UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Introducción al uso de SDL como interfaz de alto nivel para OpenGL

Apellidos, nombre	**Agustí i Melchor, Manuel** (magusti@disca.upv.es)
Departamento	**Departamento de Informática de Sistemas y Computadores (DISCA)**
Centro	Universitat Politècnica de València

1 Resumen de las ideas clave

Cuando se desarrolla una aplicación multimedia interactiva y se necesita centrarse en la lógica de funcionamiento, es deseable abstraerse de los detalles propios de la i**nterfaz de usuario**. Si, además, se tiene en cuenta que esta parte es la más diferente entre plataformas (a nivel de sistema operativo y de acceso al hardware disponible), es cuanto más aconsejable disponer de ese nivel de despreocupación de los detalles de bajo nivel que diferencian a las posibles plataformas en las que queremos desplegar la aplicación.

El lector encontrará, al leer y experimentar con el contenido de este artículo, una serie de servicios propios de interfaz con el usuario que le permitirán enfocar un buen número de **aplicaciones interactivas e inmersivas**, que puedan ser portadas entre plataformas.

En particular, esto lo podemos apreciar a la hora de desarrollar aplicaciones con **OpenGL**. En estas, el centro del interés es el renderizado de una escena tridimensional sobre un área de pantalla bidimensional. Por ello se propuso, en los inicios de OpenGL, complementarlo con una capa de alto nivel de carácter multiplataforma, que permitiera: esta separación de tareas, la portabilidad del desarrollo y minimizar el impacto en cuanto a consumo de recursos necesarios. La elección recibió el nombre de *The OpenGL Utility Toolkit* (GLUT) [1]. Fue desarrollado por Mark Kilgard [2] allá por el 1994, sobre el sistema gráfico de ventanas X de Unix (*X Window System*) y se empezaría llamando GLX. Sería portado a Microsoft Windows (WGL) por Nate Robins y en macOS nos encontramos con su propia implementación de GLUT (hasta la llegada de Metal[1]).

Además de GLUT, otras alternativas se han ido desarrollando[2], aquí se presenta un ejemplo de código de OpenGL con una de ellas: ***Simple DirectMedia Layer*** (**SDL**). Este artículo se centrará en la explicación en un ejemplo de aplicación hecha con GLUT, reescrita con SDL, para ver los cambios típicos al utilizarla en lugar de GLUT. Hay dos grandes versiones de SDL la 1.2 y la 2.0. Este trabajo se centra en la segunda, que se suele denotar como SDL2.

2 Objetivos

Una vez que el lector se lea con detenimiento este documento y explore el código que se adjunta, podrá ver un buen número de funciones de SDL con las que permite realizar acciones de interfaz de usuario y, más en concreto, será capaz de:

- Explicar las posibilidades del API de SDL.
- Explorar un ejemplo de conversión de aplicación realizada inicialmente con GLUT y que se reescribirá con SDL.
- Poner en marcha un ejemplo básico y añadirle funcionalidades de SDL.

3 Introducción

OpenGL es el estándar de facto de API[3] de gráficos 2D y 3D, independiente de sistema operativo y del sistema de ventanas. Esta especificación está implementada en un gran número de plataformas

[1] Mac OS ha desarrollado Metal como alternativa a OpenGL. Véase más sobre qué es Metal en <https://developer.apple.com/metal/> y <https://es.wikipedia.org/wiki/Metal_(API)>.

[2] El lector puede ampliar estas opciones en la página de *Related toolkits and APIs* de OpenGL disponible en <https://www.khronos.org/opengl/wiki/Related_toolkits_and_APIs>.

de computadores. Para usar las funciones del API de OpenGL hay que realizar una etapa de inicialización, compleja y diferente en cada plataforma dado que OpenGL es multiplataforma. Es una tarea que se divide en dos partes [6]:

- La carga de las funciones de OpenGL. Generalmente, para usar bibliotecas de funciones hay que incluir las cabeceras oportunas y enlazarlas en la fase de compilación. En el caso de OpenGL, no es hasta el momento de la ejecución cuándo se sabe la forma de encontrarlas y el conjunto de las que están disponibles es diferente, puesto que depende de la versión instalada y de si OpenGL que está implementado por el manejador de la tarjeta gráfica o por un componente instalado en el núcleo del sistema operativo.
- La creación del contexto para OpenGL. Esto es, el punto de unión entre el sistema de ventanas (encargado de crear una ventana, gestionar los eventos propios de la entrada del usuario) y OpenGL (la estructura de datos que el servidor de OpenGL gestiona para renderizar una escena).

Estas tareas no forman parte de la especificación de OpenGL; por lo que, a su alrededor, diferentes bibliotecas (*toolkits* en la jerga de OpenGL) han ido apareciendo para dar solución en diferentes plataformas a la creación de ventanas y a la gestión de la entrada del usuario. En el sitio web de OpenGL [6], se mencionan tres: GLUT, Freeglut y GLFW. Este artículo se centra en una cuarta alternativa: SDL, una solución multiplataforma (disponible para las tres plataformas de escritorio, Linux/Unix, macOS X, Windows y para un amplio número de sistemas de videoconsolas y dispositivos portables), que proporciona servicios para la gestión de ventanas y de eventos de entrada.

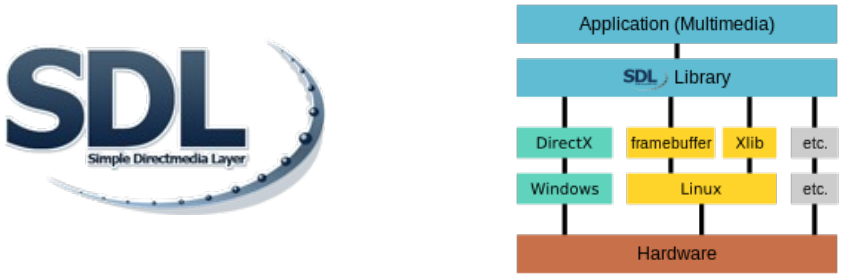
3.1 Uso de Simple DirectMedia Layer (SDL)

SDL es (véase [3] y [7]) una biblioteca para desarrollo multiplataforma, diseñada para proporcionar un nivel de abstracción, véase Figura 1, para los componentes multimedia del hardware de un computador. Ofrece acceso a bajo nivel de audio, dispositivos de entrada (como teclado, ratón y *joystick*, el hardware gráfico 3D (vía OpenGL, Vulkan. Metal, Direct3D11 o sceGu – el soporte que ofrece Sony para OpenGL en la PSP, entre otros-) y 2D a través del *framebuffer*.

Con SDL se facilita que los desarrolladores pueden realizar aplicaciones multimedia que son portables a diferentes sistemas operativos como Android, iOS, Linux, macOS o Windows y también otras plataformas de *wearables* (como Tizen) y de videoconsolas (como Switch, Oyuya, PSP, Vita o Play2). Se pueden encontrar ejemplos de reproductores de medios, emuladores y juegos (p. ej. en los catálogos de *Valve* y *Humble Bundle*). SDL 2.0 se distribuye bajo licencia zlib, lo que permite utilizar SDL en cualquier desarrollo sin costes derivados.

Es de código abierto y está escrita en C y, dependiendo de la plataforma destino, en C++ u Objective-C y hay *bindings* para otros lenguajes como C# y Python. Puede considerarse como una alternativa a las señaladas en la propia página de OpenGL para la gestión del interfaz de usuario, puesto que ofrece funciones para la gestión de eventos de usuario de entrada y también servicios para trabajar con los estándares de gráficos OpenGL, OpenGL ES y Vulkan en computadores de escritorio (sobre los gestores gráficos de Windows, macOS, X11 y Wayland), proporcionando un API para la creación de ventanas y un contexto de renderizado sobre ellas.

[3] La interfaz de programación de aplicaciones o API (*application programming interface*) es la declaración de las operaciones y estructuras de datos a las que puede acceder un desarrollador de una cierta biblioteca de funciones.



a) *b)*

Figura 1: SDL: (a) icono (imagen de [3]) y (b) niveles de abstracción para diferentes platformas soportadas por SDL (imagen de [7]).

4 Desarrollo Desarrollo

Se va a partir de un ejemplo existente, dejando de lado las operaciones propias de dibujado de OpenGL, para centrarse en el uso de las funciones de GLUT que se van a reescribir en términos de SDL. Un ejemplo interesante y dinámico es el de *glutplane* [4] (véase Figura 1), realizado por Mark J. Kilgard y otras "*demos*" interesantes del propio Mark se pueden ver en [5].

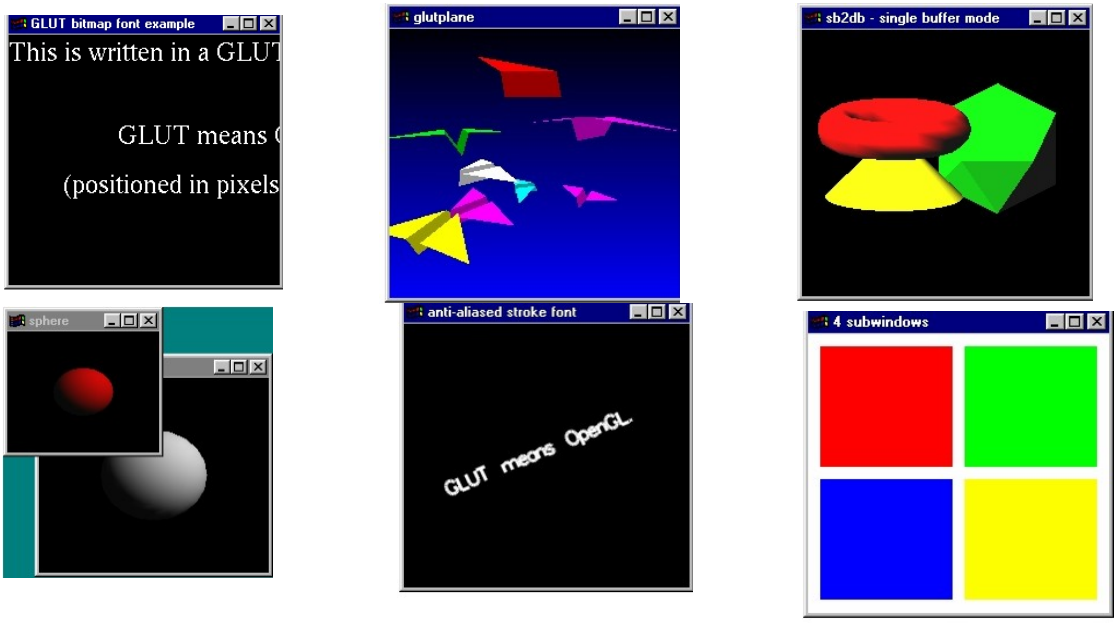


Figura 2: Algunos de los ejemplos de OpenGL con GLUT disponibles en [4] (de izquierda a derecha y de arriba abajo): bitfont, glutplane, sb2db, sphere, stroke y subwind.

En este apartado se aborda:

- Cómo poner en marcha el ejemplo de *glutplane*, el entorno de ejecución y desarrollo.
- Cómo reescribir, en *glutplane*, las acciones de GLUT en términos de SDL.

El ejemplo de *glutplane* se compila con una línea de órdenes que enlace el resultado obtenido con la librería de OpenGL (*libGL*) y la de GLUT (*libGLUT*), además de la librería matemática que necesitamos (*libm*) y, si no hay errores, se podrá ejecutar:

```
$ gcc glutplane.c -o glutplane -lglut -lGL -lm
$ ./glutplane
```

Puede encontrar el ejemplo en el repositorio creado a tal efecto en *GitHub*[4], descárguelo y así podrá seguir la exploración del mismo que propone el resto de este apartado. Así podrá comparar las variaciones que se han introducido en el código original. En general veremos que han desaparecido todas las referencias con el prefijo GLUT a otras similares con SDL como prefijo, pero veámoslas con más detalle.

4.1 Portando el ejemplo de código a SDL

Como se ha dicho, el ejemplo escogido es glut*plane* [4] (véase Figura 3) cuyo código fuente muestra la autoría y la licencia del mismo: "Copyright (c) Mark J. Kilgard, 1994.·This program is freely distributable without licensing fees and is provided without guarantee or warrantee expressed or implied. This program is -not- in the public domain.". Para hacer el paso se han consultado los ejemplos del tutorial de SDL "SDL Library Documentation"[5].

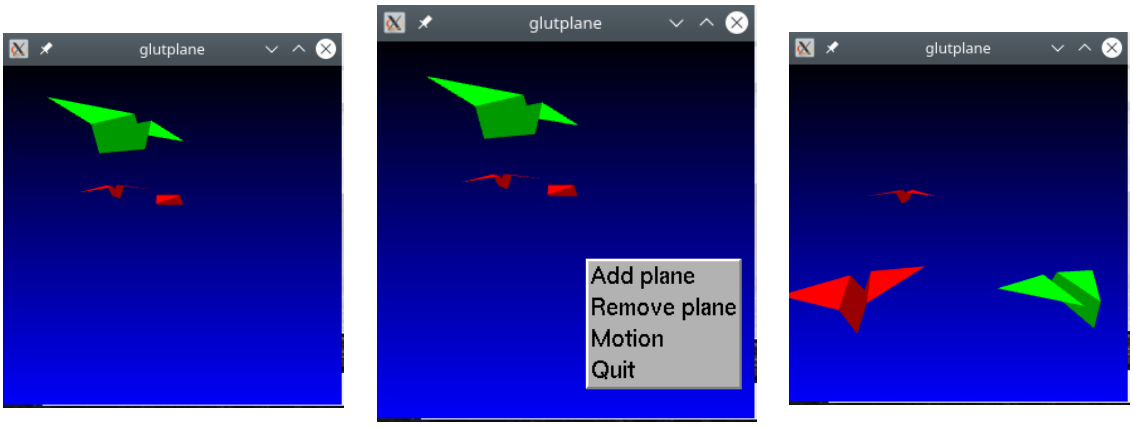


Figura 3: Salida de la versión original de glutplane: de izquierda a derecha, escena inicial, menú asociado al ratón desplegado y objetos en movimiento tras escoger "Motion" en el menú.

Veamos ahora cómo se han introducido los servicios de SDL sustituyendo a los de GLUT, buscando siempre mantener la funcionalidad de la aplicación original al máximo. En los listados que se explican a continuación, se han omitido los bloques de código que se ha considerado menos relevantes, generalmente porque no han cambiado entre las dos versiones. Buscando así acomodarse al espacio disponible para la longitud del artículo.

En el Listado 1 se pueden ver dos bloques:

- El primero, líneas 1 a la 43, muestra la directiva de cabecera que sustituye a GLUT (línea 4), por SDL. Y las funciones de dibujado de OpenGL, líneas 28 a la 44, que se mantienen igual, excepto por que se ha comentado las líneas que hacen uso de las funciones *glutSwapBuffers (en la función* draw*)* o *glutPostRedisplay (en las funciones add_plane*, *remove_plane*, *tick* y *animate)*, que no son necesarias con SDL, ya veremos en el bucle principal cómo se actualiza el contenido de la ventana.
- El segundo bloque, líneas 44 a la 51, contiene constantes y declaraciones propias de SDL (resaltadas en negrita) que son necesarias para el resto del código, de ellas destacaremos

[4] Encontrará el código a que se refiere en este artículo en el repositorio de Github <https://github.com/magusti/OpenGL_examples/SDL_API>.

[5] El capítulo 2 "Graphics and Video: Using OpenGL With SDL" está en la URL <https://www.libsdl.org/release/SDL-1.2.15/docs/html/guidevideoopengl.html>.

un identificador que utilizaremos después para cambiar la forma del cursor en pantalla (***SDL_SYSTEM_CURSOR_ARROW***), el tipo de datos que recoge las propiedades de la ventana en SDL (***SDL_Window***) y una función. (***SDL_Quit***), para cerrar la sesión de la ventana de SDL.

En el Listado 2 se abordan las dos funciones de recepción de eventos que implementan con las funciones del API de SDL las acciones sobre el interfaz de la aplicación:

- Por un lado, la función ***handle_key_down***, líneas 56 a la 80, es la encargada de implementar las acciones que en la versión de GLUT se realizan con la selección de opciones sobre un menú desplegable y que aquí se han reconvertido a atajos de teclado; manteniendo las funciones que son llamadas en el código original para mantener al máximo el paralelismo entre ambas versiones.
- Por otro lado, la función ***process_events***, líneas 90 a la 105, recoge los eventos que gestiona SDL por tipo y los reencamina; con ello podemos ver la capacidad de gestión de SDL. Así, en el caso de que los eventos que reciban se refieran a teclas pulsadas (*SDL_KEYDOWN*) lo reenvía a *handle_key_down* (y se podrían haber utilizado también eventos de tecla mantenida pulsada o tecla soltada, si hubiera sido necesario aquí). También podrían atender eventos relativos a acciones sobre la la ventana de visualización, como lo es el caso del redimensionado de la ventana (*SDL_WINDOWEVENT*) que en este caso hay que enviar al propio OpenGL para que lo procese. Y, por último, si se pulsa en el botón de cerrar la ventana se recibe un *SDL_QUIT* que es atendido liberando los recursos asociados a la ventana de SDL.

En el Listado 3 llegamos al programa principal (función ***main***). En ella podemos diferenciar dos bloques:

- En el primero, entre las líneas 103 y 141, están las inicializaciones. Como se puede observar, la función ***SDL_INIT*** (línea 108) empieza el proceso, encargándose de comprobar si el hardware de la tarjeta gráfica puede aceptar un determinado modo gráfico. Comprobará si puede establecer las propiedades del mismo que hagan que sirva de base a un a aplicación OpenGL, para lo que asigna (líneas 115 a la 119, con ***SDL_GL_SetAttribute***) el número de bits para las componentes R, G y B de color, el tamaño (en bits) de los elementos del *buffer* de profundidad de OpenGL, así como el uso del doble *buffer* para actualizar la escena. Ahora ya puede crear la ventana (línea 121, ***SDL_CreateWindow***) con un texto para la barra de título de la ventana y la propiedad ***SDL_WINDOW_OPENGL***, muy importante en este caso. La configuración de la ventana se completa con la conexión con el servidor OpenGL mediante dos llamadas (líneas 130 y 131, ***SDL_GL_CreateContext*** y ***SDL_CreateRenderer***) que asocian a OpenGL al área dibujable de la ventana y que puede definir esta comunicación en términos de, p. ej., aceleración hardware o de implementación totalmente software del proceso de renderizado de OpenGL. A partir de aquí, ya se pueden utilizar las funciones del API de OpenGL como se ve en las líneas 133 a la 141.

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  #include <math.h>
4.  #include <SDL2/SDL.h> //#include <GL/glut.h>
5.  #include <GL/gl.h>
6.  #include <GL/glu.h>
7.
8.  /* Some <math.h> files do not define M_PI... */
9.  #ifndef M_PI
10. #define M_PI 3.14159265
11. #endif
12. #ifndef M_PI_2
13. #define M_PI_2 1.57079632
14. #endif
15.
16. GLboolean appFuncionant = GL_TRUE;
17. GLboolean moving = GL_FALSE;
18.
19. #define MAX_PLANES 150  // 15 inicialment!!!
20. struct {
21.   float speed;          // zero speed means not flying
22.   GLfloat red, green, blue;
23.   float theta;
24.   float x, y, z, angle;
25. } planes[MAX_PLANES];
26.
27. #define v3f glVertex3f // v3f was the short IRIS GL name for glVertex3f
28. void draw(void) {
29. …   // se omite el resto de código por que no se ha modificado
30. //   glutSwapBuffers(); --> SDL_GL_SwapBuffers() en el bucle ppal.
31. }
32. void tick_per_plane(int i){ ... }
33. // Em las siguientes se han comentado las glutPostRedisplay();
34. void add_plane(void) { ... }
35. void remove_plane(void) { ... }
36. void tick(void) { ... }
37. void animate(void) { ... }
38. //void visible(int state) {
39. // No se muestra porque ha sido totalmente comentada
40. //}
41. // void keyboard(unsigned char ch, int x, int y) {
42. // No se muestra porque ha sido totalmente comentada
43. // }
44. int elIDCursor = SDL_SYSTEM_CURSOR_ARROW; //GLUT_CURSOR_RIGHT_ARROW;
45. int pantallaCompleta = 0;
46. SDL_Window *sdlFinestra;
47. static void aplicacio( int code ) {
48.  // Release the fullscreen mode,... etc.
49.     SDL_Quit( );
50.     exit( code );      /* Exit program. */
51. }
    ...
```

Listado 1: Ejemplo de SDL: sdl2plane.c.

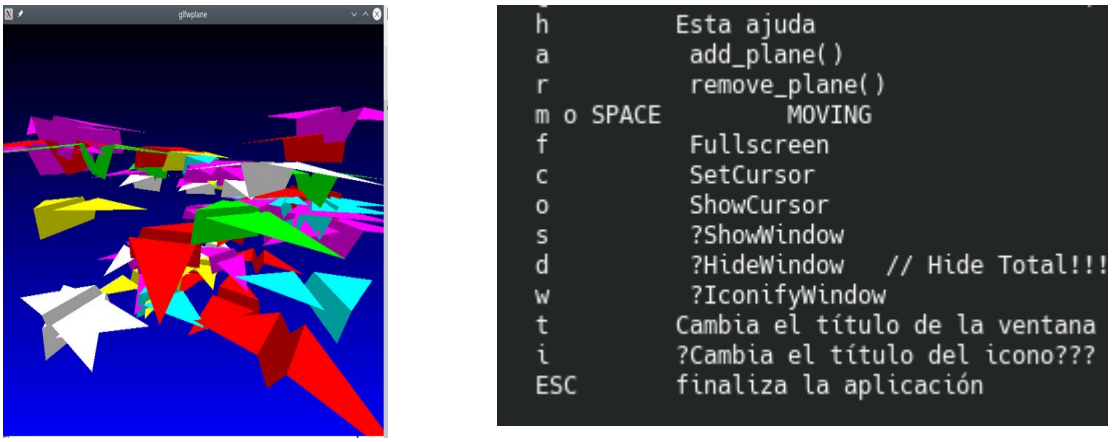
```
...
SDL_Cursor *elCursor;
static void handle_key_down( SDL_Keysym* keysym ) {
    switch( keysym->sym ) {
    case SDLK_ESCAPE: quit_tutorial( 0 ); break;
    case SDLK_a: add_plane(); break;
    case SDLK_r: remove_plane(); break;
    case SDLK_m: moving = !moving; break;
    case SDLK_SPACE:if (!moving) { tick(); } break;
    case SDLK_f:
        pantallaCompleta = !pantallaCompleta;
        if ( pantallaCompleta )
           SDL_SetWindowFullscreen( sdlFinestra,
                                    SDL_WINDOW_FULLSCREEN_DESKTOP );
        else
           SDL_SetWindowFullscreen( sdlFinestra, 0 );
        break;
    case SDLK_t:
        SDL_SetWindowTitle(sdlFinestra, "Canviant el títol!"); break;
    case SDLK_h:
      printf("Ajuda\n\ h\t Esta ajuda\n\ESC\t finaliza.\n");
      break;
    case SDLK_c:
       elCursor = SDL_CreateSystemCursor( elIDCursor++ );
       SDL_SetCursor( elCursor );
       if ( elIDCursor > SDL_SYSTEM_CURSOR_HAND )
         elIDCursor = SDL_SYSTEM_CURSOR_ARROW ;
       break;
    case SDLK_o:
       if ( SDL_ShowCursor( SDL_QUERY ) == SDL_ENABLE )
           SDL_ShowCursor( SDL_DISABLE );
       else SDL_ShowCursor( SDL_ENABLE );
       break;
    default:  break;
    }
}
static void process_events( void ) { // Our SDL event placeholder
    SDL_Event event;                    // Grab all the events off
    while( SDL_PollEvent( &event ) ) { // the queue
      switch( event.type ) {
        case SDL_KEYDOWN: //Handle key presses.
            handle_key_down( &event.key.keysym ); break;
         case SDL_WINDOWEVENT:
          if (event.window.event == SDL_WINDOWEVENT_RESIZED) {
            glViewport(0, 0, event.window.data1, event.window.data2);}
          break;
        case SDL_QUIT: // Handle quit requests (like Ctrl-c).
            SDL_DestroyWindow( sdlFinestra ); quit_tutorial( 0 );
            break;
      }
    }
}
...
```

Listado 2: Ejemplo de SDL: sdl2plane.c (2).

```
   ...
103.int main(int argc, char *argv[]) {
104.    int width = 0, height = 0; // Dimensions of our window
105.    int bpp = 0;               // Color depth in bits of our window
106.    int flags = 0;      // Flags we will pass into SDL_SetVideoMode.
107.    /* First, initialize SDL's video subsystem. */
108.    if( SDL_Init( SDL_INIT_VIDEO ) < 0 ) { // Failed, exit.
109.        fprintf( stderr, "Video initialization failed: %s\n",
110.                         SDL_GetError() );
111.        quit_aplicacio( 1 );
112.    }
113.    width = 640;
114.    height = 480;
115.    SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 5 );
116.    SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
117.    SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
118.    SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
119.    SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
120.    // request that SDL provide us with an OpenGL window
121.    sdlFinestra = SDL_CreateWindow("sdl2plane",
122.                           SDL_WINDOWPOS_UNDEFINED,
123.                           SDL_WINDOWPOS_UNDEFINED,
124.                           width, height,
125.                           SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE);
126.    if (sdlFinestra == NULL) {
127.      fprintf(stderr, "could not create window: %s\n", SDL_GetError());
128.      return 1;
129.    }
130.    SDL_GL_CreateContext( sdlFinestra );
131.    SDL_Renderer *renderer = SDL_CreateRenderer(sdlFinestra , -1, 0);
132.    // setup OpenGL state
133.    glClearDepth(1.0);
134.    glClearColor(0.0, 0.0, 0.0, 0.0);
135.    glMatrixMode(GL_PROJECTION);
136.    glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 20);
137.    glMatrixMode(GL_MODELVIEW);
138.    srandom(getpid()); //  add three initial random planes */
139.    add_plane();
140.    add_plane();
141.    add_plane();
142.//   glutMainLoop();   //start event processing
143.    while( 1 ) {
144.      process_events( );         // Process incoming events.
145.      draw(); // Draw the screen. */
146.      if ( moving ) { animate(); }
147.      SDL_GL_SwapWindow( sdlFinestra );
148.      SDL_Delay( 25 ); //ms
149.    }
150.  return 0;             /* ANSI C requires main to return int. */
151.}
```

Listado 3: Ejemplo de SDL: sdl2plane.c (y 3).

- El segundo bloque del Listado 3 lo constituye el bucle principal, entre las líneas 142 y la 150. En este se observan tres pasos:
  - El primero, es el de recoger lo eventos de interacción del usuario con la interfaz gráfica de la aplicación y se encaminan hacia la función *process_event*s, básicamente realizada con órdenes de SDL.
  - El segundo es el de actualizar el dibujo que se mostrará en pantalla, las funciones draw y animate (líneas 145 y 146) hará lo propio empleando órdenes de OpenGL.
  - Y el tercer paso, es el que actualiza estos cambios en la pantalla (líneas 147 y 148) con ***SDL_GL_SwapWindow***, y con ***SDL_Delay*** el intervalo de refresco entre un cuadro y el siguiente (a 25 ms, estamos hablando de unos 40 cuadros por segundo que es una tasa aceptable).



a) *b)*

Figura 4: Versión revisada de sdl2plane: (a) salida gráfica y (b) teclas y acciones definidas para la aplicación..

Llegados a este punto solo queda describir cómo se compila el ejemplo de *sdl2plane*. Atención, podría ser que se tuvieran instaladas las dos versiones de SDL (SDL 1.2[6] y SDL 2). Para pasar los parámetros adecuados al compilador para utilizar SDL2, se puede hacer con la orden *sdl2-config* o con *pkg-config*. Se puede confirmar y realizar con ambas de forma análoga:

```
$ sdl2-config --version
2.0.10
$ pkg-config sdl2 –modversion
2.0.10
```

Porque, p. ej., se propone para compilar la aplicación ejecutar la orden

```
$ gcc sdlplane.c   -o sdlplane `pkg-config sdl2 --cflags --libs` -lGLU -lGL -lm
```

donde se puede ver que hemos sustituido las referencias a GLUT por SDL2 y que nos ofrece una salida compatible con la versión de GLUT, véase la Figura 4a. Las acciones de menú de la versión de GLUT han sido tratadas como eventos asociados a las pulsaciones de las teclas que muestra la Figura 4b.

[6] En el caso de usar SDL 1.2 se haría: `$ gcc sdlplane.c -o sdlplane -lSDL -lGLU -lGL`

5 Conclusión y cierre

Como el lector habrá podido observar al seguir el artículo, si lo ha ido comprobando conforme lo leía, hemos explorado las posibilidades del API de *SDL*, *a partir de un e*jemplo básico existente al que se le han sustituido las funcionalidades de GLUT por las de SDL. ¡Y hemos ampliado el número de aviones, véase la Figura 4, y algunas nuevas funcionalidades también! No es posible desplegar un menú en SDL, pero quizá lo compensa la forma de gestionar el flujo de eventos que es más obvia para algunos desarrolladores y, en concreto, para adaptarlo a plataformas como las videoconsolas, que son muy particulares en estas acciones.

Quiero aprovechar para animar al lector a probar alguna otra función de SDL que están recogidas en la documentación en línea como la detección de los eventos de ratón, de las pantallas táctiles, del *gamepad* (posición, botonera y la vibración) y el audio como complementos interesantes para aportar más características multimedia a este tipo de aplicaciones[7]. Espero que, a estas alturas, tenga ejecutándose el código propuesto y comprobando que puede ver volar los aviones de papel. Puede descargar el ejemplo del repositorio creado a tal efecto en *GitHub*[8] y no cierre este documento sin haberlo comprobado antes. ¡¡ÁNIMO!!

6 Bibliografía y referencias

[1] Kilgard, M. J. (1996). GLUT - The OpenGL Utility Toolkit. Silicon Graphics, Inc. Disponible en: <https://www.opengl.org/resources/libraries/glut/glut_downloads.php>.

[2] Kilgard, M. (1994). OpenGL and X, Column 1: An OpenGL Toolkit. The X Journal. Disponible en <https://www.opengl.org/resources/libraries/glut/glut.column1.ps.gz>.

[3] Sitio web de SDL. Disponible en: <https://www.libsdl.org/>

[4] OpenGL examples. Disponible en: <https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html> .

[5] OpenGL demos. Disponible en: <https://www.opengl.org/archives/resources/code/samples/glut_examples/demos/demos.html>.

[6] Khronos Group. Getting Started. OpenGL Wiki-. Disponible en: <https://www.khronos.org/opengl/wiki/Getting_Started>.

[7] Wikipedia - Simple DirectMedia Layer. Disponible en <https://en.wikipedia.org/wiki/Simple_DirectMedia_Layer>.

[7] Las encontrará en <https://wiki.libsdl.org/SDL2/APIByCategory >.

[8] Encontrará el código a que se refiere en este artículo en el repositorio de Github <https://github.com/magusti/OpenGL_examples/OpenGL_SDLI>, como ya se ha mencionado anteriormente.