UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Ejemplos de aplicaciones 3D interactivas con OpenGL


| | |
|---|---|
| **Apellidos, nombre** | Agustí i Melchor, Manuel (magusti@disca.upv.es) |
| **Departamento** | Departamento de Informática de Sistemas y Computadores (DISCA) |
| **Centro** | Escola Tècnica Superior d'Enginyeria Informàtica<br>Universitat Politècnica de València |

# 1 Resumen de las ideas clave

A la hora de abordar un desarrollo multimedia con necesidades de representación gráfica en 3D, el uso de OpenGL [1] es un estándar de referencia. Este documento aborda la realización de aplicaciones que hagan uso de las funciones de biblioteca del API de OpenGL [4] y muestra ejemplos multimedia interactivos. Se acompañan de detalles de generación del ejecutable en entorno GNU/Linux sin perder de vista las características multiplataforma que lo hacen portable a otros dispositivos.

Buena parte de los desarrollos actuales precisan más que de una visualización fotorealista estática, de una dinámica. Esto es, una forma de trabajar que busca ofrecer una **escena tridimensional cambiante en cuanto a contenido, más que la necesidad de unos gráficos extremadamente realistas**. Los cambios que provocan este dinamismo son el resultado de la interacción con el usuario, lo que va desde la directa manipulación de parámetros utilizando las opciones del interfaz, hasta la "comunicación" entre el mundo real (exterior al computador) y los objetos existentes en la escena que se está generando.

Los objetivos del presente documento son:

- Revisar la forma de desarrollo de aplicaciones basadas en el uso de funciones de biblioteca de OpenGL para establecer los mecanismos básicos.
- Revisar ejemplos de interacción complejos para establecer como se puede implementar esta "conexión" entre mundo real y sintético.
- Establecer unas pautas para revisar cualquier ejemplo de código, de este tipo de aplicaciones, para entender su modo de funcionamiento.

**No se pretende** ser exhaustivos en abordar la funcionalidad de OpenGL, sino ofrecer un conjunto de ejemplos complejos y funcionales para motivar al lector a la exploración en esta pequeña área.

# 2 Introducción

Las bases de cómo se utiliza OpenGL para realizar una aplicación se pueden consultar en profundidad en [1]. Aquí remarcaremos los elementos clave para permitir la discusión posterior de aplicaciones más elaboradas.

La fig. 1a muestra la captura de la ejecución de los ejemplos "1-2 hello.c " y "1-3 double.c" que aparecen en [1]. Espero que el lector tenga ganas de divertirse, porque empezamos. Para ejecutar los ejemplos los compilaremos, con una línea de órdenes del estilo de:

*$ gcc fichero_fuente.c -o fichero_destino -lglut -lGLU -lGL*

donde hay que sustituir el nombre del fichero fuente al que corresponda y el que queramos que tenga el ejecutable. Los tres últimos parámetros hacen referencia a tres bibliotecas de funciones que implementan las funciones de OpenGL. Cada una representa un nivel de abstracción de operaciones relacionadas con el desarrollo de aplicaciones gráficas con OpenGL[1]: GLUT, GLU y GL.

[1] Se puede ampliar la información al respecto en "GLUT and OpenGL Utility Libraries", disponible en <https://www.opengl.org/resources/libraries/>.

Se pueden identificar observando el prefijo del nombre de las funciones que veremos en el código. Así tenemos:

- GL es el nivel de operaciones más básico y cercano al
- GLU (*OpenGL Utility Library)*, contiene operaciones de más alto nivel para crear objetos (desde un cubo hasta una tetera, p.ej.), mipmaps con texturas, etc.. El crecimiento de OpenGL ha hecho que este nivel se esté subdividiendo en "extensiones" especializadas por lo que está siendo sustituido por GLEW … pero eso está fuera de nuestros objetivos por ahora.
- GLUT (*OpenGL Utility Toolkit*), agrupa a las funciones encargadas de la gestión de eventos del sistema de ventanas de forma transparente al sistema operativo sobre el que se ejecuta. Estrictamente hablando no es parte de OpenGL, pero suele acompañarlo en muchos ejemplos ya que permite el desarrollo de un interfaz de usuario solvente y portable. A fecha de hoy se utiliza la implementación *FreeGLUT*[2] como alternativa a la implementación original de Mark J. Kilgard y que dejó de actualizarse sobre 1998. Dado que la licencia original de GLUT no permite redistribuir el código modificado, Pawel W. Olszta inició en 1999 este proyecto para ofrecer una alternativa que continua mantenida y en expansión.

Así el sencillo resultado de la fig. 1a, una ventana con fondo negro y un rectángulo blanco, se obtiene con el código que aparece en el listado 1. Ahí podemos ver que:

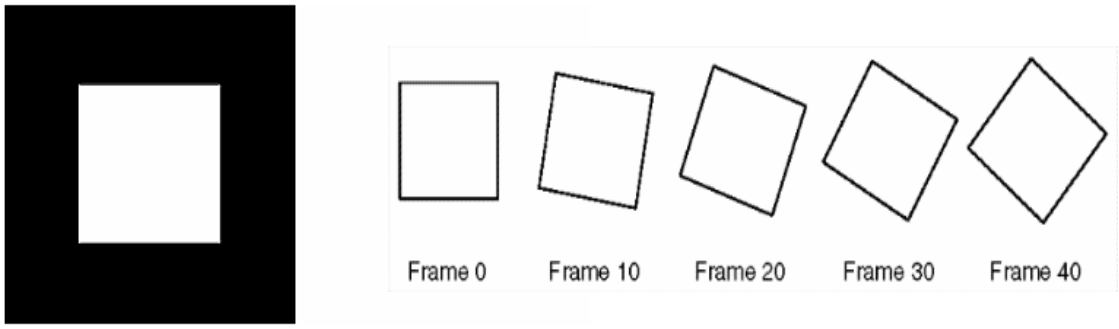
- Se realiza la inclusión de los dos ficheros de cabecera que referencias a las dos bibliotecas de funciones relativas a OpenGL que utiliza este ejemplo. Con la de *glut.h* bastaría porque incluye ya a *gl.h*, pero es habitual dejar la mención de cada una por separado.
- Las funciones con el prefijo "glut" se utilizan en el programa principal para crear la ventana de trabajo y establecer las funciones que van a responder a los eventos, tarea que es gestionada por *glutMainLoop*. En este primer caso "display" es una función que será llamada cuando el gestor de ventanas considere que es necesario repintar el contenido de la ventana y se asigna con *glutDisplayFunc*[3].
  En principio no se sale de ese bucle principal, aunque no es obligatorio hacerlo así y, hay ocasiones, en que es necesario tomar el control por que se necesita más tomar datos de entrada o las acciones de control asociadas, que mostrarlo en pantalla.
- Las primitivas gráficas son muy básicas (de la biblioteca de funciones GL) y se identifican con facilidad al observar las que tienen prefijo "gl". Están utilizadas en la implementación de la función que responde al evento de repintado de la ventana. Con *glClear* asigna el color negro al fondo. Posteriormente se asigna el color blanco como color por defecto con *glColor*. Y se crea una lista, con *glBegin*, de operaciones de pintado de vértices, *glVertex*, que han de mantener una cierta relación entre ellos.

El resultado que se muestra en la fig. 1b es la base de como realizar escenas de contenido dinámico, es una pequeña animación en la que, al pulsar un botón del ratón, un cuadrado empieza o para de dar vueltas. En este caso es la

[2] Véase en la URL <http://freeglut.sourceforge.net/>.

[3] Más detalles en <https://www.opengl.org/documentation/specs/glut/spec3/node46.html>.

temporización la que hace que cambie el contenido de la escena que se muestra.




*a)* *b)*

*Figura 1: Imágenes de ejemplos de partida obtenidos de [1]*

```
#include <GL/gl.h>
#include <GL/glut.h>

void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();
   glFlush ();
}

void init (void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE |
                                    GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");
   init ();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

*Listado 1: Ejemplo de contenido estático:1-2 hello.c* [1].

Veamos cómo se traduce en el código del ejemplo esa funcionalidad. Se han destacado en el listado 2, los nombres de las operaciones (funciones) más interesantes. Recuerde que aquí, lo que queremos destacar es la existencia de operaciones para realizar la interacción con el usuario. Esta parte recibe un menor tratamiento en la documentación existente que suele incidir en la parte del código de bajo nivel de uso de primitivas gráficas.

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>

static GLfloat spin = 0.0;

void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glPushMatrix();
  glRotatef(spin, 0.0, 0.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glRectf(-25.0, -25.0, 25.0, 25.0);
  glPopMatrix();
  glutSwapBuffers();
}

void spinDisplay(void)
{
  spin = spin + 2.0;
  if (spin > 360.0)
    spin = spin - 360.0;
  glutPostRedisplay();
}

void reshape(int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void mouse(int button, int state, int x, int y)
{
  switch (button) {
    case GLUT_LEFT_BUTTON:
      if (state == GLUT_DOWN)
        glutIdleFunc(spinDisplay);
      break;
    case GLUT_MIDDLE_BUTTON:
      if (state == GLUT_DOWN)
        glutIdleFunc(NULL);
      break;
    default:
      break;
  }
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE |
                       GLUT_RGB);
  glutInitWindowSize (250, 250);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

*Listado 2: Ejemplo de contenido dinámico:1-3 double.c* [1].

Además del evento de repintar (display), también se ha implementado, como se puede observar en el listado 2, la respuesta a los eventos de:

- *De redimensionado de la ventana (con glutReshapeFunc). Para que si el usuario modifica el tamaño de la ventana se pueda adaptar el dibujo de la escena a las nuevas condiciones,*
- *De detección de acciones del ratón (con glutMouseFunc). Permite realizar una acción en respuesta a la pulsación del botón principal (o central) del ratón que activa (o desactiva) el movimiento. , asignando la función de spinDisplay para que cambie el ángulo de rotación de la figura cada vez que se ejecute*
- *De "no hay nada que hacer" (con glutIdleFunc)*. Se ejecuta siempre que GLUT no esté procesando eventos del sistema de ventanas. Depende de las características de la máquina y de la aplicación. En general, esta función debería ser pequeña para no afectar al tiempo de respuesta de

un programa que interactúa con el usuario.

Observe que esta función termina con *glutPostRedisplay* que genera el evento de repintar la ventana, con lo que se consigue actualizar el contenido de la ventana en cuanto *spinDisplay* ha actualizado el valor que gobierna la animación.

Hay que resaltar que, a diferencia del ejemplo anterior, este ha inicializado la ventana con *GLUT_DOUBLE*. Esta definición configura la ventana para trabajar en modo de *Double Buffering*[4], lo que significa que se pinta en una zona de memoria (*buffer*) mientras se mantiene en otra el contenido a mostrar en pantalla. Es por este motivo que es necesario pedir al sistema que cambie de *buffer* para actualizar el contenido de la ventana, con lo cual se evitan problemas de saltos en las animaciones y se permite la generación de escenas en segundo plano (esto es, sin necesidad de visualizarlas en pantalla).

# 3 Ejemplos de interacción con elementos del interfaz

Nos vamos a centrar ahora en dos ejemplos que muestra la fig.2: *glutplane* (de la página de ejemplos de OpenGL [2]) y *atlantis* (de la página de *demos* de OpenGL [3]). Estos contienen el siguiente nivel en el desarrollo de aplicaciones con OpenGL, en tanto en cuanto aportan interacción con el usuario basada en menú y, aunque se podrían utilizar otras API de desarrollo[5], seguiremos utilizando GLUT y así no hay que añadir otras herramientas.

Para poder ejecutar estos ejemplos hay que modificar el fichero *Makefile* que define su generación. Es necesario modificarlo en términos de lo que indica la tabla 1, comentado algunas variables y creando otras.

Para la gestión de los menús, utilizando el API de GLUT, se crean con *glutCreateMenu*, se añaden opciones al menú con *glutAddMenuEntry* y se le asigna a un botón con *glutAttachMenu*. Las que recogen pulsaciones del teclado son las funciones que se pasan por parámetro a las funciones *glutKeyboardFunc* y *glutSpecialFunc*.

Empezando con *glutplane*, la aplicación nos permite dibujar unos aviones de papel y ofrece un menú desplegable (fig.3 izquierda) una “pequeña” interacción que añade la funcionalidad de crear otros aviones, eliminar uno, parar o empezar la animación y salir del programa. El menú se gestiona con tres funciónes:

- Se crea al inicializar la ventana con la *glutCreateMenu* que asociará la función (*menu* en este caso) que debe tratar la opción de menú que se escoja.
- Una o más *glutAddMenuEntry* que definen parejas de cadena de caracteres a mostrar y el valor numérico a devolver cuando se escoja esa opción para identificarla.
- La asignación del evento que lo lanzará, con *glutAttachMenu* y, que en este caso, se hace corresponder con el botón secundario del ratón.

[4] Véase “single vs double buffering” en <https://www.opengl.org/discussion_boards/showthread.php/197778-single-vs-double-buffering>

[5] Por ejemplo se podría haber utilizado SDL, GLFW, Allegro, SFML, Qt, wxWidgets, ...

- A lo largo del código se utilizará glutChangeToMenuEntry para modificar en tiempo de ejecución, el contenido de una opción del menú.

| Antes | Después |
|---|---|
| *TOP = ../../..*<br>*include $(TOP)/glutdefs*<br>*include $(ROOT)/usr/include/make/commondefs* | *#TOP = ../../..*<br>*#include $(TOP)/glutdefs*<br>*#include $(ROOT)/usr/include/make/commondefs*<br>*LDFLAGS = $(LLDLIBS)*<br>*GLUT = -lglut* |

*Tabla 1: Modificaciones al Makefile de atlantis.*


*Figura 2: Ejemplos"glutplane" (izquierda) y "atlantis" (derecha).*

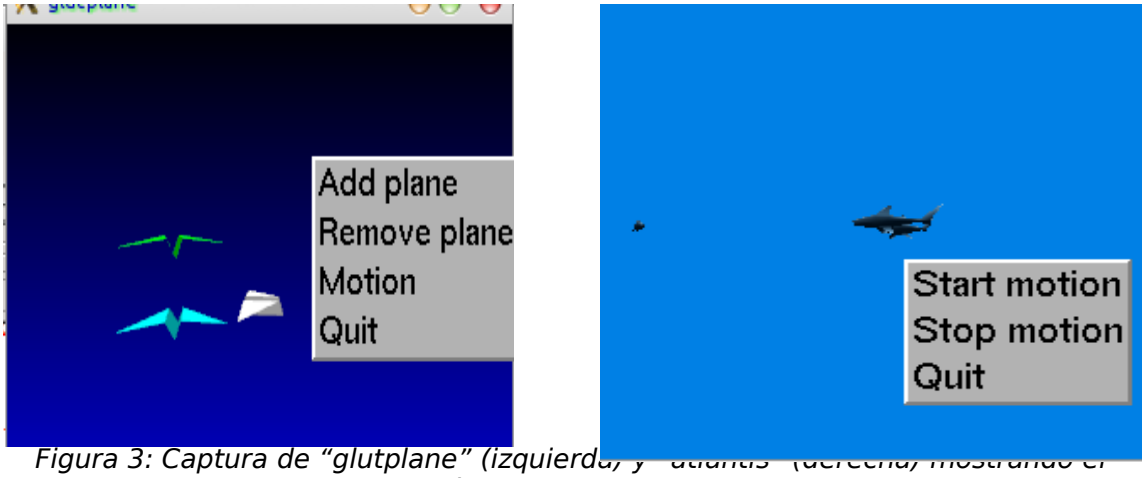



*Figura 3: Captura de "glutplane" (izquierda) y "atlantis" (derecha) mostrando el menú de cada una de ellas.*

La aplicación *glutplane* está basada en la función *draw*, asociada al evento de repintar (*glutDisplayFunc*), dibuja la lista de los aviones que se encuentren visibles y actualiza el *buffer* activo (*glutSwapBuffers*). Solo cuando la ventana de aplicación está encima de las demás se actualiza el avance de la animación, con la función *tick* y generar el evento de petición de repintar (glutPostRedisplay). También se puede interactuar con el teclado ya que se ha establecido la función *Key* como la encargada de gestionar los eventos de teclado (con el uso de *glutKeyboardFunc*). De esta forma se puede iniciar o parar la animación con la barra espaciadora (alterna entre una y otra opción), así como salir de la aplicación con la tecla Escape.

El modo de funcionamiento del ejemplo *atlantis* es muy similar: se puede iniciar o parar la animación, así como salir de la aplicación. Tanto mediante el menú, como mediante el teclado. La diferencia es que el número de objetos de la escena es fijo y son objetos complejos (tiburones, defines, ballenas y ballenatos) para los que existen operaciones específicas para definir sus modelos, las peculiaridades de su movimiento (trayectorias).

# 4 Ejemplo de interacción con los objetos sintéticos

Vamos ahora a exponer dos ejemplos más *triselec* ([2]) y *geoface* ([3]. Ambos permiten seleccionar y realizar acciones sobre los objetos que están en la escena. Un instante de su ejecución se muestra en la fig. 4.

En el caso de *triselect*, es posible seleccionar los objetos que se muestran en la escena. Son triángulos dibujados en un espacio 3D, pero la mecánica es generalizable para cualquier otro objeto (aviones de papel, ballenas, ...). Para ello hace uso de las funciones *Reshape* y *Draw*, asociadas a los típicos eventos de redimensionar (con *glutReshapeFunc*) y redibujar (con *glutDisplayFunc*), respectivamente.

En cuanto a la interacción, con el teclado (función Key asignada con *glutKeyboardFunc*) es posible acercar o alejar la cámara (con las teclas 'z' y 'Z), mostrar la información de todos los objetos visibles en el terminal (con 'f') y cambiar (con 'l') entre dibujado con modelo de alambres (líneas) o con modelado sólido (rellenando los triángulos). También es posible cambiar la rotación de la cámara con las teclas de las flechas del cursor, véase la .función *SpecialKey* (asignada con *glutSpecialFunc*).

Pero la parte más interesante es la interacción debida a la gestión de los eventos de ratón (esto es la función *Mouse* asignada con *glutMouseFunc*): nos permiten “tocar” los objetos 3D del mundo virtual y, cada vez que hacemos clic con el botón principal sobre un triángulo, éste cambia de color. Aunque podría ser otra acción ..., ¡claro! De hecho, el botón secundario lo hace crecer de tamaño y el central lo hace desaparecer. Para hacer posible esto, la función de pintado (*Draw*) tiene una llamada a la función (*Render*) que asignará un nombre (un identificador numérico con *glLoadName*) a la secuencia de operaciones que dibuja cada triángulo. Cuando el usuario toca un punto de la ventana con el ratón se le pide (en la función *DoSelect*) a OpenGL que “mire” bajo esas coordenadas, qué objeto esta allí, devolviendo ese “nombre” que se le habia asignado y que permite indexar la lista de objetos en pantalla para cambiar sus propiedades de acuerdo con lo que se deba hacer en cada caso.

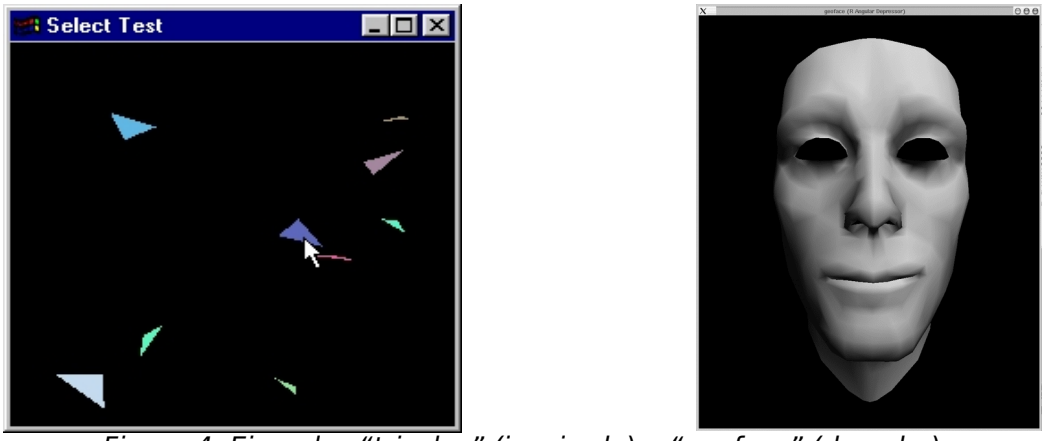



*Figura 4: Ejemplos "triselec" (izquierda) y "geoface" (derecha).*

| Antes | Después |
|---|---|
| *HEAD *create_face ( char *f1, char *f2 )*<br>*{*<br>*HEAD *h ;*<br>*h = _new ( HEAD ) ;*<br>*h->npolygons = 0 ;*<br>*h->npindices = 0 ;*<br>*h->npolylinenodes = 0 ;*<br><br><br>*read_polygon_indices ( f1, h ) ;*<br>*read_polygon_line ( f2, h ) ;*<br>*make_face ( h ) ;*<br>*returqn ( h ) ;*<br>*}* | *HEAD *create_face ( char *f1, char *f2 )*<br>*{*<br>*HEAD *h ;*<br>*h = _new ( HEAD ) ;*<br>*h->npolygons = 0 ;*<br>*h->npindices = 0 ;*<br>*h->npolylinenodes = 0 ;*<br><br>*// FALTA*<br>*h->nmuscles = 0;*<br>*// Fin de FALTA*<br><br>*read_polygon_indices ( f1, h ) ;*<br>*read_polygon_line ( f2, h ) ;*<br>*make_face ( h ) ;*<br>*returqn ( h ) ;*<br>*}* |

*Tabla 2: Modificaciones en la función create_face de make_face.c, dentro del proyecto de geoface.*

En el caso de *geoface* es necesario modificar el *Makefile* en los mimos términos en que se ha indicado en la tabla 1 con *atlantis*. Además, el ejecutable de *geoface* genera un "core", por lo que se habrá de revisar la función *create_face* (fichero *make_face.c):* es necesario asignar un valor inicial correcto al elemento *nmuscles* como se indica la tabla 2.

*Geoface* nos permite visualizar por “dentro” el objeto, la función *Key* asociada con (glutKeyboardFunc) permite cambiar la vistas. La fig. 5 muestra diferentes instantes de la ejecución de geoface: a) la visualización en modelo de alambres del objeto, lo que nos permite ver sus vértices y las conexiones entre ellos (aristas); b) el modelado sólido se lleva a cabo con la generación de superficies para agilizar el cálculo del color del objeto; y c) donde la interacción del usuario ha generado cambios en la malla con lo que muestra un “gesto” diferente al inicial (fig. 4b) en esta cara sintética: ¡una melancólica sonrisa!.

*a)* *b)* *c)*

*Figura 5: Diferentes instantes de la ejecución de “geoface”.*

Además permite modificar la malla del objeto de manera interactiva, con lo que conseguimos una funcionalidad similar a la mover los músculos de una cara: uno a uno y sin las restricciones físicas de una cara real. Para ello, la función *Special* (asociada a las teclas del cursor con *glutSpecialFunc*) permite desplazarse entre la lista de “músculos” de manera que en la barra de titulo de la ventana aparece el nombre del que está activo y se pueden variar su posición. También es posible escoger con las opciones de los desplegables del menú que tiene la aplicación y que la función *make_menus* crea y asocia al botón secundario del ratón.

# 5 Conclusión

A lo largo de este objeto de aprendizaje hemos visto una serie de ejemplos que utilizan el API de OpenGL para resaltar los elementos a observar en el caso de desarrollar aplicaciones de representación gráfica tridimensional bajo este estándar.

La naturaleza de funcionamiento de OpenGL basada en eventos y funciones asociadas que los tratan, ha centrado nuestro trabajo de exposición y observación en este documento. Nunca hemos pretendido ser exhaustivos, sino motivar al lector a la exploración ofreciéndole una serie de ejemplos complejos y funcionales sobre los que fijar la atención en determinados pasos. Así el lector será capaz de generar su propia mecánica para examinar otros nuevos ejemplos o sus propios desarrollos.

Para afianzar lo aprendido, estimado lector, te sugiero que te pongas manos a la obra introduciendo modificaciones en los ejemplos mostrados. Ya verás que interesante resulta y no olvides motrárselos a tus compañeros.

# 6 Bibliografía

[1] *Introduction to OpenGL. OpenGL Programming, Guide. Capítulo 1*. Disponible en: <http://www.glprogramming.com/red/>
[2] *OpenGL examples*. Disponible en: <https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.htm>
[3] OpenGL demos. Disponible en: <https://www.opengl.org/archives/resources/code/samples/glut_examples/demos/demos.html>
[4] Documentación del API OpenGL SDK.Disponible en:<https://www.opengl.org/sdk/docs/>
[5] C. Hock-Chuan. (2012). OpenGL Tutorial. An Introduction on OpenGL with 2D Graphics. Disponible en: <https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_Introduction.html>.