Sintaxis de los comandos de OpenGL

- Todos los comandos OpenGL comienzan con **gl**, y cada una de las palabras que componen el comando comienzan por letra mayúscula.

```
glMatrixMode(....)
glEnable(...)
```

- Las constantes (y variables de estado) en OpenGL se escriben en mayúsculas, y todas ellas empiezan por GL. Cada una de las palabras que componen la constante está separada de la anterior por "_".

```
GL_MODELVIEW
GL_COLOR_BUFFER_BIT
```

Sintaxis de los comandos de OpenGL

- Existen comandos en OpenGL que admiten distinto número de argumentos y distintos tipos. Estos comandos terminan con el sufijo **nt**, donde **n** indica el número de argumentos y **t** el tipo de los mismos.

```
glVertex2i(10,21);
```

indica que estamos usando vértices 2D de tipo entero. Por su parte

```
glVertex3f(10.0,21.1,11.7);
```

indica que estamos usando vértices 3D del tipo GLfloat.

Tipos básicos en OpenGL

- OpenGL trabaja internamente con tipos básicos específicos que son compatibles con los de C/C++.
- Conviene utilizar los tipos de OpenGL para garantizar la "**independencia del sistema**" en el que estamos ejecutando el programa.
- Por ejemplo, el método:

```
void drawDot(int x, int y) {
   glBegin(GL_POINTS);
      glVertex2i(x,y);
   glEnd();
}
```

tendría que transformar el **int** en **GLint**

Tipos básicos en OpenGL

- El programa anterior podría dar problemas en sistemas que empleen un número distinto de bits para almacenar un **int** y un **GLint**.
- La forma idónea de definir el método anterior sería:

```
void drawDot(GLint x, GLint y) {
   glBegin(GL_POINTS);
      glVertex2i(x,y);
   glEnd();
}
```

Tipos básicos en OpenGL

Sufijo	Tipo	OpenGL
b	entero de 8 bits	GLbyte
s	entero de 16 bits	GLshort
i	entero de 32 bits	GLint, GLsizei
ub	entero sin signo de 8 bits	GLubyte, GLboolean
us	entero sin signo de 16 bits	GLushort
ui	entero sin signo de 32 bits	GLuint, GLenum, GLbitfield
f	coma flotante 32 bits	GLfloat, GLclampf
d	coma flotante 64 bits	GLdouble, Glclampd

Primitivas Gráficas: puntos

- Para dibujar un punto tenemos los comandos:
  - ✓ **`glVertex2{sifd}[v](x,y);`** Especifica un vértice 2D en las coordenadas (x,y).
  - ✓ **`glVertex3{sifd}[v](x,y,z);`** Especifica un vértice 3D en las coordenadas (x,y,z).

- Para dibujar una secuencia de puntos:

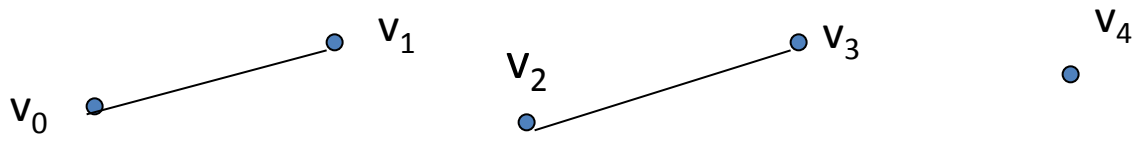
```
glBegin(GL_POINTS);
  glVertex..
  glVertex..
  ......
  glVertex..
glEnd();
```

Primitivas gráficas: segmentos

- Para dibujar líneas debemos usar:

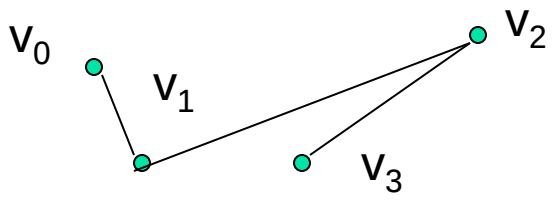
```
glBegin(GL_LINES);
  glVertex..  // v0
  glVertex..  // v1
  ......
  glVertex..  // vn
glEnd();
```

Dibuja los segmentos $\mathbf{v_0v_1}$, $\mathbf{v_2v_3}$, ..., $\mathbf{v_{n-1}v_n}$. Si el número de vértices es impar, el último vértice introducido se ignora.

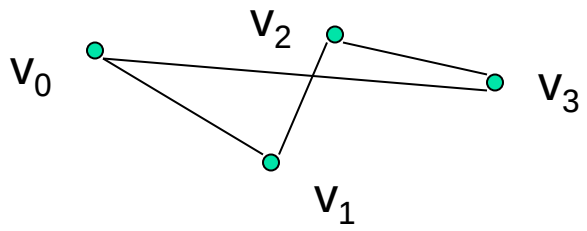


Primitivas gráficas: segmentos

- Si utilizamos la constante **`GL_LINE_STRIP`**, los segmentos dibujados se conectan: se pintan las líneas $\mathbf{v_0v_1}$, $\mathbf{v_1v_2}$, $\mathbf{v_3v_4}$, etc. Si el número de vértices es 1, entonces no se pinta nada.



- Con la constante **`GL_LINE_LOOP`** la poli-línea se cierra: se dibujan las líneas $\mathbf{v_0v_1}$, $\mathbf{v_1v_2}$,…, $\mathbf{v_{n-1}v_n}$, $\mathbf{v_nv_0}$.



Primitivas gráficas: triángulos

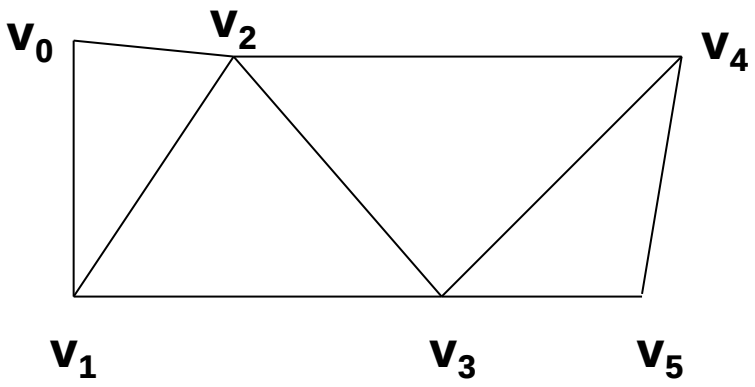
- Para dibujar triángulos debemos usar:

```
glBegin(GL_TRIANGLES);
  glVertex.. // v0
  glVertex.. // v1
  ......
  glVertex.. // vn
glEnd();
```

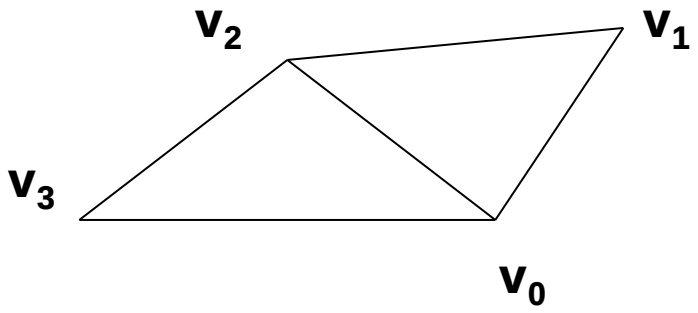
- Dibuja los triángulos $v_0v_1v_2$, $v_3v_4v_5$, ... Si el número de vértices no es múltiplo de 3, el último o los dos últimos vértices se ignoran.
- Todos los triángulos dibujados salen rellenos del color que esté activado.

Primitivas gráficas: triángulos

- Con **GL_TRIANGLE_STRIP**: se dibuja los triángulos $v_0v_1v_2$, $v_2v_1v_3$, $v_2v_3v_4$, ... El número de vértices ha de ser al menos 3.
- Con **GL_TRIANGLE_FAN** se dibujan los triángulos $v_0v_1v_2$, $v_0v_2v_3$, $v_0v_3v_4$, ... Todos los triángulos comparten el vértice v_0.
- En ambos casos el número de vértices ha de ser al menos 3.



STRIP



FAN

Primitivas gráficas: cuadriláteros

- Para dibujar cuadriláteros se usa:

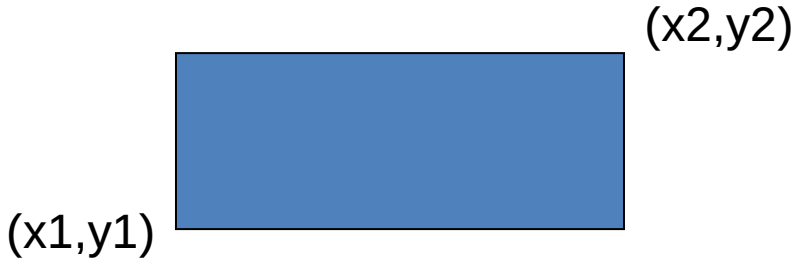
```
glBegin(GL_QUADS);
  glVertex..      // v0
  glVertex..      // v1
  ......
  glVertex..      // vn
glEnd();
```

- Dibuja los cuadriláteros $v_0v_1v_2v_3$, $v_4v_5v_6v_7$, ... Si el número de vértices no es múltiplo de 4, se pueden descartar hasta los tres últimos vértices.
- Con **GL_QUAD_STRIP** se conecta cada cuadrilátero con el siguiente. Si el número de vértices es impar, el vértice final se ignora.

Primitivas gráficas: polígonos

- Debemos utilizar la constante **`GL_POLYGON`**. Sólo pinta un polígono, que tiene tantos lados como vértices estén especificados entre `glBegin(..)` y `glEnd()`.
- OpenGL **sólo garantiza un funcionamiento adecuado para el caso de polígonos convexos**. La constante **`GL_POLYGON`** sólo debe utilizarse para la creación de este tipo de polígonos. En otro caso el comportamiento de OpenGL es impredecible.
- Conviene enumerar los vértices en sentido antihorario, pues así estaremos modelando la cara frontal del polígono.

Primitivas gráficas: rectángulos alineados



- Como los rectángulos alineados son un tipo especial de polígonos, podemos dibujarlos usando **GL_POLYGON**:

```
glBegin(GL_POLYGON);
   glVertex2?(x1,y1); glVertex?(x2,y1);
   glVertex2?(x2,y2); glVertex?(x1,y2);
glEnd();
```

- También podemos conseguirlos mediante el comando:

```
glRect{sifd}(type x1, type y1, type x2, type y2)
```

Primitivas gráficas

- La omisión de **`glEnd()`** no genera error de compilación, pero impide que se dibuje el gráfico. En general hay que tener cuidado con los errores de sintaxis, porque muchos de ellos no son detectados por el compilador.

- Entre **`glBegin(..)`** y **`glEnd()`** puede aparecer información sobre el color, textura, iluminación, etc, de la figura. También pueden aparecer comandos C++, por ejemplo una instrucción **`for`**.

- La enumeración de los vértices de un polígono conviene realizarla en el sentido contrario de las agujas del reloj para modelar las caras frontales.

- Todos los polígonos aparecen rellenos del color que esté activo.

Variables de estado de OpenGL

- Para definir colores usaremos el modelo RGB. Los valores que usaremos para especificar un color estarán normalizados en el intervalo [0, 1].

- El color activo se establece con

```
glColor3f(GLfloat r, GLfloat g, GLfloat b)
```

- El color de fondo se establece con

```
glClearColor(GLclampf r, GLclampf g,
             GLclampf b, GLclampf alfa)
```

El último argumento configura el grado de transparencia. Usaremos 1.0 para expresar que el fondo no es transparente.

- El color de fondo se aplica a la ventana con

```
glClear(GL_COLOR_BUFFER_BIT)
```

Variables de estado de OpenGL

- El tamaño con que se dibujan los puntos se establece con

```
glPointSize(GLfloat size)
```

- El grosor de las líneas se establece con

```
glLineWidth(Glfloat w)
```

- La forma en que se representan las regiones rellenas se establece con

```
glPolygonMode(GLenum face,GLenum mode)
```

  - **`face`** puede ser: `GL_FRONT, GL_BACK, GL_FRONT_AND_BACK`
  - **`mode`** puede ser: `GL_POINT, GL_LINE, GL_FILL`.

- La configuración de estas variables afecta a cualquier primitiva cuyo `glBegin()` aparezca más tarde.