2016/2017

Práctica 2: Uso de GLUT. Parte 1.

CREANDO INTERFACES DE USUARIO

MARÍA GORETTI SUÁREZ RIVERO

Tabla de contenido

Introducción

En este informe se verá cómo se han desarrollado distintas tareas durante la segunda práctica de la asignatura que nos han permitido aprender más sobre cómo gestionar los eventos correspondientes a dispositivos de entrada salida en OpenGL, como son el teclado y el ratón. Además, hemos aprendido a trabajar con las librerías Glew y FreeGlut, y hemos concluido la práctica trabajando sobre las consecuencias de modificar el tamaño de la ventana, así como la relación existente entre el espacio lógico y el físico.

Desarrollo

Para esta práctica se han desarrollado un total de tres tareas.

Tarea 1

En esta primera tarea nos hemos enfocado en aprender a gestionar los eventos producidos por el teclado. Para ello, hemos dibujado una figura de modelo de alambre de una tetera, la cual es fácil de generar usando un simple comando en OpenGL, y hemos establecido que según la tecla que se pulse en el teclado, se cambie el color del modelo. Como era una pequeña tarea, solo contemplamos las teclas 1, 2, 3 y 4, las cuales cambian la tetera a color rojo, verde, azul y amarillo respectivamente, y para hacer uso de teclas especiales, usamos la tecla F1, la cual devuelve el color blanco original a la tetera.

Para poder realizar esta tarea hemos creado un nuevo proyecto de Visual Studio el cual ha sido configurado para que su directorio de includes tuviese “C:/OpenGL/include”, para que su directorio de librerías tuviese “C:/OpenGL/lib” y que en su sección de Entrada dentro de las preferencias del Vinculador estuviesen las librerías “freeglut.lib” y “glew32.lib”.

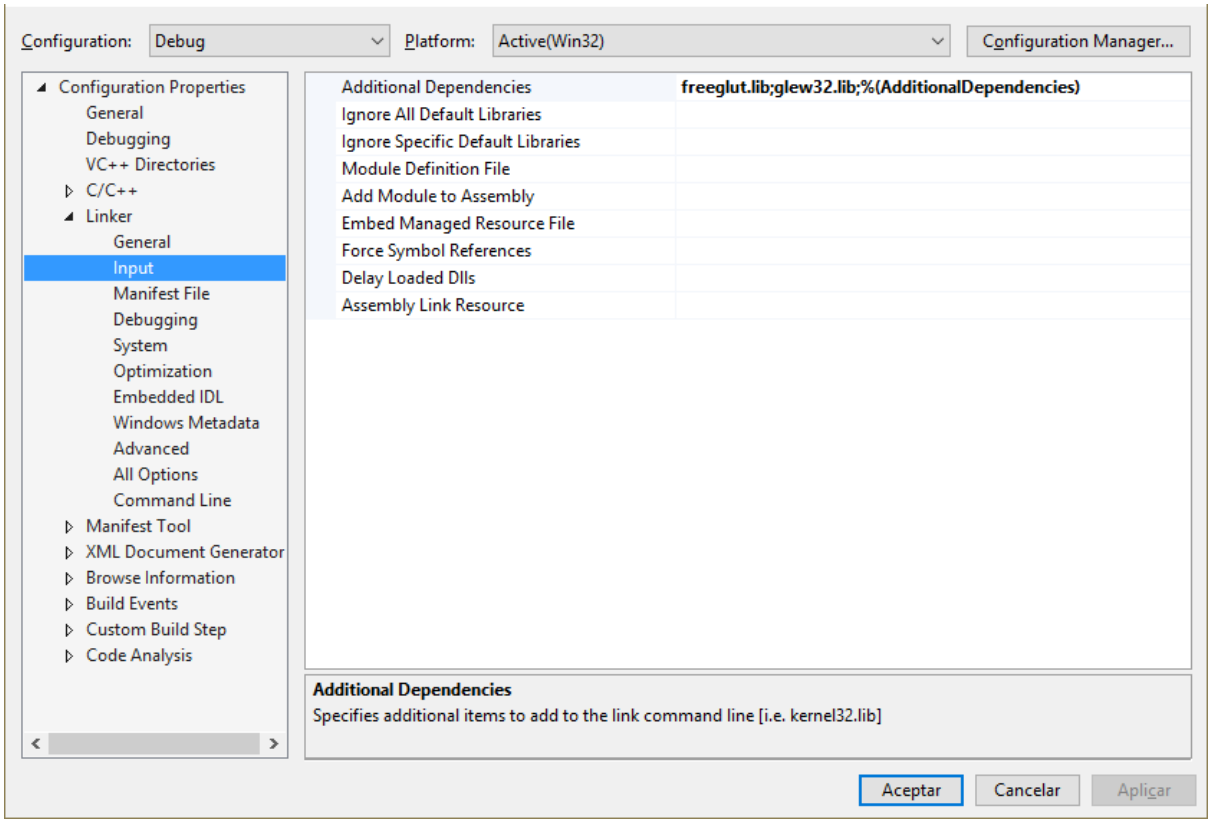


Figura 1: Configuración del proyecto de Visual Studio

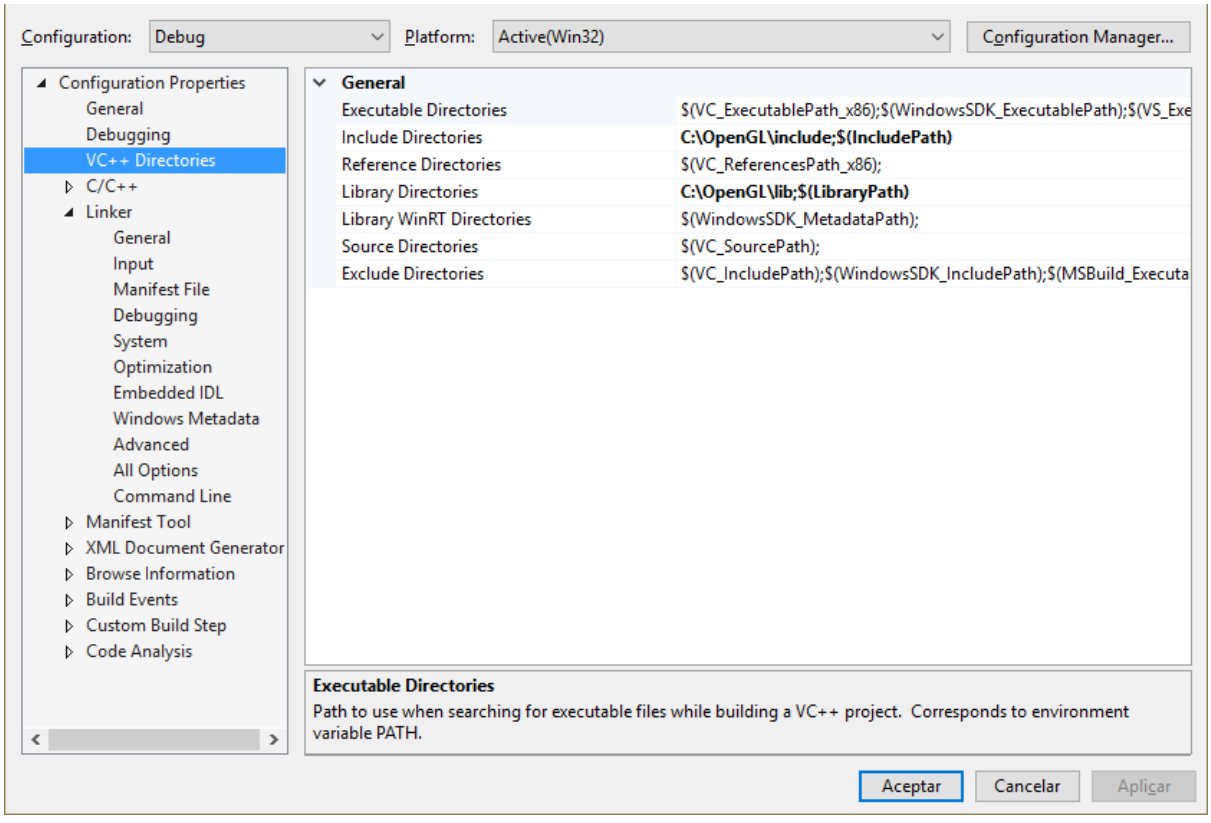


Figura 2: Configuración de archivos include y lib del proyecto Visual Studio

Hecha la configuración del proyecto, empezamos a elaborar un nuevo fichero .cpp que contendría lo necesario para crear nuestra pequeña aplicación OpenGL. Lo primero que hicimos fue incluir las librerías necesarias, entre ellas “glew.h” y “freeglut.h”, las cuales nos permitirían usar las funciones que necesitamos. Luego declaramos 3 variables globales que usaríamos para indicar los 3 colores básicos: rojo, verde y azul.

Declaradas las variables y las librerías, procedimos a inicializar GLEW con la pequeña función *InitGlew()* que podemos ver en el código. La librería GLEW expone eficientes métodos, en tiempo de ejecución, para determinar qué extensiones de OpenGL son soportadas. Todas las extensiones de OpenGL son listadas en un solo archivo de cabecera, que se genera automáticamente respecto la lista oficial de extensiones. Por tanto, gracias a esta librería podemos usar transparentemente todas las extensiones de OpenGL.

La inicialización del entorno OpenGL también es importante y es lo que se realiza en el método *Init()*. En él, establecemos el fondo negro del entorno con *glClearColor(0.0,0.0,0.0,0.0)*, realizamos algunas operaciones con matrices con *glMatrixMode()* y *glLoadIdentity()* que ya veremos más adelante en la asignatura, y establecemos el espacio lógico de trabajo tal y como se hizo en la práctica anterior y como puede verse en *Figura 3*, usando la función *glOrtho()*.

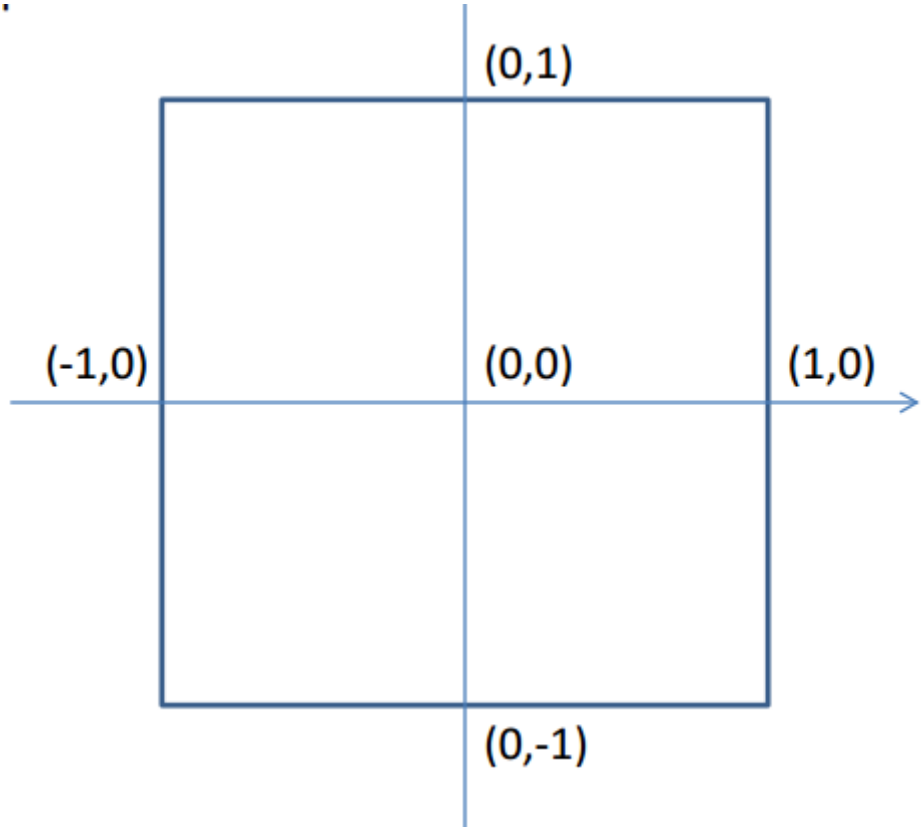


Figura 3: Coordenadas del espacio de trabajo a utilizar

Una vez configurado todo nuestro espacio de trabajo, configuramos la función *Display()* que se encargará de representar la figura tal y como la queremos. Como se ve en el método, primero borramos todo lo existente en el *framebuffer* antes de realizar cualquier

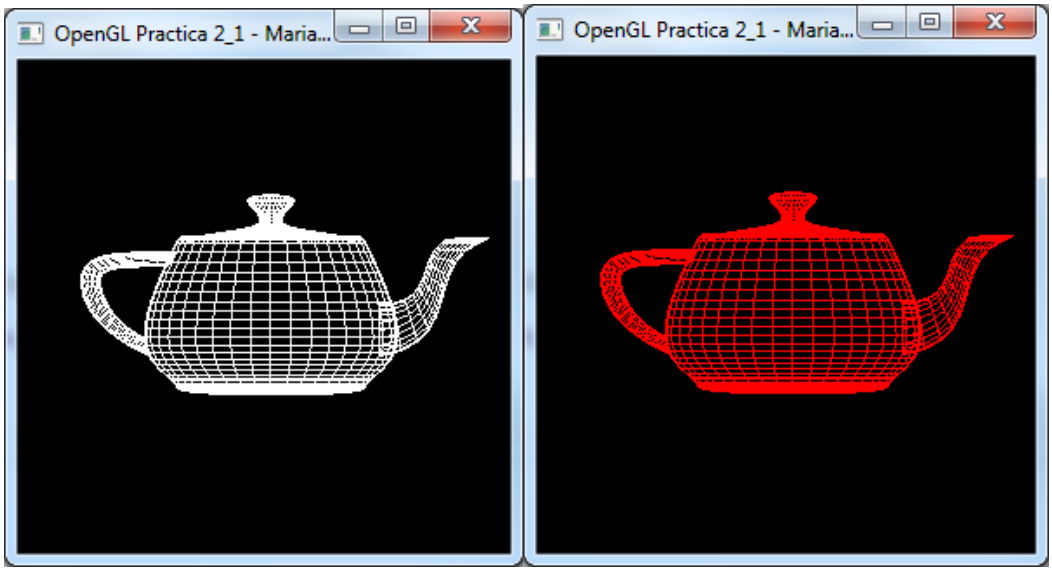
acción con *glClear(GL_COLOR_BUFFER_BIT)*. Completado esto, procederemos a indicar de qué color queremos que esté lo que vamos a dibujar con la función *glColor3f()* a la cual le pasamos como parámetros las tres variables globales (*red*, *green* y *blue*) definidas anteriormente, ya que ellas indicaran el valor de cada componente RGB. Luego, dibujamos la tetera con *glutWireTeapot()*. Como ya hemos especificado el color y el modelo de lo que queremos pintar, actualizamos finalmente el *framebuffer* con *glFlush()*.

A esta altura, empezamos a crear los métodos que se utilizarían para gestionar los eventos del teclado. Se creó un primer evento llamado *Teclado1* para las teclas habituales que, usando un *switch*, evaluaba el valor de la tecla pulsada, y si correspondía con 1, 2, 3 o 4, cambiaba los valores de las variables globales *red*, *green* y *blue* para configurar el nuevo color correspondiente a la tecla pulsada. Tras establecer esos nuevos valores, se actualiza la imagen llamando a *glutPostRedisplay()*.

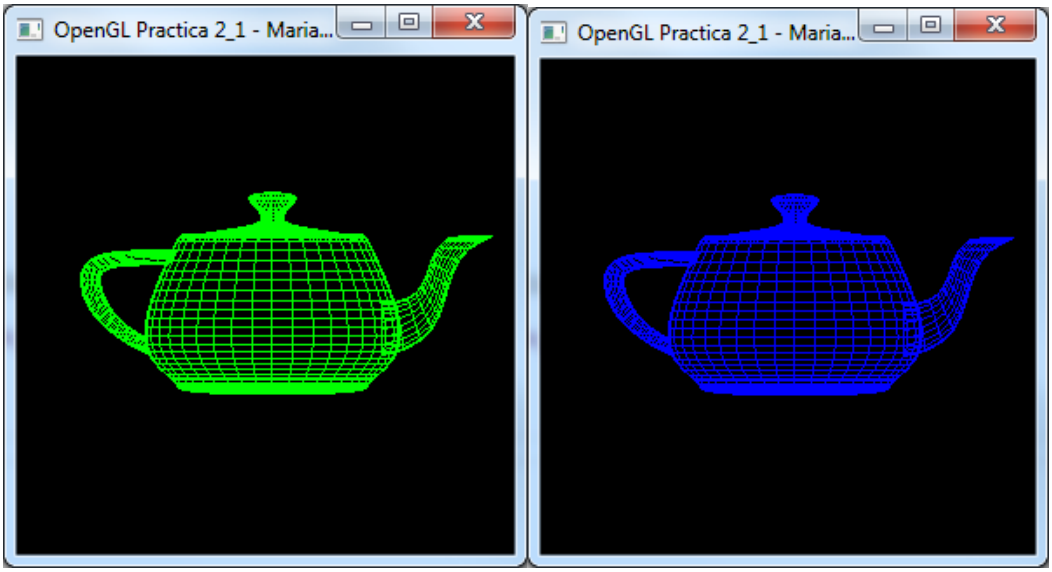
El otro evento para gestionar los eventos de teclado se llama *Teclado2* y se usa para establecer qué sucede cuando pulsamos una tecla especial, como F1. Vemos, de hecho, como con un *if* comprobamos si la tecla que ha sido pulsada es F1, y en caso de serlo, establecemos los valores necesarios de las variables globales *red*, *green* y *blue* para que la tetera se vea blanca. Tras esto, actualizamos la imagen llamando a *glutPostRedisplay()*.

Finalmente, se estableció el *main()* del programa, encargado de inicializar todo el contexto. Se empieza inicializando GLUT con la función *glutInit()* a la cual le pasamos los propios parámetros del *main()*. Tras esto, establecemos la posición y el tamaño que tendrá la ventana con *glutInitWindowPosition()* y *glutInitWindowSize()*. Utilizamos luego *glutInitDisplayMode()* para especificar con sus parámetros que utilizaremos RGBA en vez de RGB, ya que este primero posee transparencia (aunque realmente en esta práctica no se usa, y que utilizaremos un buffer simple ya que no vamos a realizar animaciones. Después creamos la ventana con glutCreateWindow(), pasándole por parámetro el título que esta tendrá, establecemos un icono para el cursor, inicializamos el entorno llamando a los métodos *InitGlew()* e *Init()* previamente elaborados, y establecemos las funciones que trataran el dibujo de la figura y las que se encargaran de gestionar los eventos de teclado. Para indicar que el método *Display()* se utilizará como función para dibujar el modelo se utiliza la función *glutDisplayFunc()* y se le pasa como parámetro el nombre del método encargado. Usamos de la misma manera *glutKeyboardFunc(Teclado1)* para especificar que el gestor de eventos de teclado para letras comunes será *Teclado1* y *glutSpecialFunc(Teclado2)* para especificar que el gestor de eventos de teclado relacionado con las teclas especiales será *Teclado2*. Hecho esto, y antes de terminar el método, llamamos a *glutMainLoop()* para dejar a la aplicación en un bucle constante que nos permita trabajar con ella y que no se cierre tras ejecutar una vez todos los métodos.

Al ejecutarla, podemos ver como aparece inicialmente una tetera blanca como en las figuras de abajo, y como al apretar 1 la figura se vuelve roja, al apretar 2 se vuelve verde, etc. De la misma manera, al apretar F1, se vuelve de nuevo al color blanco original.



Figuras 4 y 5: Visualización inicial de la tetera y al pulsar 1



Figuras 6 y 7: Visualización de la tetera al apretar 2 y 3

P2_Tarea1.cpp*

(Ámbito global) InitGlew()

```
#include <stdio.h>

#include <GL\glew.h>
#include <GL\freeglut.h>

float red = 1.0, green = 1.0, blue = 1.0;

void InitGlew() {
    GLenum glew_init = glewInit();
    if (glew_init != GLEW_OK) {
        fprintf(stderr, "Error: %s\n", glewGetErrorString(glew_init));
    } else {
        fprintf(stdout, "Status: Using GLEW %s\n", glewGetString(GLEW_VERSION));
    }
}

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

void Display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(red, green, blue);
    glutWireTeapot(0.5);
    glFlush();
}
```

Código 1: Funciones de inicialización y display()

P2_Tarea1.cpp*

(Ámbito global)

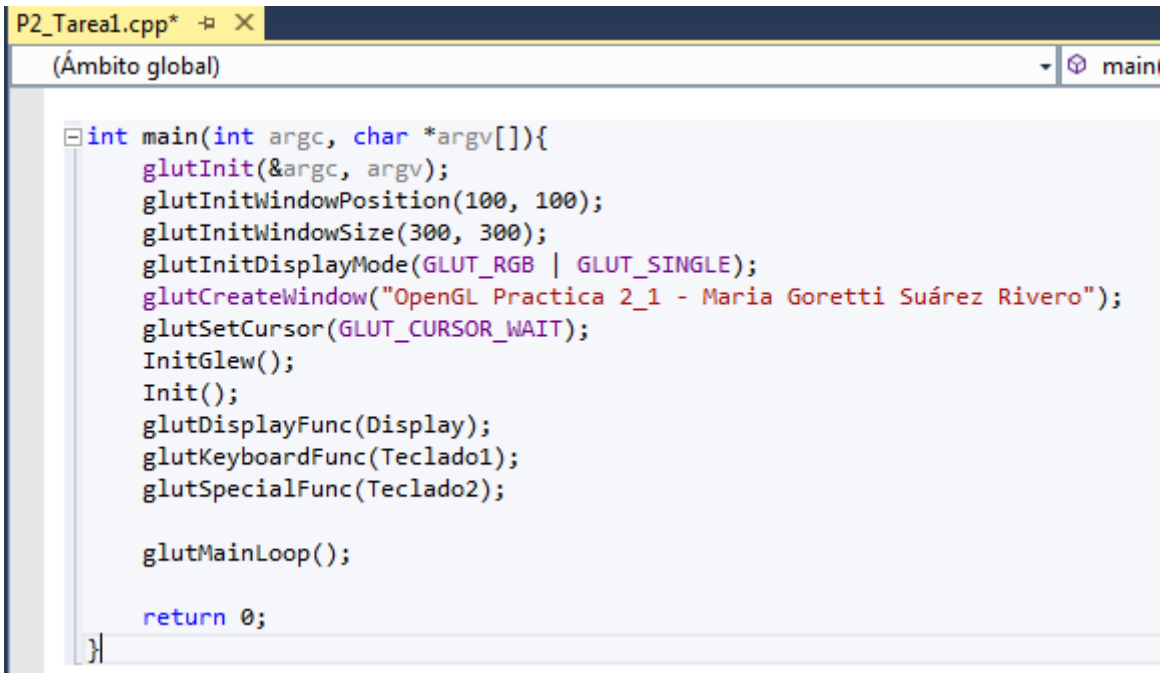
```
void Teclado1(unsigned char key, int x, int y) {
    switch (key) {
    case '1':
        red = 1.0;
        green = 0.0;
        blue = 0.0;
        break;
    case '2':
        red = 0.0;
        green = 1.0;
        blue = 0.0;
        break;
    case '3':
        red = 0.0;
        green = 0.0;
        blue = 1.0;
        break;
    case '4':
        red = 1.0;
        green = 1.0;
        blue = 0.0;
        break;
    }

    glutPostRedisplay();
}

void Teclado2(int key, int x, int y) {
    if (key == GLUT_KEY_F1){
        red = 1.0;
        green = 1.0;
        blue = 1.0;
    }
    glutPostRedisplay();

}
```

Código 2: Funciones de modificación de colores según lo leído del teclado



Código 3: Funciones principal

Tarea 2

Para esta segunda tarea creamos dos ventanas que fueron inicializadas de forma diferente, teniendo cada una un color de fondo distinto y nada más que mostrar al usuario. Sin embargo, existe una peculiaridad, y es que cada ventana está asociada a un gestor de ratón distinto, ya que en esta tarea nos concentraremos en aprender a gestionar los eventos producidos por el ratón. Una de las ventanas capturará cuando se presiona el botón izquierdo de la otra, mientras que la otra capturará cuando se libera el botón derecho del ratón, y ambas acciones serán dichas por la consola para comprobar su funcionamiento.

En esta tarea creamos un proyecto de Visual Studio con la misma configuración que en la tarea anterior, y el archivo .cpp comenzó con la declaración de las mismas librerías y de la función *InitGlew()*, ya que de aquí en adelante será totalmente necesario hacerlo. Sin embargo, las variables globales declaradas son distintas. En esta tarea declaramos dos enteros, *win1* y *win2*, que serán los identificadores de cada una de las ventanas que vamos a crear.

Al tener dos ventanas distintas en esta tarea, necesitamos dos funciones *Init()* para inicializar el contenido de cada una. Creamos una función *Init1()* que serviría para inicializar la primera ventana con un color de fondo rojo, establecido con la función *glClearColor(1.0, 0.0, 0.0, 0.0)*. Tras esto, establecimos operaciones matriciales como en la tarea anterior con *glMatrixMode(GL_PROJECTION)* y *glLoadIdentity()* y establecimos el mismo espacio de trabajo con *glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 0.0f)*.

Creamos otra función de inicialización, *Init2()*, que cotenía lo mismo que *Init1()*, salvo que el color establecido con *glClearColor()* en este caso correspondía al azul.

De la misma manera, creamos dos funciones de *Display()*. Ambas funciones, tanto *Display1()* como *Display2()* contenían lo mismo, simplemente limpiar el *framebufferr* y luego actualizarlo, sin nada que mostrar al usuario. Ambas funciones podrían haberse unificado perfectamente en una sola y haber establecido a posteriori que cada ventana utilizase el mismo método para redibujar el contenido.

Tras esto, creamos los dos gestores de eventos para el ratón. El primer gestor se llama *MouseVentana1()* y, como puede verse en el código, se encarga de comprobar mediante un *if* si el botón que se ha pulsado corresponde con el botón izquierdo, haciendo uso de la variable GLUT_LEFT_BUTTON y si además el botón está presionado, comparando su estado con GLUT_DOWN. Si esto es así, se mostrará por consola un mensaje.

El segundo gestor se llama *MouseVentana2()* y trabaja de la misma forma, pero en este caso se encarga de comprobar de si el botón es el derecho comparándolo con GLUT_RIGHT_BUTTON y si acaba de ser despulsado comparando su estado con GLUT_UP. Si se da este caso, se mostrará por consola otro mensaje, indicando este estado.

Finalmente, declaramos el *main()* de la aplicación. Comenzamos inicializando GLUT y configurando la ventana y su visualización al igual que en la tarea anterior, con los mismos parámetros inclusive. A continuación, creamos una ventana usando *glutCreateWindow()* y su resultado se lo asociamos a la variable global *win1*, almacenando así el identificador de la primera ventana. Una vez creada, llamamos al método de inicialización de GLEW y luego al primer inicializador, *Init1()*. Establecemos *Display1()* como la función de redibujo que utilizará esta ventana, así como *MouseVentana1()* como el gestor de eventos de ratón que esta poseerá, usando para esto último *glutMouseFunc()*. Tras esto, creamos la segunda ventana, asociando su identificador a *win2*, y llamando a posteriori a *Init2()* y estableciendo *Display2()* como la función que usará para repintar dentro de la ventana. Acabamos asociando el segundo gestor de eventos de ratón, *MouseVentana2()* a esta segunda ventana, usando *glutMouseFunc()*. Finalmente, entramos en bucle con *glutMainLoop()*.

Al ejecutar, podemos ver algo parecido a lo que se ven en la figura de abajo (*Figura 8*). Al apretar el botón izquierdo en la primera ventana, aquella del fondo rojo, sale un mensaje en la consola indicando que lo hemos presionado, y de la misma manera si soltamos el botón derecho sobre la segunda ventana, aquella de fondo azul, sale un mensaje en la consola indicando que lo hemos soltado. Si hacemos cualquier otra cosa con el ratón, no sucederá nada, ya que no se contempla en el código otro tipo de evento.

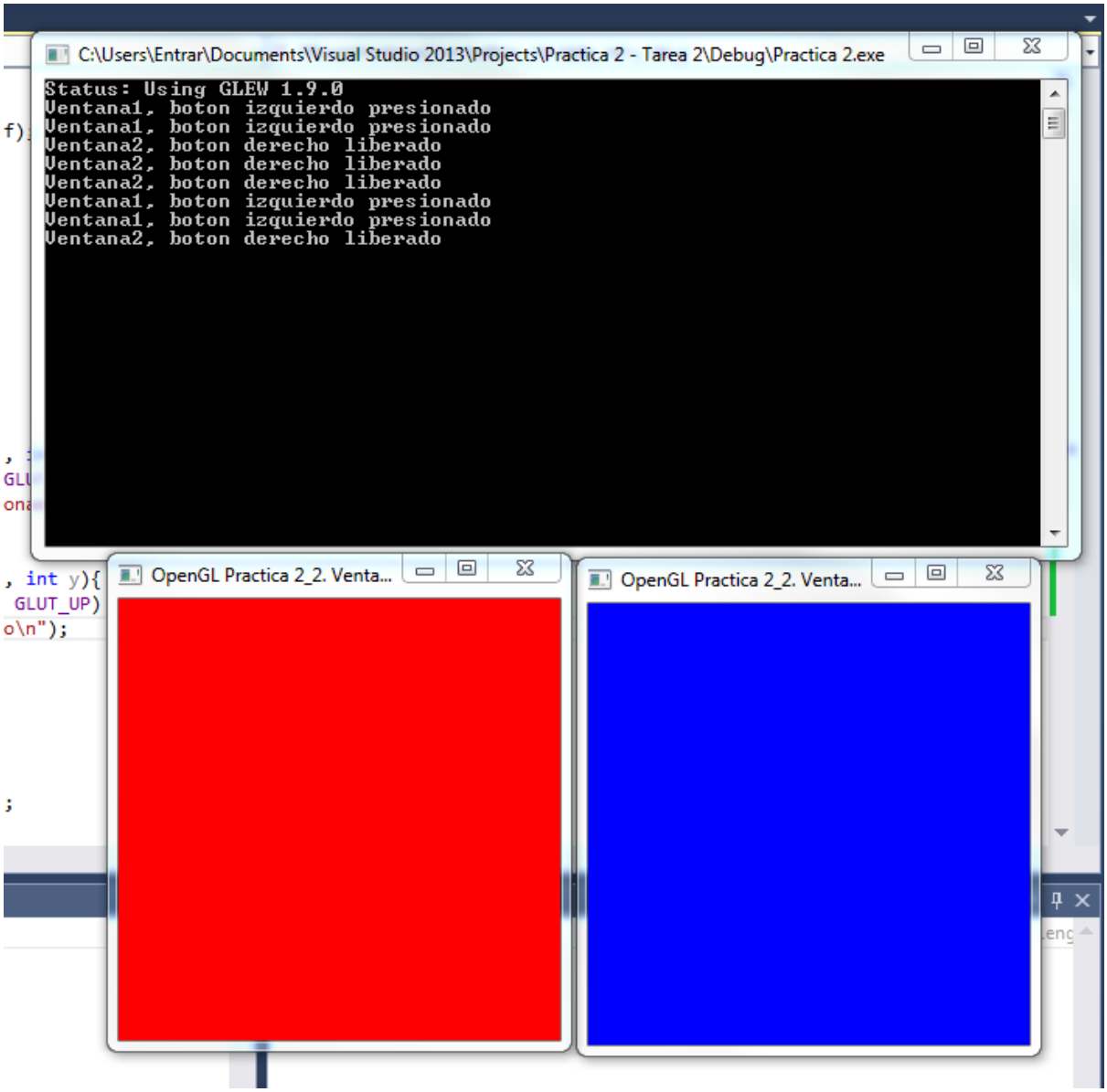


Figura 8: Funcionamiento del gestor de eventos de ratón

P2_Tarea2.cpp

(Ámbito global)

```
#include <stdio.h>

#include <GL\glew.h>
#include <GL\freeglut.h>

int win1, win2;

void InitGlew() {

    GLenum glew_init = glewInit();
    if (glew_init != GLEW_OK) {
        fprintf(stderr, "Error: %s\n", glewGetErrorString(glew_init));
    }
    else {
        fprintf(stdout, "Status: Using GLEW %s\n", glewGetString(GLEW_VERSION));
    }
}

void Init1() {
    glClearColor(1.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 0.0f);
}

void Init2() {
    glClearColor(0.0, 0.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 0.0f);
}

void Display1() {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

void Display2() {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
```

Código 4: Funciones de inicialización y de visualización

P2_Tarea2.cpp

(Ámbito global) Init2()

```
void MouseVentana1(int button, int state, int x, int y){
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        printf("Ventana1, boton izquierdo presionado\n");
}

void MouseVentana2(int button, int state, int x, int y){
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
        printf("Ventana2, boton derecho liberado\n");
}

int main(int argc, char *argv[]){
    glutInit(&argc, argv);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(300, 300);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);

    win1=glutCreateWindow("OpenGL Practica 2_2. Ventana1 - Maria Goretti Suárez Rivero");
    InitGlew();
    Init1();
    glutDisplayFunc(Display1);
    glutMouseFunc(MouseVentana1);

    win2 = glutCreateWindow("OpenGL Practica 2_2. Ventana2 - Maria Goretti Suárez Rivero");
    Init2();
    glutDisplayFunc(Display2);
    glutMouseFunc(MouseVentana2);

    glutMainLoop();

    return 0;
}
```

Código 5: Funciones para los eventos generados por el ratón y función principal

Tarea 3

En esta tercera y última tarea trabajaremos con el redimensionamiento de las ventanas y cómo el modificar el tamaño de estas hace que se modifique la relación de aspecto de la imagen (la relación de dimensiones existente entre los ejes x e y). Para ilustrar el efecto de tener o no una función que se encargue de redefinir las dimensiones del dibujo cuando la ventana cambia, se crearán dos ventanas, una en la que se gestionará el cambio de dimensiones y otra en la que no se hará. Además, hay que tener en cuenta que lo que define el dibujo es el espacio lógico, y es este el que hay que cambiar para ajustar al físico y que la imagen no aparezca distorsionada. Por tanto, cuando se deforme la ventana, lo ideal sería recalcular cuál será el espacio lógico ideal que tenga una relación ancho-alto que se corresponda con el nuevo espacio físico.

Creamos un nuevo proyecto de Visual Studio con las mismas características que los dos anteriores, y le añadimos un archivo .cpp que incluirá de nuevo las librerías glew.h y

freeglut.h así como la inicialización de GLEW con el método *InitGlew()*. Añadimos la misma función de inicialización (*Init()*) que usamos en la primera tarea, y al igual que en la segunda tarea, declaramos dos variables globales (*win1* y *win2*) que nos servirían para identificar cada una de las ventanas que fuésemos a crear.

Al igual que hicimos en la segunda tarea, creamos dos funciones de *Display()* con el mismo contenido ya que usamos un método para una ventana y otro para la otra ventana. Dentro de cada método, empezamos borrando todo lo que pueda haber en el *framebuffer*, luego pintamos un modelo de alambre de la tetera como hicimos en la primera tarea con la función *glutWireTeapot(0.5)* y una vez hecho el dibujo, actualizamos el *framebuffer* con *glFlush()*.

Para esta tarea hemos definido un método llamado *Dimensiones()*, el cual será considerado como la función de gestión del cambio de dimensiones de la aplicación, y a la cual se llamará para adaptar el dibujo en caso de que el espacio físico cambie. Para saber cómo se va a redimensionar, hay que partir de la situación inicial, y es que la ventana se creará con unas dimensiones físicas de 300x300 y proyectará unas dimensiones lógicas de 2x2. Por tanto, para mantener la correcta proporción cuando las dimensiones físicas cambien tanto su ancho como su alto, se deberán implicar unas dimensiones lógicas dX y dY que se calculan de la siguiente manera:

- Si ancho > alto: dY=2 y dX=2*ancho/alto
- Si alto > ancho: dX=2 y dY = 2*alto/ancho

Además, el espacio ortogonal se debe situar centrado, desde -dX/2 hasta dX/2 e igual en el eje Y. De esta manera, la ventana que tenga como función de *reshape* este método, la figura que se visualizará mantendrá una relación de aspecto correcto, pero su tamaño puede variar.

Por último, definimos el *main()*, inicializando glut y la ventana tal y como se hizo en las anteriores tareas. Creamos una primera ventana a la cual le asociamos una función de dibujo *Display1()* pero sin asociar ninguna función para gestionar la redimensión. Luego, creamos una segunda ventana, la inicializamos, le asociamos la función de dibujo *Display2()* y a esta si le asociamos *Dimensiones* como la función de *reshape* usando *glutReshapeFunc(Dimensiones)*. Luego, entramos en bucle con *glutMainLoop()*.

Al ejecutar la aplicación, podemos ver que se generan dos ventanas con una tetera dibujada. Sin embargo, si aumentamos el tamaño de la primera ventana, vemos como la tetera no conserva la relación de aspecto, algo que sí ocurre en la segunda ventana a la cual le hemos asociado la función de *reshape*.

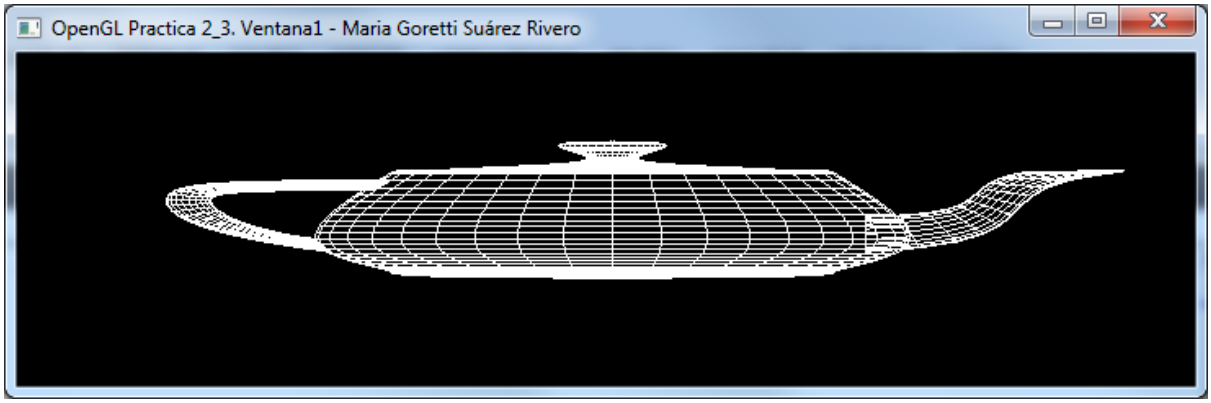


Figura 9: Ventana sin función de *reshape* asociada

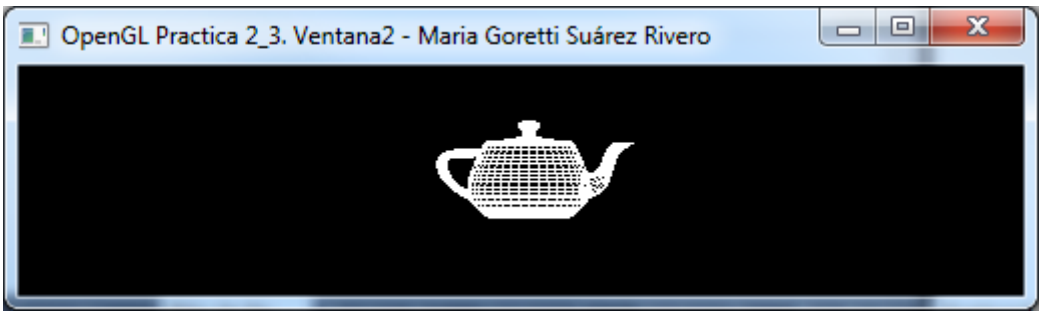


Figura 10: Ventana con función de *reshape* asociada

P2_Tarea3.cpp*

(Ámbito global) — InitGlew()

```
#include <stdio.h>

#include <GL\glew.h>
#include <GL\freeglut.h>

int win1, win2;

void InitGlew() {
    GLenum glew_init = glewInit();
    if (glew_init != GLEW_OK) {
        fprintf(stderr, "Error: %s\n", glewGetErrorString(glew_init));
    }
    else {
        fprintf(stdout, "Status: Using GLEW %s\n", glewGetString(GLEW_VERSION));
    }
}

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

void Display1() {
    glClear(GL_COLOR_BUFFER_BIT);
    glutWireTeapot(0.5);
    glFlush();
}

void Display2() {
    glClear(GL_COLOR_BUFFER_BIT);
    glutWireTeapot(0.5);
    glFlush();
}
```

Código 6: Funciones de inicialización y de visualización

P2_Tarea3.cpp*

(Ámbito global) Init()

```
  //Todo pasa porque tenemos que manejar el espacio fisico y el espacio logico para que se ajust
void Dimensiones(int ancho, int alto) {
    float dx = 2.0;
    float dy = 2.0;

    if (ancho > alto) {
        dx = 2.0*(float)ancho / (float)alto;
    }

    if (alto > ancho){
        dy = 2.0*(float)alto / (float)ancho;
    }

    glViewport(0, 0, ancho, alto);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-dx / 2.0, dx / 2.0, -dy / 2.0, dy / 2.0, 1.0f, 0.0f);
    glutPostRedisplay();
}

int main(int argc, char *argv[]){
    glutInit(&argc, argv);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(300, 300);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);

    win1 = glutCreateWindow("OpenGL Practica 2_3. Ventana1 - Maria Goretti Suárez Rivero");
    InitGlew();
    Init();
    glutDisplayFunc(Display1);

    win2 = glutCreateWindow("OpenGL Practica 2_3. Ventana2 - Maria Goretti Suárez Rivero");
    Init();
    glutDisplayFunc(Display2);
    glutReshapeFunc(Dimensiones);

    glutMainLoop();

    return 0;
}
```

Código 7: Función para controlar la redimensión del objeto y función principal

Conclusión

En esta práctica, tal y como se ha podido ver durante el desarrollo de las tres tareas, hemos podido ver cómo gestionar los eventos de teclado en OpenGL, cambiando el color del modelo de la tetera según el número o tecla especial que presionasemos. Pudimos ver también cómo gestionar los eventos del ratón, para que se mostrasen ciertos mensajes por pantalla según si apretamos una tecla del ratón u otra. Por último, una vez tuvimos controlada toda la gestión de eventos, aprendimos a redefinir las dimensiones del dibujo al nivel de funciones OpenGL para preservar la relación de aspecto cada vez que redimensionásemos la ventana. Con esta última tarea, aprendimos también sobre la relación entre el espacio lógico en el que se sitúa el dibujo y el espacio físico.

Bibliografía

- Documentación de la práctica (Moodle)