table of content

Drawing using indices

Command for drawing using indices

```
glDrawElements(GLenum mode, Glsizei count, GLenum
type, void* indices)
```

- `mode` – selected primitive (e.g. `GL_TRINAGLES`),
- `count` – amount of indices in the array,
- `type` – type of indices (e.g. `GL_INT`),
- `indices` – pointer on **indices array**.

Drawing using indices: example

```
glClear (GL_COLOR_BUFFER_BIT);
GLfloat triangle[] = {
  10.0, 10.0, 1.0, 0.0, 0.0,
  100.0, 300.0, 0.0, 0.0, 1.0,
  200.0, 10.0, 0.0, 1.0, 0.0};
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glVertexPointer(2, GL_FLOAT, 5*sizeof(GL_FLOAT), &triangle[0]);
glColorPointer(3, GL_FLOAT, 5*sizeof(GL_FLOAT), &triangle[2]);
static GLubyte indices[]={0,1,2}; // definice pole indexu
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, indices);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
glFlush();
```

Extension of the `glDrawElements()`

Command for drawing using multiple arrays of indices

```
glMultiDrawElements(GLenum mode, GLsizei∗ count,
GLenum type, void∗∗ indices, GLsizei primcount)
```

- count – array with amounts of indices in arrays,
- indices – array of arrays of indices,
- primcount – number of arrays of indices.

```
GLubyte firstIndices[] = {0, 1, 2, 3};
GLubyte secondIndices[] = {2, 4, 7, 8};
GLsizei count[] = {4, 4};
GLvoid∗ indices[2] = {firstIndices, secondIndices};
glMultiDrawElements(GL_LINES,count,GL_UNSIGNED_BYTE,indices,2);
```

Drawing using vertices

- `glDrawArrays(GLenum type, GLint first, GLsizei count)`,
- `glMutiDrawArrays(GLenum type, GLint* first, GLsizei* count, GLsizei primcount)`
- It works directly with vertices, not indices
- Draws from the `first` vertex to `first` + `count` vertex.

Drawing using vertices:

```
...
glVertexPointer(2, GL_FLOAT, 5*sizeof(GL_FLOAT),&triangles[0]);
glColorPointer(3, GL_FLOAT, 5*sizeof(GL_FLOAT),&triangles[2]);

glDrawArrays(GL_TRIANGLES, 0, 6);
...
```

Comparison of the methods

Methods for drawing of primitives:

- `glDrawElements` – works with indices,
- `glMultiDrawElements` – works with arrays of indices,
- `glDrawRangeElements` – works with ranges of indices,
- `glDrawArrays` – works with vertices,
- `glMutiDrawArrays` – works with arrays of vertices,

Table of content

Takeaway

- We must explain the OpenGL what values will be send (enabling).
- Pointers define where the values are.
- Drawing methods take indexes or vertices and renders required primitive.