



# Lecciones de OpenGL.

## Capítulo 2: Animando nuestra escena.

## Introducción.

---

En este capítulo vamos a aprender lo necesario para crear movimiento en nuestras aplicaciones gráficas. Empezare dandote una mala noticia: OpenGL solo puede pintar imágenes fijas. Pero también te puedo dar una buena noticia: puede hacerlo a un ritmo condenadamente rápido. Tan rápido que puede llegar a engañar al ojo humano haciendo que las imágenes fijas pasadas a gran velocidad, den una impresión de movimiento.

En este capítulo vamos a tratar el doble buffer, la función IDLE de glut y la función `glutPostRedisplay()`.

## Pintando una tetera.

---

Lo primero de vamos a hacer es cambiar el cubo de la lección anterior por una tetera. Lo bueno de la tetera es que siempre podremos apreciar cual es su parte posterior y anterior. Para hacerlo usaremos la función del glut `glutWireTeapot` o `glutSolidTeapot`, dependiendo de si queremos pintar un cuerpo sólido o solo sus líneas.

Puedes probar el código completo aquí para hacerte una idea:

```
1.
2.#include <cstdlib>
3.#include <GL/glut.h>
4.
5.
6.void cambiarTamanyo(int w, int h) {
7.
8.    //evitar la division por cero
9.    if(h == 0)
10.        h = 1;
11.
12.    float ratio = 1.0* w / h;
13.
14.    //seteamos la matriz de rproyeccion
15.    glMatrixMode(GL_PROJECTION);
16.    glLoadIdentity();
17.
18.    // Ponemos el viewport ocupando todo la superficie de la
    ventana
19.    glViewport(0, 0, w, h);
20.
21.    // Ponemos la opcion de perspectiva. No usamos la opcion de
    vista ingenieril u ortografica.
22.    gluPerspective(45,ratio,1,1000);
23.    glMatrixMode(GL_MODELVIEW);
24.
25.}
26.
```

```

27. void renderScene(void) {
28.     //borramos el buffer del color
29.     glClear(GL_COLOR_BUFFER_BIT);
30.
31.     glLoadIdentity();
32.     gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, -1.0, 0.0f, 1.0f, 0.0f);
33.
34.     static float xRot = 0.0f;
35.
36.     //rotamos la tetera un grado por frame
37.     xRot += 1.0f;
38.     glRotatef(xRot, 0.0f, 1.0f, 0.0f);
39.     //pintamos la tetera
40.     glutWireTeapot(1.0);
41.
42.     //le decimos a opengl para que pinte todo lo que le hemos
    enviado
43.     glFlush();
44. }
45.
46. int main(int argc, char **argv) {
47.     glutInit(&argc, argv);
48.     glutInitDisplayMode(GLUT_DEPTH | GLUT_SINGLE | GLUT_RGBA);
49.     glutInitWindowPosition(100, 100);
50.     glutInitWindowSize(320, 320);
51.     glutCreateWindow("Tutorial 3D. Animacion");
52.     glutDisplayFunc(renderScene);
53.
54.     glutReshapeFunc(cambiarTamanyo);
55.
56.     glutMainLoop();
57.     return EXIT_SUCCESS;
58. }

```

Las líneas 34 – 39 del código son un mecanismo para girar la tetera. Se llama transformación y lo veremos en posteriores capítulos.

Copia el código en un proyecto de glut y compilalo. Debería aparecer algo así:



Lo primero que vamos a introducir en el código es una manera de ver aproximadamente cuando se pinta la tetera (de hecho, es cuando se pinta la escena entera).

Vamos a incluir en la cabecera lo siguiente:

```
#include <iostream>

using namespace std;
```

Dentro de la función `renderScene`, vamos a añadir el siguiente código:

```
static int countFrames = 0;
cout << "Pintando el cuadro " << countFrames++ << endl;
```

Si ahora minimizas la pantalla o la redimensionas veras que en la consola de texto que se abre detras de la aplicación aparece el texto con la cuenta de cuadros pintados. Date cuenta, además, de que si no tocamos nada, ese mensaje no aparece constantemente.

La animación consiste en pintar constantemente, aunque no pase nada. Durante toda la vida del programa, tenemos que llamar constantemente a la función `renderScene` sin parar.

Así que manos a la obra: lo primero que vamos a hacer es un sistema para llamar a la función `renderScene` muchas veces. Si te fijas, en el `main`, tenemos una función llamada **`glutDisplayFunc()`**, a la que le pasamos `renderScene` como callback function. Al hacer esto, le estamos diciendo a `glut` que cada vez que quiera repintar, llame a `renderScene`.

Existe otra función en `glut`, llamada `glutPostRedisplay()` que le dice a `glut` que fuerce un repintado.

Vamos a hacer un experimento. Si al final de `renderScene` ponemos `glutPostRedisplay()`, veras como en la consola los numeros de cuadros pintados se disparan. Esto sin embargo, no es una buena practica puesto que **`renderScene`** llama a **`glutPostRedisplay`** y **`glutPostRedisplay`** llama a **`renderScene`** con lo que ninguna de las dos funciones acaba nunca pudiendo agotar los recursos de la maquina. Cierra la ventana y borra **`glutPostRedisplay()`**.

## La función idle

---

La función idle se llama en realidad `glutIdleFunc`, se llama en el main, como el resto de las funciones que reciben callback y se le pone como parámetro la función que queremos que sea llamada constantemente siempre que se pueda. En nuestro caso es la función `renderScene`. Añadamos esto dentro del `main()`

```
int main()
{
    //codigo que ya estaba

    glutIdleFunc(renderScene);
    glutMainLoop();
    //resto de código
}
```

Ahora vamos a ver como nuestra tetera gira y nuestra consola se llena de mensajes, y el numero de cuadros pintados aumenta. Pero si te fijas bien, la animacion va un tanto a saltos. Dependiendo del hardware que tengas lo notarás más o menos. Por que? Pues porque estamos realizando cambios en el mismo cuadro que pintamos en pantalla.

## El doble buffer

---

OpenGL dispone de un sistema de dos o más buffers o sectores de memoria donde se va poniendo la información procesada lista para pintarse. En el main, ahora mismo, hemos inicializado nuestro programa usando esto

```
glutInitDisplayMode(GLUT_DEPTH | GLUT_SINGLE | GLUT_RGBA);
```

El `GLUT_SINGLE` significa que usamos solo un buffer, por tanto subiremos la informacion y pintaremos con un solo buffer. Lo tenemos que cambiar por `GLUT_DOUBLE`, para decirle a openGL que a partir de ahora pintaremos en el buffer oculto (back buffer) mientras pintamos el buffer visible (front buffer).

Pero esto, por si solo, no sirve. Necesitamos indicarle a OpenGL que intercambie los buffers una vez tengamos la escena pintada. Para ello, al final de la función `renderScene`, substituiremos `glFlush` por `glutSwappBuffers()`.

**`glutSwappBuffers()` realiza implícitamente un `glFlush` por lo que no es necesario conservarlo**

El programa quedaria definitivamente asi:

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

#include <iostream>
using namespace std;

void cambiarTamanyo(int w, int h) {

    //evitar la division por cero
    if(h == 0)
        h = 1;

    float ratio = 1.0* w / h;

    //seteamos la matriz de proyeccion
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Ponemos el viewport ocupando todo la superficie de la
    ventana
    glViewport(0, 0, w, h);

    // Ponemos la opcion de perspectiva. No usamos la opcion de
    vista ingenieril u ortografica.
    gluPerspective(45, ratio, 1, 1000);
    glMatrixMode(GL_MODELVIEW);

}

void renderScene(void) {
    //borramos el buffer del color
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, -1.0, 0.0f, 1.0f, 0.0f);
    static float xRot = 0.0f;

    //rotamos la tetera un grado por frame
    xRot += 1.0f;
    glRotatef(xRot, 0.0f, 1.0f, 0.0f);
    //pintamos la tetera
    glutWireTeapot(1.0);

    //le decimos a opengl para que pinte todo lo que le hemos
```

```

enviado e intercambie
    glutSwapBuffers();

    static int countFrames = 0;
    cout << "Pintando el cuadro " << countFrames++ << endl;

}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("Tutorial 3D. Animacion");
    glutDisplayFunc(renderScene);

    glutReshapeFunc(cambiarTamanyo);
    glutIdleFunc(renderScene);

    glutMainLoop();
    return EXIT_SUCCESS;
}

```

Para finalizar, decir que la animación, tal como la hemos planteado aquí, no es correcta. Dependiendo de si el hardware grafico es más o menos potente, se pintarán más o menos cuadros y eso hará que la tetera gire a diferentes velocidades dependiendo de la maquina donde lo ejecutemos. Se ha optado por esta formula por simplicidad, pero avisamos que no seria correcto.

## Ejercicios

---

según el código, podrias hacer que la tetera gire más rápida?

Puedes construir un sistema que diga cuanto se tarda en dibujar cada cuadro? Puedes hacer, a partir del sistema anterior, crear un sistema que diga cuantos cuadros por segundo estamos pintando?

## Enlaces

---

Página oficial del compilador GNU:

<http://gcc.gnu.org/>

Página oficial de la IDE Code::blocks:

<http://www.codeblocks.org/>

Página oficial del grupo Khronos (especificación de OpenGL):

<http://www.khronos.org/>

Página oficial de OpenGL:

<http://www.opengl.org/>

Página oficial del grupo de aprendizaje gAp:

<http://www.aprendeprogramacion.net/>