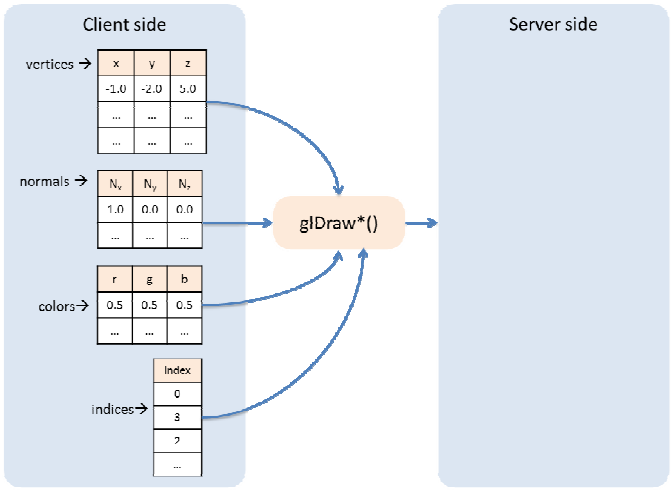
VERTEX ARRAYS

Avantatges:

- Redueix el nombre de crides a funcions OpenGL.
- Evita el processament redundant de vèrtexs compartits per més d'una primitiva

Limitacions:

- Els atributs (normals, colors, coordenades de textura, etc) s'han d'especificar **per vèrtex**, no per cara. Si un mateix vèrtex té atributs diferents per cada cara que el comparteix, una possible solució és duplicar el vèrtex juntament amb els seus atributs.
- Les dades romanen en el client; per tant, les dades s'han d'enviar al servidor cada frame (a diferència dels Vertex Buffer Objects) .



PAS 1. ESPECIFICAR LES DADES DELS ARRAYS

Aquestes funcions proporcionen a OpenGL un apuntador a les dades, així com número de coordenades o components per atribut (normalment 3 ó 4) i el tipus. No hi ha cap còpia de les dades; en el moment de pintat OpenGL utilitzarà directament arrays.

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

- **size**: número de coordenades per cada vèrtex. Valors possibles: 2,3,4.
- **type**: tipus de cada coordenada. Habitualment serà GL_FLOAT o GL_DOUBLE.
- **stride**: offset (mesurat en bytes) entre l'inici de les coordenades de dos vèrtexs consecutius. Si cada atribut està en un array independent, és 0.
- **pointer**: apuntador al array de vèrtexs.

Anàlogament, hi ha crides per especificar les normals, les coordenades de textura, els colors, etc.:

glNormalPointer(GLenum **type**, GLsizei **stride**, const GLvoid* **normals**)
glColorPointer(GLint **size**, GLenum **type**, GLsizei **stride**, const GLvoid* **colors**)
glTexCoordPointer(GLint **size**, GLenum **type**, GLsizei **stride**, const GLvoid* **texcoords**)
...

L'array d'índexs (si es fa servir **glDrawElements**) s'especifica directament a la crida de pintat. **No feu servir glIndexPointer**(), que és per una altra cosa...

PAS 2. ACTIVAR ELS ARRAYS

Aquestes crides determinen quins arrays s'utilitzaran quan es cridi a les funcions de pintat glArrayElement(), glDraw*(). Com a mínim caldrà activar l'array de vèrtexs.

glEnableClientState(GL_VERTEX_ARRAY); // activa array de coordenades dels vèrtexs
glEnableClientState(GL_NORMAL_ARRAY); // activa array amb components de normals
glEnableClientState(GL_TEXTURE_COORD_ARRAY); // activa array amb coordenades de textura
glEnableClientState(GL_COLOR_ARRAY); // activa array de colors RGB/RGBA
~~**glEnableClientState**(GL_INDEX_ARRAY);~~ // activa array de colors mode color-index (**obsolet**)
...

PAS 3. RENDERING

Hi ha diferents opcions. Ordenades de menys a més eficient:

OPCIÓ 1. UNA CRIDA PER VÈRTEX

glArrayElement(GLint index)

Exemple:

glBegin(GL_TRIANGLES);
glArrayElement(2);
glArrayElement(5);
glArrayElement(3);
glEnd();

En el cas d'un cub, els arrays tindrien 8 elements (8 vèrtexs), sense cap ordre específic.

	x	y	z
0	0.0	0.0	0.0
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	0.0	1.0	1.0
4	1.0	0.0	0.0
5	1.0	0.0	1.0
6	1.0	1.0	0.0
7	1.0	1.0	1.0

OPCIÓ 2. UNA CRIDA PER TOTES LES PRIMITIVES, SENSE ÍNDEXS

glDrawArrays(GLenum mode, GLint first, GLsizei count)

- **mode**: GL_TRIANGLES, GL_QUADS,... (qualsevol dels modes vàlids per glBegin)
- **first**: primer element del array a utilitzar. Normalment serà 0.
- **count**: número d'elements del array a utilitzar. Normalment serà la mida del array.

Aquesta opció **requereix que els vèrtexs estiguin ordenats formant primitives**. Per exemple, si es fa servir GL_TRIANGLES, tres vèrtexs consecutius de l'array han de definir un triangle. El mateix s'aplica a la resta de primitives que requereixen més d'un vèrtex.

En el cas d'un cub, els arrays tindrien 24 ó 36 elements, depenent de si les cares s'envien com QUADS (6 cares *4 vèrtexs/cara) o TRIANGLES (12 cares *3 vèrtexs/cara) .

	x	y	z
0	0.0	0.0	0.0
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	0.0	1.0	1.0
...			
23			

Usant GL_QUADS, 4 vèrtexs consecutius han de definir una cara

OPCIÓ 3. UNA CRIDA PER TOTES LES PRIMITIVES, AMB ÍNDEXS

glDrawElements(GLenum mode, GLsizei count, GLenum type, void* indices)

- **mode**: GL_TRIANGLES, GL_QUADS,... (qualsevol dels modes vàlids per glBegin)
- **count**: mida de l'array de índexos. Exemples:
  - Un cub enviat amb GL_TRIANGLES tindrà 12 cares * 3 índexs/cara = 36 índexs
  - Un cub enviat amb GL_QUADS tindrà 6 cares * 4 índexs /cara = 24 índexs
- **type**: tipus dels índexs. Habitualment GL_UNSIGNED_INT
- **indices**: apuntador al array amb els índexs

Exemple:

glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, indices)

equival a:

```
glBegin(GL_TRIANGLES);
for (i=0; i<count; ++i)
  glArrayElement(indices[i]);
glEnd();
```

PAS 4. DESACTIVAR ELS ARRAYS

És convenient desactivar els arrays que s'hagin fet servir; una crida posterior a glDraw*() podria usar per exemple normals però no colors.

```
glDisableClientState(GL_VERTEX_ARRAY);        // desactiva array de coordenades dels vèrtexs
glDisableClientState(GL_NORMAL_ARRAY);        // desactiva array amb components de normals
...
```

RESUM OPCIONS DE PINTAT AMB VERTEX ARRAYS

Nomenclatura taula:

- T = número de triangles
- V = numero de vèrtexs
- A = número d'atributs per vèrtex (ex. 3: coords vèrtex, color, normal)

Mode pintat	Crides OpenGL	Mida dels arrays	Ús d'índexos	Tots els vèrtexs duplicats	Estalvi crides OpenGL	Pot evitar processament redundant de vèrtexs
Mode inmediat	T*3*A	-	-	-	-	No
glArrayElement	T*3	V	Un índex a cada crida.	No	Petit	Si
glDrawArrays	1	3*V	No; els vèrtexs han d'estar ordenats formant primitives	Sí	Gran	No
glDrawElements	1	V	Un array d'índexos	No	Gran	Si

VERTEX BUFFER OBJECTS (VBO)

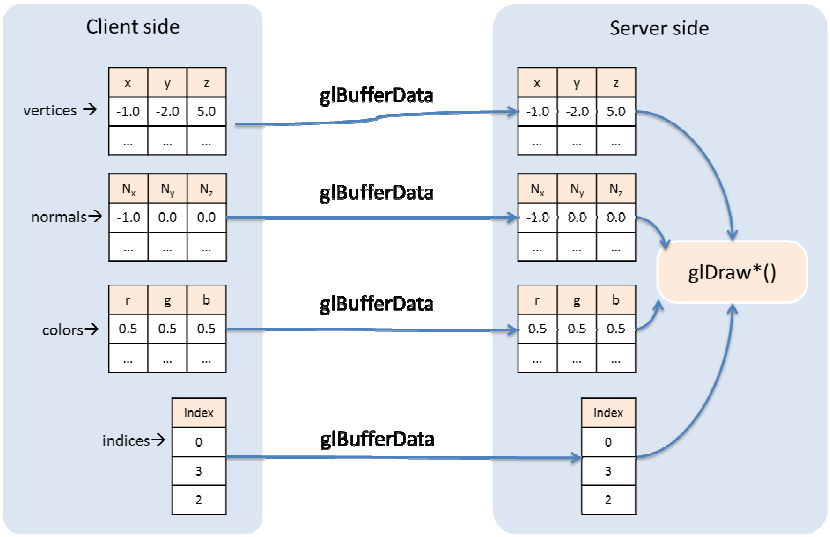
Un VBO és bàsicament un vertex array que s'emmagatzema al servidor OpenGL, no al client.

Avantatges:

- Els vèrtexs (i els seus atributs) només s'envien al servidor un cop (quan es crea el VBO) en comptes de cada frame.

Limitacions:

- El servidor ha de tenir prou memòria per emmagatzemar les dades.
- L'actualització dinàmica de les dades (per exemple canviar dinàmicament les coordenades dels vèrtexs) és més complicada.



PAS 1. INICIALITZACIÓ

GENERAR IDENTIFICADORS (UN PER CADA ARRAY)

Cada array (coordenades, colors, normals, índexs...) és un VBO que s'identifica amb un GLuint. Aquests identificadors ens permetran dibuixar múltiples VBOs més còmodament.

```
GLuint ids[NUM];
glGenBuffers(NUM, ids);
```

ESPECIFICAR LES DADES DE CADA BUFFER

Primer cal "activar" un buffer amb **glBindBuffer**(), donant el seu identificador. Després es copien les dades del client al servidor amb **glBufferData()**.

Pels atributs (vertices, normals, colors...) es fa servir el target GL_ARRAY_BUFFER:

```
glBindBuffer(GL_ARRAY_BUFFER, id)                          // activa el buffer id
glBufferData(GL_ARRAY_BUFFER, sizeof(data), data, usage)   // es copia l'array del client al servidor.
```

- **data** és un apuntador a l'array (vertices, normals, colors...)
- **usage** és habitualment GL_STATIC_DRAW, encara que hi ha altres possibilitats.

Després de la crida a glBufferData, es pot alliberar la memòria del buffer en el client.

Pels índexs, si es fan servir, es fa servir el target GL_ELEMENT_ARRAY_BUFFER:

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, **id**) // activa un buffer d'índexs
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(**indices**), **indices**, **usage**)

PAS 2. CADA FRAME

INDICAR, PER CADA ATRIBUT, EL TIPUS I L'OFFSET DE LES DADES

glBindBuffer(GL_ARRAY_BUFFER, id) // activa el buffer corresponent
glVertexPointer(3, GL_FLOAT, 0, (GLvoid*) **0**); // idem amb glNormalPointer, glColorPointer,...
glEnableClientState(GL_VERTEX_ARRAY); // indica que es farà servir a glDrawElements

Les tres crides anteriors s'han de repetir per cada atribut (si estan en arrays independents).

RENDERING PRÒPIAMENT DIT

Podem fer servir el mateix glDrawElements dels VAs:

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, id) // activa el buffer d'indexs
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, (GLvoid*) **0**)

IMMEDIATAMENT DESPRÉS DEL RENDERING

glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
...
glBindBuffer(GL_ARRAY_BUFFER,0);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,0);

PAS 3. AL ACABAR, CAL ESBORRAR EL VBO

glDeleteBuffers(NUM, ids)