UNIVERSITAT POLITÈCNICA DE VALÈNCIA

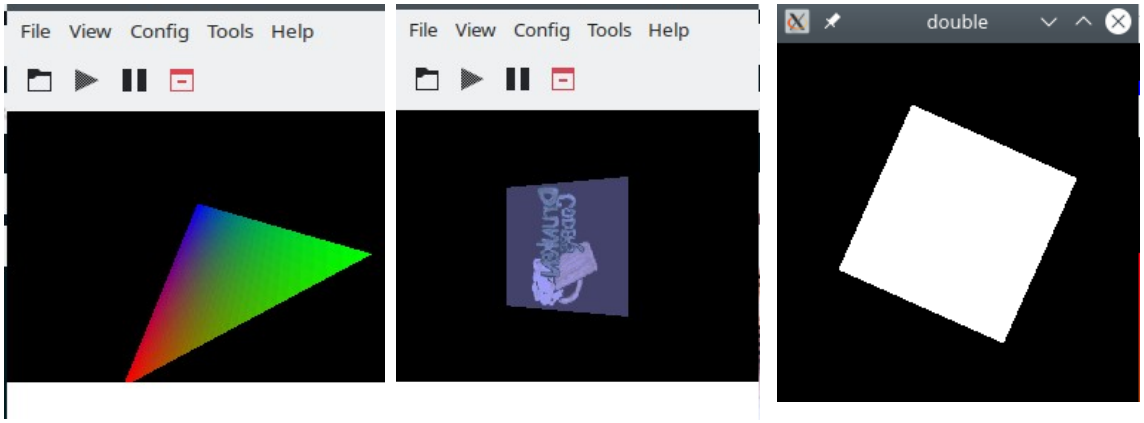
Introducción al desarrollo de aplicaciones inmersivas con OpenGL clásico

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

En diferentes tipos de aplicaciones utilizamos interfaces gráficas que ofrecen un contenido visual tridimensional para proporcionar al usuario la información con el nivel de atracción y de detalle que esperamos le resulte interesante e impresionante.

Sucede así desde los desarrollos para videojuegos a los recientes usos de tecnologías **multimedia inmersivas** que han dado pie a aplicaciones de realidad virtual (RV) o aumentada (RA). En 2D o en 3D, estas aplicaciones demandan no solo la salida gráfica sino también complementarla con el análisis de imágenes procedentes de una o varias cámaras, la reproducción del audio (posiblemente en 3D) e, incluso, la síntesis o el reconocimiento de voz. La Figura 1 muestra ejemplos básicos en la consola NDS y el computador de escritorio.



a) *b)*

Figura 1: Ejemplos de OpenGL sobre: (a) NDS y (b) X Window en Linux.

En este artículo se va a presentar un ejemplo básico de aplicación sobre OpenGL en modo clásico para abordar la salida gráfica, 2D y 3D, de una aplicación multiplataforma. Este modo de desarrollo de una aplicación sobre OpenGL se puede utilizar desde plataformas como videoconsolas (como la NDS que no disponía de una GPU) hasta ejemplos que podemos encontrar sobre OpenGL ES[1] en otras plataformas, como el computador de escritorio o los "mobile systems" (que abarca diferentes equipos empotrados como videoconsolas, teléfonos, coches o aviación) que tienen limitada la capacidad de recursos, mayormente en cuanto a potencia energética se refiere.

Se pondrá especial énfasis en ver **cómo plantear el bucle principal** de gestión de eventos, desde el punto de vista de OpenGL como actor principal, frente al punto de vista de una aplicación multimedia que, además de interfaz gráfico, ha de tomar en cuenta otro tipo de eventos para decidir cómo evoluciona la aplicación.

El trabajo mostrado se ha realizado sobre el sistema operativo Linux, pero los resultados son portables a otras plataformas.

[1] Para ampliar sobre OpenGL ES puede consultar en <https://www.khronos.org/opengles/>.

2 Objetivos

Una vez que el alumno haya leído el documento y explorado la aplicación de ejemplo que se describe, será capaz de:

- Desplegar la aplicación de ejemplo en su propio computador.
- Describir los elementos básicos de una aplicación basada en OpenGL, funcionando sobre la premisa de ejecutarse sin necesidad de una GPU.
- Examinar la gestión de eventos para poder tomar el control de la evolución de la aplicación.

3 Introducción

En el contexto de aplicaciones multimedia, el uso de OpenGL como especificación para el uso de gráficos (principalmente en 3D) es un estándar desde finales de 1990 que se ha ido actualizando conforme se sucedían los cambios en el hardware gráfico de los computadores y especialmente con la introducción de las GPUs [1], hoy tenemos aplicaciones interactivas complejas con gráficos 3D de alta resolución, en tiempo real y a frecuencias superiores a los treinta cuadros por segundo (o *frames per second*, fps), llegando habitualmente a valores superiores a los 60 fps.

Para conseguir estos valores, el modo de proceder de OpenGL ha evolucionado y podemos encontrar, entre la cantidad de ejemplos disponibles en libros y en la red, la coletilla de "**OpenGL clásico**" para diferenciar el cambio de proceder que se inicia [2] en la versión 2.0 con la inclusión del soporte de GPUs y el lenguaje GLSL[2] y que termina en la versión OpenGL 3.2 dando lugar al "OpenGL moderno".

Este modo de desarrollo, de lo que se ha dado en denominar ***OpenGL clásico*** (también denominado **immediate mode**) es el que se ha venido utilizando desde los ejemplos originales de OpenGL[3] de SGI hasta la versión 3.0 de OpenGL. Está basado en una secuencia fija de operaciones de renderizado[4] [5] o "**Fixed Pipeline**. Desde entonces, su uso se ha marcado como descontinuado (*deprecated*) en las sucesivas versiones de OpenGL, pero no por ello ha dejado de utilizarse. Aunque es menos eficiente, sigue siendo el modo preferido por quienes se inician en este campo por su menor complejidad de funcionamiento.

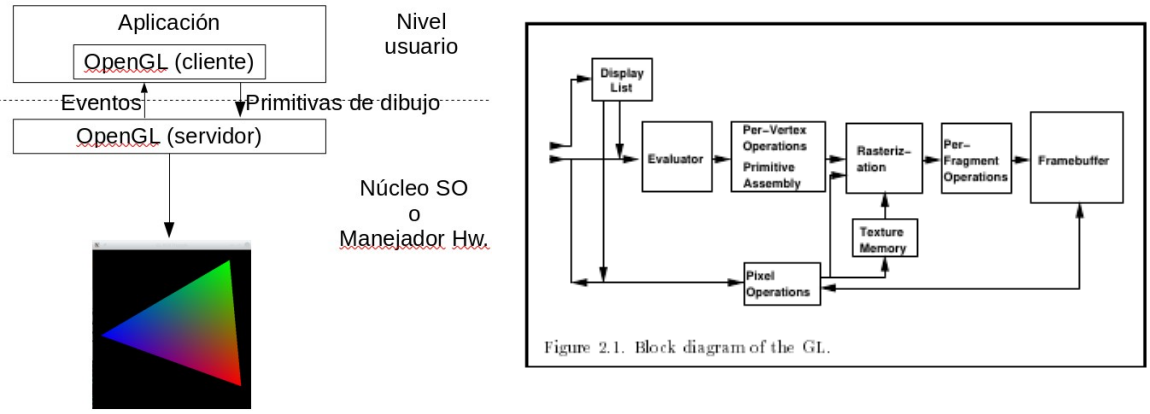
Las operaciones de renderizado de OpenGL se reparten, como muestra la Figura 2a, entre dos niveles: el cliente de OpenGL y el servidor de OpenGL. Cuando una aplicación inicializa un contexto gráfico compatible con OpenGL está trabajando a nivel de usuario utilizando el API de OpenGL para lanzar las primitivas gráficas que describen la escena 3D a pintar en la ventana y esperará recibir eventos, si los ha registrado o declarado, para

[2] Puede leer sobre qué es GLSL en <https://www.khronos.org/opengl/wiki/Core_Language_(GLSL)>.

[3] "OpenGL – Examples" disponible en la URL <https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html>.

[4] Este término, en el contexto de gráficos, describe el proceso de descripción y proyección para pintar una escena 3D en una pantalla 2D.

poder modificar su flujo de operaciones a raíz de la interacción con el usuario. El servidor de OpenGL, que puede estar a nivel del manejador (o *driver*) de la tarjeta gráfica ofreciendo toda la aceleración que esta ofrece; o a nivel de núcleo del Sistema Operativo (SO), donde proporcionar una implementación totalmente software de OpenGL. De forma equivalente, a nivel de gráficos por computador se habla de una "tubería", Figura 2b, por la que avanzan los cálculos desde las primitivas gráficas que genera la aplicación, hasta el contenido de cada píxel del área de dibujo de la ventana o pantalla (*frambuffer*).



a) *b)*

Figura 2: Comunicación en OpenGL: (a) eventos a nivel de la aplicación y (b) a nivel de operaciones gráficas (rendering pipeline) [9].

3.1 Instalación y compilación

Para las diferentes plataformas se puede encontrar ampliamente comentado en el sitio Web de Khronos[5], pero como se acaba de comentar al respecto de la comunicación servidor/cliente de OpenGL y, dado que OpenGL solo es una especificación[6], no existe un producto único (una única librería o SDK) que haya que instalar:

- En Linux tenemos Mesa, que ofrece tanto implementación software como aceleración gráfica.
- En macOS, Apple incluye su propia implementación de OpenGL y ha acabado desarrollando todo un nivel que se denomina Metal.
- En MS Windows, se basa en el uso de los manejadores en que los fabricantes de tarjetas gráfica (como NVIDIA o AMD) hacen sus propias implementaciones encaminadas a aprovechar al máximo la funcionalidad de su hardware. Y, también, compite con ellos con su propio nivel gráfico a nivel de SO (DirectX).

En cada caso, actualizar el componente que implementa OpenGL es suficiente para el usuario de aplicaciones, pero no para el desarrollador. En el caso presente, sobre Linux[7], se puede utilizar la orden:

---

[5] En el apartado "Getting Started" <https://www.khronos.org/opengl/wiki/Getting_Started> y en el de FAQ <https://www.khronos.org/opengl/wiki/FAQ>,

[6] Véase en <https://www.khronos.org/opengl/wiki/FAQ#Is_OpenGL_Open_Source?>.

```
$ sudo apt-get install build-essential libglu1-mesa-dev freeglut3-dev \
mesa-common-dev
```

Después de que se hayan instalado las bibliotecas de desarrollo necesarias, para obtener información sobre la disponibilidad de OpenGL, se puede comprobar con la orden

```
$ glxinfo -B
name of display: :0
display: :0  screen: 0
direct rendering: Yes
Extended renderer info (GLX_MESA_query_renderer):
    Vendor: Intel (0x8086)
    Device: Mesa Intel(R) UHD Graphics 630 (CML GT2) (0x9bc8)
    Version: 21.2.6
    Accelerated: yes
    Video memory: 3072MB
    Unified memory: yes
    Preferred profile: core (0x1)
    Max core profile version: 4.6
    Max compat profile version: 4.6
    Max GLES1 profile version: 1.1
    Max GLES[23] profile version: 3.2
OpenGL vendor string: Intel
OpenGL renderer string: Mesa Intel(R) UHD Graphics 630 (CML GT2)
OpenGL core profile version string: 4.6 (Core Profile) Mesa 21.2.6
OpenGL core profile shading language version string: 4.60
OpenGL core profile context flags: (none)
OpenGL core profile profile mask: core profile

OpenGL version string: 4.6 (Compatibility Profile) Mesa 21.2.6
OpenGL shading language version string: 4.60
OpenGL context flags: (none)
OpenGL profile mask: compatibility profile

OpenGL ES profile version string: OpenGL ES 3.2 Mesa 21.2.6
OpenGL ES profile shading language version string: OpenGL ES GLSL ES 3.20
```

Esta es una versión reducida de las capacidades de *glxinfo*, pero es suficiente para hacerse una idea de lo que puede esperar ver en su equipo: la identificación del renderizador (en este caso Mesa), el hardware disponible (en este caso una tarjeta Intel UHD Graphics 630) y las versiones de OpenGL, OpenGL ES y GLSL soportadas.

[7] Inspirado en “How to Install OpenGL on Ubuntu Linux” <http://www.codebind.com/linux-tutorials/install-opengl-ubuntu-linux/> y “Cómo instalar Mesa (OpenGL) en Linux Mint” <https://es.wikihow.com/instalar-Mesa-(OpenGL)-en-Linux-Mint>.

4 Desarrollo

Ahora que ya se dispone de una instalación de OpenGL y se ha comprobado su disponibilidad viendo sus características, se va a mostrar una aplicación que es el resultado de juntar un ejemplo del libro rojo [6] (referencia obligada al hablar de OpenGL), que pinta un triángulo estático en la ventana y otro de [7] (una de las referencias más interesantes que están disponibles en la red), que describe cómo animar un objeto sencillo haciéndolo rotar sobre su centro de gravedad continuamente.

El resultado, Figura 3, es un triángulo cuyo interior muestra la gradación de colores entre los tres primarios situados en sus vértices y cuya posición cambia de forma continua al recibir un clic del ratón en cualquier posición de la ventana.

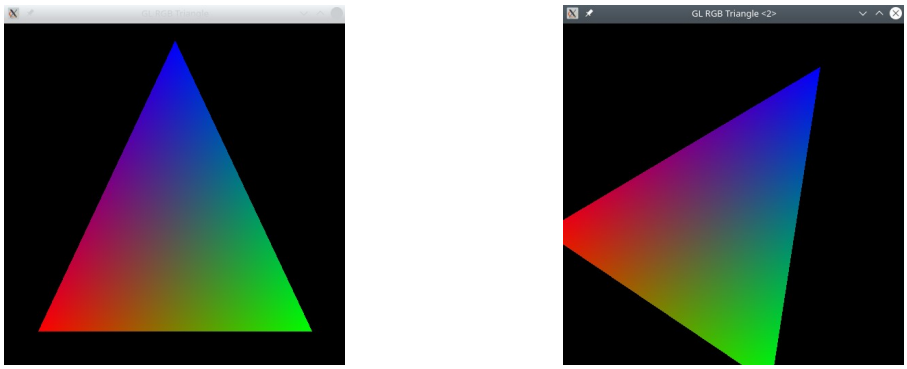


Figura 3: Capturas de la ejecución del ejemplo first-triangle_animat.

El código que lleva a cabo esta aplicación se puede encontrar al completo en Github [8]. A continuación, se muestra en los listados 1 y 2 su contenido principal, para después ver cómo modificar el bucle principal y, así, podrá reaccionar la aplicación en base a otros eventos que puede requerir una aplicación inmersiva y que no son únicamente los propios de la gestión gráfica. En el resto de este punto se va a recorrer el contenido de esos listados para presentar las instrucciones de OpenGL que aparecen, junto a los comentarios encontrados en los ficheros de código tomados de referencia.

El Listado 1 muestra la cabecera mínima necesaria para una aplicación OpenGL, junto con la GLUT que implementará el cliente de OpenGL. Además, también aparecen las tres funciones que se registrarán para atender a los eventos: *display*, *spinDisplay* y *mouse*. En estas funciones se pueden reconocer las líneas con el prefijo *gl* o *GL* que son las propias de OpenGL (la parte del servidor) y *glut*, que lo son del nivel de interfaz de pantalla GLUT (el cliente de OpenGL).

En el Listado 2 vemos el resto de la aplicación, el programa principal empieza creando la parte de gestión de interfaz (el cliente de OpenGL, en este caso con GLUT), línea 49 a la 54, que conducirá a la creación de una ventana, con un determinado tamaño en píxeles y posicionada en unas coordenadas de pantalla. Como curiosidad, hay que hacer notar que OpenGL trabaja con un sistema de coordenas 3D centrado en (0,0,0) que no tiene porqué coincidir con la esquina superior izquierda en la que, habitualmente, se empieza a contar la posición de los píxeles en una imagen en 2D, esto es el (0,0).

```
1.  #include <GL/gl.h>
2.  #include <GL/glut.h>
3.
4.  static GLfloat spin = 1.0; //Para la animación
5.
6.  void display() {  // Display function will draw the image.
7.      glClearColor( 0, 0, 0, 1 );  // (In fact, this is the default.)
8.      glClear( GL_COLOR_BUFFER_BIT );
9.
10.     glRotatef(spin, 0.0, 0.0, 1.0);
11.     glBegin(GL_TRIANGLES);
12.     glColor3f( 1, 0, 0 ); // red
13.     glVertex2f( -0.8, -0.8 );
14.     glColor3f( 0, 1, 0 ); // green
15.     glVertex2f( 0.8, -0.8 );
16.     glColor3f( 0, 0, 1 ); // blue
17.     glVertex2f( 0, 0.9 );
18.     glEnd();
19.     glutSwapBuffers(); //Required to copy color buffer onto the screen.
20. }
21. void spinDisplay(void) {
22.    spin = spin + 0.001;
       if (spin > 360.0) spin = spin - 360.0;
23.    glutPostRedisplay();
24. }
25. void reshape(int w, int h) {
26.    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
27.    glMatrixMode(GL_PROJECTION);
28.    glLoadIdentity();
29.    glOrtho(-50.0, 50.0, -50.0, 50.0, 1.0, 1.0);
30.    glOrtho(-50.0, 50.0, -50.0, 50.0, 1.0, 1.0);
31.    glMatrixMode(GL_MODELVIEW);
32.    glLoadIdentity();
33. }
34. void mouse(int button, int state, int x, int y) {
35.    switch (button) {
36.       case GLUT_LEFT_BUTTON:
37.          if (state == GLUT_DOWN)
38.             glutIdleFunc( spinDisplay );
39.          break;
40.       case GLUT_MIDDLE_BUTTON:
41.          if (state == GLUT_DOWN)
42.             glutIdleFunc( NULL );
43.          break;
44.       default:
45.          break;
46.    }
47. }
    ..
```

Listado 1: Ejemplo de aplicación bajo el modelo de OpenGL clásico: first-triangle__animat.c.

A partir de ahora la ventana ya recibe eventos, acciones del sistema que le atañen a ella y para que pueda gestionarlos se registran funciones que serán invocadas por OpenGL cuando el evento sea de un determinado tipo. A estas funciones se las llama *callback* porque son la respuesta a ciertos eventos y hay que registrarlas. Así, cada una se encarga de:

- La línea 55 muestra cómo se registra una función para responder al evento de redibujado. Esta es la función mínima que cabe esperar de una aplicación de dibujo, la encargada de pintar en pantalla, que es llamada cada cierto tiempo o cuando es puesta la aplicación en primer plano en el escritorio.
- La línea 56 establece una función que será invocada cuando cambie la geometría de la ventana, porque el usuario ha utilizado los controles de la barra de título de la ventana o porque ha "estirado" la misma, por ejemplo, manipulando los bordes con el ratón.
- La línea 57 asigna una función que será ejecutada cuando se mueva el ratón o se pulse alguno de sus botones.

```
    ...
48. int main( int argc, char** argv ) {
49.     glutInit(&argc, argv);
50.     glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
51.     glutInitWindowSize(500,500);
52.     glutInitWindowPosition(100,100);
53.     glutCreateWindow("GL RGB Triangle");
54.
55.     glutDisplayFunc(display);
56.     glutReshapeFunc(reshape);
57.     glutMouseFunc(mouse);
58.
59.     glutMainLoop();
60.     return 0;
61. }
```

Listado 2: Ejemplo de aplicación bajo el modelo de OpenGL clásico: first-triangle__animat.c (continuación).

Por otro lado, la línea 59 invoca a la función que cede el control a OpenGL (*glutMainLoop)*, el servidor de OpenGL asociado a la ventana toma el control de la aplicación e irá distribuyendo el mismo entre su gestión interna del renderizado en pantalla con la llamada a las funciones de *callback* registradas. Esta función no termina hasta que el usuario cierra la ventana y, como no hay más instrucciones, será también la finalización de la aplicación

Y ya solo falta por saber cómo compilar la aplicación, por ejemplo en Linux, con C/C++ (*gcc* o *g++*), hay que enlazar la aplicación con OpenGL (a través de la implementación de Mesa en este caso) y, si se utiliza el API de GLUT para la gestión de eventos, con una orden como:

```
$ gcc ficheroFuente.c -o ficheroEjecutable -lGL -lglut
```

5 Tomando el control

En la versión que se ha descrito en el apartado 4 Desarrollo se ha visto el punto de vista de OpenGL como actor principal: es la prioridad mostrar una escena gráfica en la ventana, atendiendo, cuando es posible, a la interacción del usuario a través del teclado y/o el ratón.

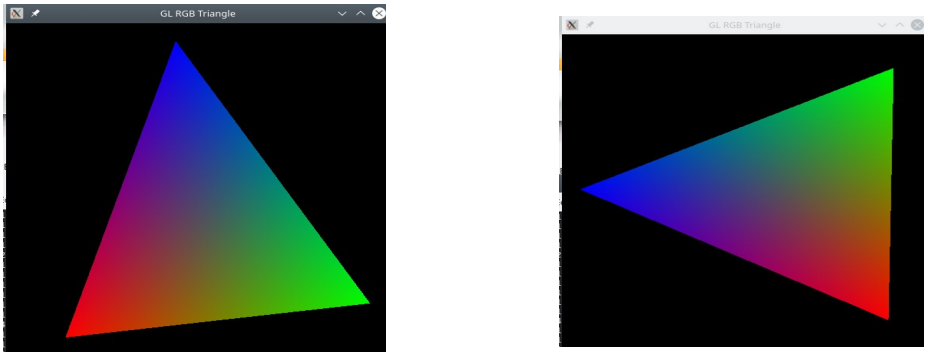


Figura 4: Dos instantes de la ejecución de first-triangle_senseMainLoop.

Ahora, se pondrá especial énfasis en ver **cómo plantear el bucle principal** de gestión de eventos, desde el punto de vista de una aplicación inmersiva que ha de tomar en cuenta otro tipo de eventos para decidir cómo ha de evolucionar la aplicación.

Esta versión del código del anterior ejemplo que se ha llamado *first-triangle_senseMainLoop.c* se puede ver en el Listado 3 y la ejecución del mismo en la Figura 4, aunque por brevedad de la exposición solo se muestra la parte que ha cambiado.

```
1.  int main( int argc, char** argv ) {  // Initialize GLUT and
2.    glutInit(&argc, argv);
3.    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
4.    glutInitWindowSize(500,500);
5.    glutInitWindowPosition(100,100);
6.    glutCreateWindow("GL RGB Triangle - MainLoopEvent");
7.    //glutDisplayFunc(display);  redrawn.
8.    //glutReshapeFunc(reshape);
9.    //glutMouseFunc(mouse);
10.   //
11.   //glutMainLoop(); // Run the event loop!  Does not return.
12.   while( 1 ) {
13.     glutMainLoopEvent();
14.
15.     display();
16.     spinDisplay();
17.
18.     glutPostRedisplay();
19.   }
20.  return 0;
21. }
```

Listado 3: Modificación del programa principal para gestionar el control de la aplicación: first-triangle_senseMainLoop.c.

Se observará que se han quitado todas las funciones registradas, líneas 7 a 9, ahora se hará directamente desde el código del bucle principal.

La gestión de eventos ya no la dirige el servidor de OpenGL, se ha comentado la línea 11, y se ha incluido un bucle, líneas 12 a 19, para realizar la secuencia de pedir al servidor de OpenGL que actualice los eventos (línea 13, aunque también se podría haber llevado a cualquier otro punto de este bucle e, incluso es habitual verla como la última instrucción del mismo), pintar explícitamente en pantalla (línea 15), gestionar las variables internas

de la aplicación (línea 16) y pedir al servidor de OpenGL que actualice el contenido de la pantalla (línea 18).

Hay que hacer notar que aquí se puede gestionar la reproducción de un audio o tomar una captura de vídeo desde la cámara para realizar un cálculo que decida cómo evoluciona la aplicación. Pero no se puede preguntar por qué evento (de teclado o ratón) ha podido suceder sin registrar las funciones de *callback* de nuevo (así sucede con GLUT, freeglut u openglut). Si se necesita, se ha reimplentar esta parte con GLFW[8] o SDL[9].

6 Conclusión y cierre

A lo largo de este objeto de aprendizaje se ha explorado la forma de trabajo de aplicaciones bajo OpenGL funcionando en modo clásico. Hemos revisado un ejemplo de código con algunas funciones que permiten el dibujado y la interacción con el usuario y se ha ofrecido una alternativa al bucle de gestión de eventos donde la aplicación toma las decisiones de cuándo y cómo avanzar.

Sería interesante ver esta misma realización desde el punto de vista del OpenGL "moderno", para comparar las dos grandes versiones de OpenGL. Para comprobar que realmente has aprendido deberías descargar el código y comprobar tú mismo lo que se dice en este artículo y, como no, dejar paso a tu creatividad ¡¡ÁNIMO!!

7 Bibliografía

[1] Kirk, D. B., Hwu, W. W. (2013). Chapter 2 - History of GPU Computing, Programming Massively Parallel Processors (Second Edition). Morgan Kaufmann. ISBN 9780124159921, Disponible en la URL <https://doi.org/10.1016/B978-0-12-415992-1.00002-X>.

[2] History of OpenGL. (2022). OpenGL Wiki. Disponible en la URL <http://www.khronos.org/opengl/wiki_opengl/index.php?title=History_of_OpenGL&oldid=14895 >.

[3] Woo, M., Neider, J. y Davis, T. (1997). The OpenGL Programming Guide. Disponible en la URL <http://www.glprogramming.com/red/chapter01.html>.

[4]. Eck, D. J. (2021). Introduction to Computer Graphics. Disponible en la URL <https://math.hws.edu/graphicsbook/source/glut/>.

[5] Chris X Edwards. Fixed vs. Programmable Pipeline or Older vs. Newer. Programming OpenGL On--Linux. Disponible en la URL <https://xed.ch/help/opengl.html#py>.

[6] OpenGL Programming Guide. The Official Guide to Learning OpenGL, Version 1.1. Sitio web del libro de Woo, M., Neider, J., Davis T. y el OpenGL Architecture Review Board. (1997). Disponible en la URL <http://www.glprogramming.com/red/chapter01.html>.

[7] Eck, D. J. (2021). Introduction to Computer Graphics. Version 1.3. Disponible en la URL <https://math.hws.edu/graphicsbook/>.

[8] Agustí-Melchor, M. (2023). OpenGL clasic. Repositorio en Github del código de ejemplo. Disponible en la URL <https://github.com/magusti/OpenGL_examples/OpenGL_clasic> .

[8] Véase el sitio web de GLFW "An OpenGL Library" <http://www.glfw.org/>.

[9] Que ofrece además soporte para las otras tareas de acceso al audio o a la imagen. Véase al respecto en su sitio web <https://wiki.libsdl.org/SDL2/>.

[9] Segal, M., Akeley, K. y Frazier, C.. (1994). The OpenGL Graphics System: A Specification (Version 1.0). Disponible en la URL <https://khronos.org/registry/OpenGL/specs/gl/glspec10.pdf>.