

# **Trabajo Final de IMD**

**Alumno: José Luís Cortés**

**Grado de Ingeniería informática**

**Integración Medios Digitales**

**Trabajo propuesto:  
Revisar y ampliar la “Pecera IMD”**

**Valencia, 10 de abril de 2016.**

## Índice:

	Pág.
1. Introducción.....	5
1.1. Plataforma de desarrollo.....	5
1.2. Compilación y sus problemas.....	6
1.3. Análisis del código y soluciones.....	7
1.4. Funciones a completar o eliminar.....	8
1.4.1. Funciones de imágenes y texturas.....	8
1.4.2. Funciones de teclado y cursor.....	9
1.4.3. Función de ratón.....	11
1.4.4. Función OpenCV y archivos de cabecera.....	12
2. Estructura de datos de elementos visuales.....	12
2.1. Estructura principal.....	12
2.2. Estructura secundaria.....	13
3. Función de animación y repintado.....	14
3.1. Repintado de la pantalla.....	14
3.1.1. Función pintaFoto.....	14
3.1.2. Vector foto y carga de datos.....	15
3.1.3. Fondo de la pecera.....	16
3.1.4. Pintado de objetos y rango coordenadas.....	18
3.1.5. Pinta la comida de peces.....	18
3.1.6. Intercambio de búferes.....	19
3.2. Animación de la pantalla.....	19
3.2.1. Contadores y gestión de tiempo.....	20
3.2.2. Movimiento de la sirena.....	21
3.2.3. Movimiento de los peces.....	23
3.2.4. Código completo de la función.....	25
4. Peces con texturas, nadando en profundidad ( $z \neq 0$ ).....	32
5. Añadir elementos dentro de la pecera.....	33
5.1. Añadir objetos en 3D.....	33
5.2. Añadir peces con texturas.....	33
5.2.1. Captura, pecera con peces.....	34
5.3. Añadir sirena, dibujo vectorial.....	35
5.3.1. Captura, pecera con sirena.....	35
5.4. Añadir adornos al fondo.....	36
5.4.1. Captura, pecera con adornos.....	37
6. Mover burbujas y comida.....	37
6.1. Mover burbujas.....	37
6.2. Mover comida de peces.....	38
7. Incorporar interacción con usuario.....	41
7.1. Detección de cara.....	41
7.1.1. Trabajo previo.....	41
7.1.2. Ejemplo del detector de caras.....	41
7.1.3. Función “webCamOn”.....	42
7.1.4. Función “webCamOff”.....	42
7.1.5. Función “webCamDetCara”.....	43
7.1.6. Captura de la WebCam.....	43
7.2. Un sonido fuerte asusta los peces.....	44

8.	Ampliaciones .....	44
9.	Conclusiones y trabajos futuros.....	45
9.1.	¿Cuánto de los objetivos se ha alcanzado? .....	45
9.2.	¿Qué se podría hacer a partir de aquí?.....	45
10.	Bibliografía y referencias. ....	46

## Ilustraciones:

	Pág.
Ilustración 1-Pecera librerías y versiones .....	5
Ilustración 2-Primera ejecución .....	7
Ilustración 3-Peces originales .....	33
Ilustración 4-Peces añadidos .....	34
Ilustración 5-Pecera, con peces .....	34
Ilustración 6-La sirena .....	35
Ilustración 7-Pecera, con sirena .....	35
Ilustración 8-Adornos originales.....	36
Ilustración 9-Adornos castillos .....	36
Ilustración 10-Adornos del fondo .....	36
Ilustración 11-Pecera, con adornos .....	37
Ilustración 12-Pecera, con burbujas .....	38
Ilustración 13-Pecera, con peces comiendo .....	40
Ilustración 14-Archivos borrados.....	41
Ilustración 15-WebCam de pecera .....	43

### **\* Descripción.**

Esta actividad consiste en revisar el proyecto de la pecera, realizado por otros compañeros el año pasado.

Quitando o completando la funcionalidad parcialmente implementada. Incorporar nueva funcionalidad, con peces de distintos tipos y colores. Dotar de movimiento en profundidad a los peces. Añadir interacción del usuario con los peces.

### **\* Objetivos.**

Se indican los objetivos principales que se pretende conseguir con la realización de esta actividad.

- Análisis del código anterior.
- Crear una estructura de datos para los "peces" y elementos visuales.
- Tener peces con texturas (fotos) nadando en 2D ( $z=0$ ), libre de errores.
- Añadir elementos a la pecera. (adornos, peces distintos, objetos 3D)
- Dotar de movimiento en profundidad a los peces ( $z \neq 0$ ).
- Incorporar la interacción con el usuario.

# 1. Introducción.

En este apartado vamos a tratar unos aspectos iniciales como el entorno de trabajo en el que se realizara la actividad, los diversos incidentes y sus respectivas soluciones durante el análisis y compilación del código original de la versión anterior de la pecera.

## 1.1. Plataforma de desarrollo.

Para la realización de esta actividad se van a usar dos plataformas diferentes:

Una maquina virtual instalada en un PC de escritorio, con un sistema Ubuntu v14.04. Esta será la maquina principal de desarrollo y pruebas, pero no dispone de cámara o micrófono.

Un portátil con un sistema Ubuntu v14.04 instalado en una partición física, esta maquina se usara para pruebas finales ya que cuenta con cámara y micrófono incorporados.

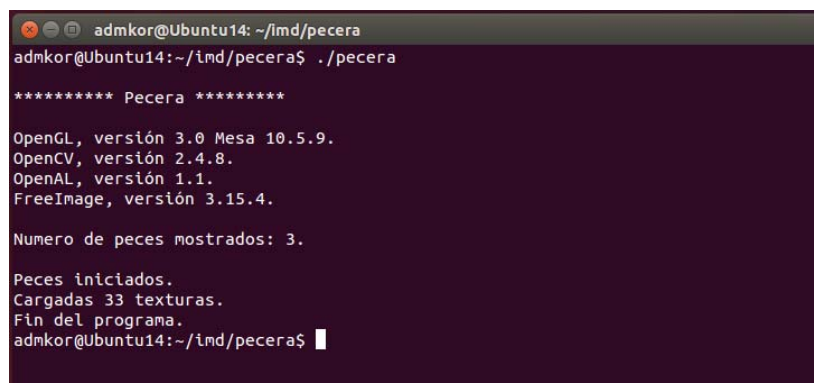
Ambas maquinas presentan la misma versión de ubuntu y librerías para facilitar al máximo la portabilidad del código de una a otra.

Estas librerías se han instalado de la siguiente forma:

gcc y g++:	sudo apt-get install build-essential
OpenGL:	sudo apt-get install mesa-common-dev freeglut3-dev
OpenCV:	sudo apt-get install libopencv-dev python-opencv opencv-doc
OpenAL:	sudo apt-get install libopenal-dev libalut-dev
FreeImage:	sudo apt-get install libfreeimage-dev

Versiones de las librerías usadas pueden verse al inicio del programa, como se muestra en la ilustración “Ilustración 1-Pecera librerías y versiones”. O por línea de comandos con las instrucciones:

Gcc y g++:	gcc -v o g++ -v
OpenGL:	pkg-config --modversion gl
OpenGL utilidades:	pkg-config --modversion glu
OpenCV:	pkg-config --modversion opencv
OpenAL:	pkg-config --modversion openal



```
admkor@Ubuntu14: ~/imd/pecera
admkor@Ubuntu14:~/imd/pecera$ ./pecera

***** Pecera *****

OpenGL, versión 3.0 Mesa 10.5.9.
OpenCV, versión 2.4.8.
OpenAL, versión 1.1.
FreeImage, versión 3.15.4.

Numero de peces mostrados: 3.

Peces iniciados.
Cargadas 33 texturas.
Fin del programa.
admkor@Ubuntu14:~/imd/pecera$
```

Ilustración 1-Pecera librerías y versiones

## 1.2. Compilación y sus problemas.

El primer problema se presento ya que la pecera disponía de dos proyectos “pecera+v2.zip” y “pecera\_lab.zip”. Los dos parecían proceder de la misma versión del año pasado realizado en lenguaje c y no en java como estaba la versión anterior a al v3 de la pecera.

**El primero “pecera+v2.zip”** tenia problemas con las librerías y había que remplazar sus cabeceras, además la función “void InitAlut()” no estaba implementada.

```
##include <GLUT/glut.h>
##include <OpenGL/gl.h>
#include "GL/freeglut.h"
#include "GL/gl.h"
```

**El segundo “pecera\_lab.zip”** si que parece ser la versión final de la pecera v3, la función “void InitAlut()” esta implementada y además hace uso de las librerías de OpenAL para dotar a la pecera de sonido.

El problema es que no compilaba con las instrucciones facilitadas en la memoria de la pecera v3. Por lo que partiendo del trabajo 3 de IMD monte la instrucción de compilación incorporando en ella la librería de OpenAL, por consiguiente esta orden permite usar las tres librerías simultáneamente, OpenGL (Gráficos), OpenCV (Interacción con usuario), OpenAL (sonido).

```
g++ pecera.c -o pecera -lalut -lopenal -lfreeimage -g `pkg-config --cflags
openal opencv glu gl` -lglut `pkg-config --libs openal opencv glu gl`
```

Y para facilitar su ejecución se incorporo el archivo “Makefile” con las modificaciones oportunas.

```
# Simple Makefile
CXX = g++
#CXX = gcc
PROJ = pecera
#EDITOR = gvim
EDITOR = emacs

all: $(PROJ)
$(PROJ): $(PROJ).c
    $(CXX) $(PROJ).c -o $(PROJ) -lalut -lopenal -lfreeimage -g `pkg-config --cflags openal opencv
glu gl` -lglut `pkg-config --libs openal opencv glu gl`
clean:
    -rm -f *.o core $(PROJ)
edit:
    -$(EDITOR) $(PROJ).cpp &
...
```

De esta forma para compilar el proyecto basta con hacer un:

**make**

Y su ejecución con:

**./pecera**

Por ultimo cabe destacar que “**pkg-config --libs openal opencv glu gl**” y “**pkg-config --libs openal opencv glu gl**” incorporaran las rutas de las librerías necesarias independientemente de la versión de estas y lo hará en tiempo de ejecución. Lo que permite ser usado en distintos sistemas.

### 1.3. Análisis del código y soluciones.

Una vez en ejecución, mi primera impresión fue que había algún problema con alguna librería ya que no mostraba peces sino cuadrados que se desplazaban por la pantalla. Puede observarse este hecho en la ilustración “Primera ejecución”.

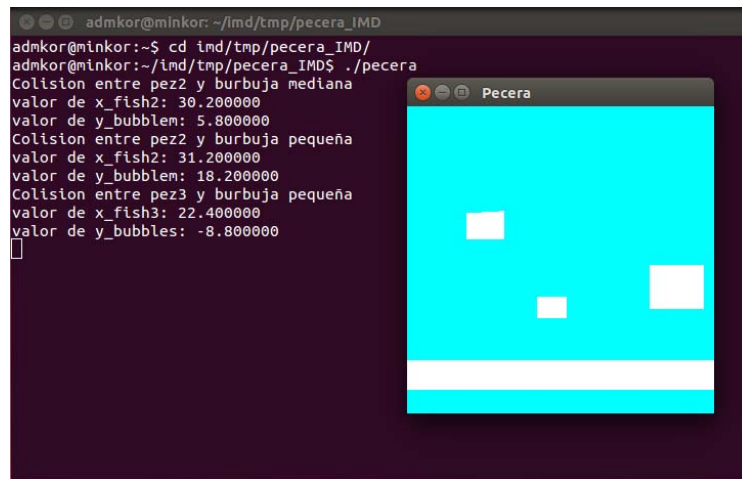


Ilustración 2-Primera ejecución

Al observar el código me fije que las instrucciones de carga de textura estaban comentadas y al descomentarse se producían errores.

```
void cargarTexturas(){  
    // Texturas pez tipo 1  
    glGenTextures(1, &fish_1);  
    glBindTexture(GL_TEXTURE_2D, fish_1);  
    //loadImageFile("imagenes/fish1_black.png");  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);  
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);  
    ...  
}
```

Una vez estudiado el problema con detenimiento observe que la raíz del problema estaba en la falta de las librerías de “FreeImage”. Una vez instaladas y descomentadas las líneas de carga de texturas la pecera apareció correctamente.

## 1.4. Funciones a completar o eliminar.

Todas las funciones relacionadas con OpenGL se han reutilizado o adaptado. Como la función “loadImageFile” o “Init()”.

### 1.4.1. Funciones de imágenes y texturas

Esta función se encarga de leer los archivos de fotos o imágenes y generar la textura para OpenGL.

```
// *** Carga fotos ***
void loadImageFile(char* nombre){
    // Deteccion del formato, lectura y conversion a BGRA
    FREE_IMAGE_FORMAT formato = FreeImage_GetFileType(nombre,0);
    FIBITMAP* imagen = FreeImage_Load(formato, nombre,0);
    if(imagen==NULL) printf("Error al cargar la imagen\n");
    FIBITMAP* imagen32b = FreeImage_ConvertTo32Bits(imagen);

    // Lectura de dimensiones y colores
    int w = FreeImage_GetWidth(imagen32b);
    int h = FreeImage_GetHeight(imagen32b);
    GLubyte* texeles = FreeImage_GetBits(imagen32b);

    // Carga como textura actual
    glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_BGRA_EXT,
GL_UNSIGNED_BYTE, texeles);

    // Liberar recursos
    FreeImage_Unload(imagen);
    FreeImage_Unload(imagen32b);
}
```

Esta otra función inicia el pintado de la pantalla y carga todas las texturas necesarias para pintarlas mas tarde.

```
// *** Inicia el pintado de pantalla ***
void Init(){

    // Color de borrado
    glClearColor(0.0, 4.0, 7.0, 1.0);
    // Habilitar la visibilidad (z-buffer)
    glEnable(GL_DEPTH_TEST);

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    // *** Inicia las fotos (pone coordenadas, velocidad, ...) ***
    IniciaFotos();

    // *** Carga todas las texturas y fotos ***
    for (int i=0; i < FotosMax; i++)
    {
        glGenTextures(1, &foto[i].textura);
        glBindTexture(GL_TEXTURE_2D, foto[i].textura);
        loadImageFile( foto[i].archivo );
    }
}
```



```

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        if (i != 0) glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    }
    printf("Cargadas %i texturas.\n", FotosMax);

    // *** Activa las texturas ***
    glEnable(GL_TEXTURE_2D);
}

```

### 1.4.2. Funciones de teclado y cursor

Otras funciones como las del control de teclado, cursor, ratón y menús se han rehecho completamente, ya que la nueva funcionalidad de la pecera así lo requería.

Las funciones del teclado y menús son muy parecidas, gestionan el control de las funciones de la pecera.

A continuación se presenta la del teclado y en ella se puede observar la funcionalidad disponible en la pecera, son una simple llamada a función y puede que con algo mas de código.

```

// *** Gestión de teclas pulsadas ***
void onKey(unsigned char key, int x, int y){
    switch (key) {

        // *** Salir ***
        case 27:          /* Esc, q, Q will quit */
        case 'q':
        case 'Q':
            exit(0);
            break;

        // *** Reinicia pecera ***
        case 'R':
        case 'r':
            IniciaFotos();
            break;

        // *** Pez asustado ***
        case 'A':
        case 'a':
            pezAsustado = True;
            printf("Peces asustados.\n");
            break;

        // *** Pez normal ***
        case 'S':
        case 's':
            pezAsustado = False;
            printf("Fin del susto, manual.\n");
            break;

        // *** Activa Webcam (y seguimiento de sirena) ***
        case 'V':

```

```

case 'v':
    if (webCamActiva == False)
        webCamActiva = webCamOn();
    else
        printf("La WebCam ya esta activa.\n");

    break;

// *** Desactiva WebCam ***
case 'B':
case 'b':
    if (webCamActiva == True)
    {
        webCamOff();
        webCamActiva = False;
        foto[2].rotax = -1; // esconde sirena
        foto[2].rotay = -1;
    } else
        printf("La WebCam ya esta cerrada.\n");
    break;

// *** Damos de comer a los peces ***
case 'C':
case 'c':
    if (estanComiendo == False)
        IniComida();
    else
        printf("Aun están comiendo.\n");
    break;

// *** Muestra la ayuda ***
case 'H':
case 'h':
    ayuda();
    break;
}
}

```

Con las teclas del cursor podemos aumentar o disminuir el número de peces mostrados en la pecera. También controlamos el volumen del sonido de fondo. Todo esto controlado por la siguiente función.

```

// *** Teclas del cursor ***
void onSpecialUp(int tecla, int x, int y){

    switch (tecla)
    {
        // *** Cursor arriba - Aumenta el nº de peces ***
        case GLUT_KEY_UP:
            if (numPeces < FotosComida - FotosPez)
            {
                numPeces++;
                printf("Numero de peces mostrados: %i \n", numPeces);
                glutPostRedisplay();
            }
            break;

        // *** Cursor abajo - Disminuye el nº de peces ***
        case GLUT_KEY_DOWN:

```

```

    if (numPeces > 0)
    {
        numPeces--;
        printf("Numero de peces mostrados: %i \n", numPeces);
        glutPostRedisplay();
    }
    break;

    // *** Cursor izq - Sube el volumen ***
    case GLUT_KEY_RIGHT:
    if (volumen < 1.0)
    {
        volumen += 0.05;
        alSourcef(fuente,AL_GAIN,volumen);    // pone volumen(ganancia)
        printf("Sube el volumen a %f.\n", volumen);
    }
    break;

    // *** Cursor izq - Disminuye el volumen ***
    case GLUT_KEY_LEFT:
    if (volumen > 0.0)
    {
        volumen -= 0.05;
        alSourcef(fuente,AL_GAIN,volumen);    // pone volumen(ganancia)
        printf("Baja el volumen a %f.\n", volumen);
    }
    break;
}
}
}

```

### 1.4.3. Función de ratón

Con el ratón se puede dar de comer a los peces, clicando en la parte superior de la pantalla. O llamar a la sirena clicando en la parte inferior. Muestro la función con controla el ratón.

```

// *** Control del ratón ***
void mouse(int button, int state, int x, int y){

    // *** Tamaño ventana ***
    double ventAncho = glutGet(GLUT_WINDOW_WIDTH);
    double ventAlto = glutGet(GLUT_WINDOW_HEIGHT);

    // *** Tantos por uno de x, y ***
    double porX = x / ventAncho;
    double porY = y / ventAlto;

    // *** La sirena se va ***
    // se generan dos eventos uno al pulsar el boton (state == GLUT_DOWN)
    // y otro al soltar lo (state == GLUT_UP)
    if(button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
    {
        foto[2].rotax = -1;
        foto[2].rotay = -1;
        foto[2].rotaz = 0;
        foto[2].invDir = True;
        printf("La Sirena se esconde.\n");
    }
}

```

```

// *** La sirena te busca ***
//GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON
if(button == GLUT_LEFT_BUTTON && porY > 0.30 && state == GLUT_DOWN)
{
    foto[2].rotax = porX * 80;
    foto[2].rotay = 40 - (porY * 40);
    foto[2].rotaz = 0;
    foto[2].invDir = True;
    printf("La Sirena te busca.\n");
}

// *** Damos de comer a los peces ***
if (button == GLUT_LEFT_BUTTON && porY < 0.30 && state == GLUT_DOWN &&
estanComiendo == False)
    IniComida();
}

```

#### 1.4.4. Función OpenCV y archivos de cabecera

Las funciones relacionadas con la detección de caras y OpenCV, han sido desechadas totalmente. Generaban muchos errores y eran demasiado complejas.

Esto junto con la eliminación de los archivos de cabecera, innecesarios en Ubuntu. Ha permitido la eliminación de un gran número de archivos y la consecuente simplificación de la pecera.

Más adelante se hablara de la detección de caras.

## 2. Estructura de datos de elementos visuales

Usamos dos tipos de estructura de datos para simplificar la gestión y mantener unidas todas las características de cada foto.

### 2.1. Estructura principal

La primera y principal estructura de toda la pecera el tipoFoto, esta almacena las texturas de todos los elementos a pintar en la pecera, junto con su posición y otras características que se mencionan a continuación:

- **Archivo:** Nombre y ruta del archivo a cargar.
- **Textura:** Aquí se carga la textura que se usa para pintar en la pecera.
- **x, y, z:** Son las coordenadas para localizar el objeto en pantalla.
- **velox, veloy, veloz:** Es la velocidad a la que se desplaza el objeto en cada una de sus componentes x, y, z.

- **veloActx, veloActy, veloActz:** Cuando un pez alcanza un borde de la pecera su velocidad invierte su signo, causando un cambio brusco en su dirección. Para evitar este y dotar a los peces de un movimiento más real, se usan estas variables, que son las que realmente marcan la velocidad del pez. El objetivo es que cuando un pez llega al borde desacelere lentamente, gire y acelere hasta alcanzar su velocidad normal. En definitiva cuando *velox* cambie su signo, *veloActx* se decrementa o incrementa rápidamente, para dotar a los peces de movimiento realista.
- **zoomx, zoomy:** Permite variar el tamaño de la textura y permitir hacer todos los peces de igual tamaño o elegir algunos para que sean mas grandes que otros. También permite tener burbujas de distintos tamaños con la misma textura.
- **rotax, rotay, rotaz:** Indica el ángulo de rotación que se aplica a las texturas y así dar la sensación de que el pez da la vuelta cuando invierte su dirección.
- **rotaActx, rotaActy, rotaActz:** Para evitar que el pez gire de un salto, usamos el mismo principio que con *veloActx*. El objetivo que el pez gire lentamente mientras reduce y acelera su velocidad y así conseguir un movimiento realista. Durante la rotación en pantalla se observa en efecto algo peculiar, como si el pez tomara impulso al girar, no me ha desagradado y lo he dejado.
- **invDir:** Permite que el pez pueda invertir su dirección (invierte el signo de su velocidad) en cualquier momento sin tener que llegar al borde de la pecera. Esto da cierta personalidad a los peces ya que permite que sus movimientos sean más caóticos.

Esta estructura se usa para crear la variable “foto”, como se ve en este código:

```
struct tipoFoto
{
    char    archivo[1024];           // nombre de la foto
    GLuint  textura;                // textura de la foto
    float   x, y, z;                // posición de la foto
    float   velox, veloy, veloz;     // velocidad
    float   veloActx, veloActy, veloActz; // velocidad actual
    float   zoomx, zoomy;           // zoom
    float   rotax, rotay, rotaz;     // rotación
    float   rotaActx, rotaActy, rotaActz; // rotación actual
    int     invDir;                 // invierte su dirección al azar
} foto[FotosMax];
```

## 2.2. Estructura secundaria

La segunda estructura de datos que se usa es para crear triángulos, la comida que se usa para dar de comer a los peces. Se usa esta segunda estructura y no la primera por el alto numero de triángulos necesarios, minimizando el uso de recursos.

En esta estructura no están las texturas ya que se usan las que ya están cargadas en la estructura principal y se reutilizaran las veces que sean necesarias. Además algunos datos como la “rotación” o “velocidad actual” no son necesarios, ya que el movimiento es muy lento y carece de cambio de sentido.

Si requiere un parámetro booleano “sePinta” que indica si se tiene que dibujar o no ese triangulo, y así poder dar la sensación de que los peces se van comiendo la comida antes de que llegué al suelo. Este booleano se simula con el uso de dos constantes “True” y “False” definidas al inicio del programa.

```
struct tipoComida
{
    float  x, y, z;                // posición de la foto
    float  velox, veloy, veloz;    // velocidad
    float  zoomx, zoomy;          // zoom
    int     sePinta;              // se pinta
} comida[PapelesMax];
```

Todos los elementos grafico en pantalla estarán asociados con una de estas estructuras de datos y será el motor grafico es encargado de pintar.

### 3. Función de animación y repintado

Las funciones de animación o movimiento y pintado de pantalla son el núcleo de toda la pecera, y las que hacen uso intensivo de las estructuras principales, ya comentadas en el punto “[Estructura de datos de elementos visuales](#)”.

#### 3.1. Repintado de la pantalla

La función de repintado (display) es la que se encarga de pintar todos los elementos de la pecera, usando para ello la estructura principal o variable foto, se trata de un vector de tipo “tipoFoto” que corresponde con la estructura principal.

##### 3.1.1. Función pintaFoto

Llamando a la función “pintaFoto(i)”, conseguimos pintar la textura i del vector “foto”, con las características que se indiquen para esa textura. También hay que destacar que se disponen de unas contantes que permiten delimitar en que posiciones del vector están los diferentes tipos de texturas, como se puede observar en el siguiente trozo de código.

```
#define FotosAdor 3    // Donde empiezan los adornos.
#define FotosBru  10   // Donde empiezan las burbujas.
#define FotosPez  13   // Donde empiezan los peces.
#define FotosComida 29 // Donde empiezan las comidas.
#define FotosMax  33   // Numero de foto y texturas
```

### 3.1.2. Vector foto y carga de datos

Cabe destacar que antes de poder usar la estructura principal de datos, con el vector foto. Hay que cargar los datos de cada elemento, sus coordenadas, zoom, velocidad, etc. Además de las texturas. Operación que se realiza conjuntamente por las funciones “Init” e “iniciarFotos”.

Muestro el código parcial de esta función:

```
/ *** Inicia el pintado de pantalla ***
void Init(){

    // Color de borrado
    ...

    // *** Inicia las fotos (pone coordenadas, velocidad, ...) ***
    IniciaFotos();

    // *** Carga todas las texturas y fotos ***
    for (int i=0; i < FotosMax; i++)
    {
        glGenTextures(1, &foto[i].textura);
        glBindTexture(GL_TEXTURE_2D, foto[i].textura);
        loadImageFile( foto[i].archivo );
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        if (i != 0) glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    }
    printf("Cargadas %i texturas.\n", FotosMax);

    ...

}
```

La función “iniciarFotos” además de cargar los datos en el vector foto, se usa para reiniciar la pecera. Por eso evita que los peces estén asustados o comiendo.

Muestro el código parcial de esta función:

```
// *** Inicia las fotos y texturas ***
void IniciaFotos(){
    int x;

    // *** Inicia los peces en estado normal ***
    pezAsustado = False;    // peces no asustados.
    estanComiendo = False; // peces no comen.

    // *** Inicia los datos ***
    for (int i=0; i < FotosMax; i++)
    {
        foto[i].x=0; foto[i].y=0; foto[i].z=0;    // posición de la foto
        foto[i].velox=0; foto[i].veloy=0; foto[i].veloz=0;    // velocidad
    }
    ...
}
```

```

// *** Nombre de las fotos ***
// ref: http://www.tutorialspoint.com/cplusplus/cpp\_data\_structures.htm
strcpy(foto[0].archivo, "foto/fondo.jpeg");
strcpy(foto[1].archivo, "foto/coral.png");
strcpy(foto[2].archivo, "foto/Sirena.png");

...

strcpy(foto[31].archivo, "foto/Comida-Roja.png");
strcpy(foto[32].archivo, "foto/Comida-Verde.png");

// *** Pone coordenadas del fondo ***
x=0;
foto[x].x = -1; foto[x].y = -1; foto[x].z = 0;
foto[x].zoomx = 1; foto[x].zoomy = 1;

// *** Pone coordenadas del coral ***
x=1;
foto[x].x = -20; foto[x].y = -20; foto[x].z = -0.5;
foto[x].zoomx = 0.4; foto[x].zoomy = 0.4;

// *** Pone coordenadas del sirena ***
x=2;
foto[x].x = 120; foto[x].y = 10; foto[x].z = 0;
foto[x].veloActx=foto[x].x; foto[x].veloActy=foto[x].y; foto[x].veloActz=foto[x].z;
foto[x].zoomx = 1.8; foto[x].zoomy = 0.8;
foto[x].velox= -0.8; foto[x].veloy= -0.4; foto[x].veloz= -0.0;
foto[x].rotay = 180; foto[x].invDir = False;

...

// *** Pone coordenadas de peces payaso 2 ***
x=27;
foto[x].x = 10; foto[x].y = 18; foto[x].z = -6;
foto[x].zoomx = 0.6; foto[x].zoomy = 0.4;
foto[x].velox= -0.6; foto[x].veloy= 0.4; foto[x].veloz= -0.0;
foto[x].rotay = 180; foto[x].invDir = False;

// *** Pone coordenadas de peces payaso 3 ***
x=28;
foto[x].x = 20; foto[x].y = 14; foto[x].z = -6;
foto[x].zoomx = 0.6; foto[x].zoomy = 0.4;
foto[x].velox= -0.6; foto[x].veloy= 0.4; foto[x].veloz= -0.0;
foto[x].rotay = 180; foto[x].invDir = False;

printf("Peces iniciados.\n");
}

```

### 3.1.3. Fondo de la pecera

Empezamos por el borrado de la pantalla y pintado del fondo de la pecera, código íntegramente reutilizado de mis compañeros del año pasado que hicieron un excelente trabajo con OpenGL.

```

// *** Pinta la pantalla ***
void display(){

```



```

// *** Borra pantalla ****
glClearColor(0.5,0.5,0.5,0.5); // Color del borrado.
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// Seleccionar la matrix del modelo
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// Sitúo y oriento la cámara
gluLookAt(    0, 0, 20 /* ojo */,
             0.0, 0.0, 0.0 /* punto al que miro */,
             0.0, 1.0, 0.0 /* vertical subjetiva */);

// *** Pinta fondo ***
glBindTexture(GL_TEXTURE_2D, foto[0].textura);
glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_TEXTURE_BIT);
glDisable(GL_DEPTH_TEST);
glDisable(GL_LIGHTING);
glEnable(GL_TEXTURE_2D);
glDisable(GL_TEXTURE_GEN_S);
glDisable(GL_TEXTURE_GEN_T);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//Texel menor que pixel
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
//Texel mayor que pixel
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
//La textura se repite en abcisas
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
//La textura se repite en ordenadas
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
//Asigna solo el color de la textura al fragmento

// --- Cargar el fondo con la textura corriente ---
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
glOrtho(-1,1,-1,1,-10,10);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();

glBegin(GL_POLYGON);
glTexCoord2f(0,0); glVertex3f(-1,-1,0);
glTexCoord2f(1,0); glVertex3f(1,-1,0);
glTexCoord2f(1,1); glVertex3f(1,1,0);
glTexCoord2f(0,1); glVertex3f(-1,1,0);
glEnd();

glPopMatrix();
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);

glPopAttrib();

```

### 3.1.4. Pintado de objetos y rango coordenadas

Continuando con el pintado del coral y resto de objetos.

Cabe destacar que para pintar el coral he tenido que establecer los límites de la pantalla y lo he hecho de forma experimental, estableciendo para la **coordenada x un rango de -40 a 40**, para la **coordenada y un rango de -20 a 20** y para la **coordenada z un rango de 0 a -20**.

Este último rango debe ser negativo para evitar que los peces atravesasen el cristal frontal de la pecera, situándose por detrás del punto de visión.

```
// *** Pinta coral ***
for(foto[1].x = -40; foto[1].x < 40; foto[1].x = foto[1].x + 5)
    pintaFoto(1);

// *** Pinta adornos ***
for (int i = FotosAdor; i < FotosBru; i++)
    pintaFoto(i);

// *** Pinta burbujas ***
for (int i = FotosBru; i < FotosPez; i++)
    pintaFoto(i);

// *** Pinta los peces ***
for (int i = FotosPez; i < FotosComida; i++)
    if ( i < FotosPez + numPeces ) pintaFoto (i);

// *** Pinta la sirena ***
if (foto[2].invDir == True) pintaFoto (2);
```

### 3.1.5. Pinta la comida de peces

Finalmente llegamos al pintado de la comida, pero antes de llamar a la función “pintaFoto(i)” hay que trasladar los datos desde la estructura secundaria o vector “comida[i]” a una de las 4 texturas principales disponible para la comida vector “foto[a]”.

```
// *** Pinta la comida ***
if (estanComiendo == True)
{
    int contaComida = 0;
    int a;

    for (int i=0; i < PapelesMax; i++)
    {
        if ( comida[i].sePinta == True)
        {
            a = FotosComida + contaComida;

            foto[a].x = comida[i].x;
            foto[a].y = comida[i].y;
            foto[a].z = comida[i].z;

            foto[a].zoomx = comida[i].zoomx;
```

```

        foto[a].zoomy = comida[i].zoomy;

        foto[a].rotaActx = 0;
        foto[a].rotaActy = 0;
        foto[a].rotaActz = 0;

        pintaFoto (a);
    }
    contaComida = (contaComida + 1) % 4;
}
}

```

### 3.1.6. Intercambio de búferes

Completamos la operación intercambiando los búferes, el recién pintado a pantalla y el de pantalla para pintar.

```

// *** Intercambia los búferes ***
glutSwapBuffers();
}

```

## 3.2. Animación de la pantalla

Esta función es la que se encarga de modelar todo el movimiento que se produce en la pecera desde el movimiento realista de los peces nadando, pasando por cuando se asustan o cuando tienen que comer, hasta llegar al movimiento y seguimiento que realiza la sirena de nuestra pecera, con ayuda de nuestro detector de caras.

A sido muy gratificante **ver crecer esta pequeña función**, desde su nacimiento con unas pocas líneas para el movimiento de las coordenadas x, y, de hay su nombre “muevePeces”. Hasta alcanzar su madurez, siendo no solo capaz de modelar el movimiento de todos los objetos, sino controlando el tiempo que están comiendo los peces o cuanto dura el susto a los peces, hasta llegar ha ser la que controla la sobrecarga que el detector de caras produce en el sistema, controlando cada cuantas veces de llama ha la función de detección de caras evitando una ralentización extrema en el funcionamiento de la pecera.

Es una función bastante extensa, pero no he visto ningún motivo para tener que dividirla en varias funciones ya que todas serian llamadas desde el mismo punto y con los mismos argumentos. Por lo que la reutilización de código no se daría ya que para ello las funciones tienen que ser llamadas desde varios puntos o con distintos argumentos que les permitan realizar acciones distintas.

La única razón para fragmentar la función seria la portabilidad del código, cosa que queda entredicho ya que la función usa varias variables globales que dificultan su portabilidad. Por lo que **he optado por mantenerlo todo junto en esta función** pero de forma estructurada y documentada, indicando que realiza cada una de sus secciones.

### 3.2.1. Contadores y gestión de tiempo

Para minimizar el uso de variables globales la función usa variables locales estáticas, que permiten recordar su valor entre llamadas a esta función. Estas se usan como simples contadores, para controlar el tiempo que dura la comida de los peces, cuanto dura el susto o cada cuantas veces se llama al detector de caras.

```
// *** Mueve los peces ***
void muevePeces(int valorSinUso){

    static int pezAsustadoContador;
    static int webCamContador;
    static int comerContador;

    int x;

    // *** Inicia el generador de números aleatorios ***
    srand((unsigned int) time(NULL));

    // *****
    // *** Desactiva el susto de peces, después unos seg ***
    // *****
    if (pezAsustado == False)
        pezAsustadoContador = 40;
    else
    {
        pezAsustadoContador--;
        if (pezAsustadoContador <= 0)
        {
            // *** Fin del susto ***
            pezAsustado = False;

            // *** Peces que mueren del susto ***
            for (int i = FotosPez; i < FotosMax; i++)

                if (rand() % 100 < 80) foto[i].rotaActz = 180;

            printf("Fin del susto, automático (algunos peces se han muerto).\n");
        }
    }
}

// *****
// ***** Detecta Caras (con webCam activa) *****
// *****
if ( webCamActiva == True )
{
    webCamContador = (webCamContador + 1) % 4;
    if (webCamContador == 0)
    {
        webCamDetCara();
        foto[2].rotax = 80 - (caraX * 80); // en la webcam x esta invertida
        foto[2].rotay = 40 - (caraY * 40);
        foto[2].invDir = True; // activa el pintado
    }
}

...

```

```

// *****
// ***** Mueve la comida *****
// *****
if (estanComiendo == False)
    comerContador = 300;    // tiempo de espera para comer.
else
    if (comerContador <= 0)
    {
        // *** Fin de la comida ***
        estanComiendo = False;
        printf("Fin de la comida.\n");

    } else {

        // *** Están comiendo ***
        comerContador--;

        for (int i=0; i < PapelesMax; i++)
            if ( comida[i].sePinta == True)
            {
                comida[i].x += comida[i].velox;
                comida[i].y += comida[i].veloy;
                comida[i].z += comida[i].veloz;

                if (comida[i].y < -15 || drand48() < ( (comerContador > 100) ? 0.01
: 0.03 ) )
                    comida[i].sePinta = False;
            }
    }
}

```

### 3.2.2. Movimiento de la sirena

En el movimiento de **la sirena se han reutilizado las variables** que facilita la estructura de datos principal, pero debido a las necesidades especiales del movimiento de la sirena que siguen al detector de caras, las variables **tienen significados distintos a los habituales**.

Se que esto dificulta la legibilidad del código pero evita el uso de nuevas variables, además al inicio del código se documenta la funcionalidad que tienen cada una de estas variables, aunque su nombre no concuerde mucho con su funcionalidad.

Hay que tener en cuenta que **el detector de caras devuelve un tanto por uno**. Es decir da un valor del 0 al 1 que indica en que porcentaje de la pantalla esta. Por ejemplo que la cara se detecta en la mitad derecha de la pantalla a una altura del centro de la pantalla, tendríamos una  $rotax=0.25$  y  $rotay=0.50$  indicando que “x” debe ser el 25% de la anchura de la pantalla e “y” debe ser el 50% del alto de la pantalla.

No es muy preciso pero suficiente para que la sirena te siga correctamente, he ignora los tamaños o resoluciones de pantalla que se usen, ya que hace uso de porcentajes no de pixeles.

A continuación comento el uso de las variables para el movimiento de la sirena.

- **x, y, z:** Son la posición actual de la sirena en pantalla.
- **rotax, rotay, rotaz:** Es la posición a la que tiene que llegar la sirena. Es el tanto por uno del detector de caras por el ancho, alto u hondo de la pantalla, respectivamente. Siempre son valores positivos. Y hacen referencia a al ancho, alto u hondo totales de la pantalla.
- **veloActx, veloActy, veloActz:** Estas variables indican la posición actual de la sirena en función del ancho, alto u hondo totales de la pantalla. Estas variables siempre siguen a “rotax, rotay, rotaz” ya que quieren ser iguales. Estas variables acaban convertidas en “x, y, z” al final del proceso. El motivo por el cual se usan es para simplificar las funciones de búsqueda de la sirena ya que solo tienen que contemplar un solo caso, cuando las coordenadas son positivas.
- **velox, veloy, veloz:** Es la velocidad a la que se desplaza la sirena, esta velocidad se reduce (se divide) conforme se acerca la sirena al punto indicado por el detector de caras. Su signo también depende del lugar donde se encuentre el detector de caras.
- **invDir:** Se usa para ocultar a la sirena cuando se esconde. Básicamente indica si la sirena debe pintarse o no.
- **rotaActx, rotaActy, rotaActz:** Indica la rotación que se da a la sirena, evita que la sirena nade hacia atrás.

Código encargado del movimiento de la sirena:

```
// *****
// ***** Mueve la sirena *****
// *****
x=2;
if ( foto[x].invDir == True)
{
    // Posición real en pantalla:      x, y, z (positivas o negativas);
    // Posición a la que quiere ir:    rotax, rotay, rotaz (positivas siempre);
    // Posición actual:                veloActx, veloActy, veloActz (positivas siempre);
    // Velocidad:                      velox, veloy, veloz;
    // Se pinta:                       invDir;
    // Rotación:                       rotaActx, rotaActy, rotaActz

    double distancia;

    // ##### Control x #####

    // *** La sinera se esconde (1º se va y después se esconde) ***
    if (foto[x].rotax == -1.0 && foto[x].rotay == -1.0)
    {
        foto[x].rotax = 140;
        foto[x].rotay = 10;
        if (foto[x].velox < 0) foto[x].rotax *= -1;
    }
    if (fabs(foto[x].veloActx) > 120 ) foto[x].invDir = False;
```

```

// *** Busca la posición (ajusta la dir de la velo) ***
if (foto[x].veloActx > foto[x].rotax && foto[x].velox > 0)
    foto[x].velox = foto[x].velox * -1;

if (foto[x].veloActx < foto[x].rotax && foto[x].velox < 0)
    foto[x].velox = foto[x].velox * -1;

// *** Controla la rotación según su dirección ***
if (foto[x].velox > 0) { foto[x].rotaActx=0; foto[x].rotaActy=180; }
if (foto[x].velox <= 0) { foto[x].rotaActx=0; foto[x].rotaActy=0; }

// *** Suaviza el avance ***
distancia = (fabs(foto[x].veloActx - foto[x].rotax) / 80) * 100;
if (distancia > 20 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox;
else if (distancia > 10 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox / 1.5;
else if (distancia > 6 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox / 2;
else if (distancia > 4 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox / 4;

// *** Actualiza la posi real **
// x va de -40 a 40 y tiene una anchura de 80.
foto[x].x=foto[x].veloActx - 40;

// ##### Control y #####

// *** Busca la posición (ajusta la dir de la velo) ***
if (foto[x].veloActy > foto[x].rotay && foto[x].veloy > 0)
    foto[x].veloy = foto[x].veloy * -1;

if (foto[x].veloActy < foto[x].rotay && foto[x].veloy < 0)
    foto[x].veloy = foto[x].veloy * -1;

// *** Suaviza el avance ***
distancia = (fabs(foto[x].veloActy - foto[x].rotay) / 40) * 100;
if (distancia > 20 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy;
else if (distancia > 10 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy / 1.5;
else if (distancia > 6 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy / 2;
else if (distancia > 4 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy / 4;

// *** Actualiza la posi real **
// y va de -20 a 20 y tiene una anchura de 40.
foto[x].y=foto[x].veloActy - 20;
}

```

### 3.2.3. Movimiento de los peces

Por ultimo llegamos al movimiento de los peces.

El comportamiento de los tres ejes x, y, z es muy similar.

Inicialmente los peces se movían de izquierda a derecha y al llegar al borde volvían a salir por la derecha.

En la actualidad es mucho mas complejo, para empezar hay un numero variable de peces en la pecera, por lo que los peces que no se pintan no deben moverse para evitar desperdiciar recursos.

```
// *****  
// ***** Mueve los peces *****  
// *****  
for (int i = FotosPez; i < FotosComida; i++)  
{  
    // *** Sale si el pez no se pinta ***  
    if ( i >= FotosPez + numPeces ) break;  
  
    // ##### Control x #####
```

El pez respetara los limites de la pantalla siempre que no este asustado, si esta asustado puede romperlos para salir huyendo.

```
    // *** Llega al limite derecho ***  
    if (foto[i].x > 40 -1 && foto[i].velox > 0 && pezAsustado == False)  
        foto[i].velox = foto[i].velox * -1;  
  
    // *** Llega al limite izquierdo ***  
    if (foto[i].x < -40 && foto[i].velox < 0 && pezAsustado == False)  
        foto[i].velox = foto[i].velox * -1;
```

Hay que hacer una mención especial para el eje y, si los peces están comiendo hay que hacer que todos estén en la mitad superior de la pantalla para que coman y en caso contrario que puedan bajar hasta el fondo.

```
    // ##### Control y #####  
  
    // *** Llega al limite superior ***  
    if (foto[i].y > 15 && foto[i].veloy > 0 && pezAsustado == False)  
        foto[i].veloy = foto[i].veloy * -1;  
  
    // *** Llega al limite inferior ***  
  
    if (foto[i].y < ( estanComiendo == False ) ? -15 : 5) && foto[i].veloy < 0 &&  
    pezAsustado == False)  
        foto[i].veloy = foto[i].veloy * -1;
```

El cambio de dirección al azar tampoco se no permite si el pez esta asustado y se exige que el pez halla alcanzado el 80% de su velocidad normal, esto evita que los peces cambien de dirección varias veces en poco tiempo, provocando un efecto visual muy molesto.

```
    // *** Cambio de sentido al azar (evita cambios continuados) ***  
    if  
    (  
        rand() % 100 < 0.2 && foto[i].invDir == True && pezAsustado == False &&  
        (foto[i].velox > 0 && foto[i].veloActx > foto[i].velox * 0.8 ||  
        foto[i].velox < 0 && foto[i].veloActx < foto[i].velox * 0.8)  
    )
```



```

        foto[i].velox = foto[i].velox * -1;

// *** Cambia la dirección con suavidad ***
// ref: http://www.cplusplus.com/reference/cmath/abs/
if ( foto[i].veloActx > foto[i].velox )
    foto[i].veloActx = foto[i].veloActx - fabs(foto[i].velox/40);

if ( foto[i].veloActx < foto[i].velox )
    foto[i].veloActx = foto[i].veloActx + fabs(foto[i].velox/40);

```

El pez debe rotar para evitar nadar hacia atrás, esta rotación se hace lentamente para evitar que de un salto y visualmente parece que el pez tome impulso para girar.

```

// *** Controla la rotación según su dirección ***
if (foto[i].veloActx > 0) { foto[i].rotax=0; foto[i].rotay=180; }
if (foto[i].veloActx <= 0) { foto[i].rotax=0; foto[i].rotay=0; }

// *** Cambia la rotación con suavidad ***
if ( foto[i].rotaActy > foto[i].rotay )
    foto[i].rotaActy = foto[i].rotaActy - 45;

if ( foto[i].rotaActy < foto[i].rotay )
    foto[i].rotaActy = foto[i].rotaActy + 45;

```

El pez tendrá una velocidad normal, salvo si esta muerto en este caso su velocidad será 4 veces mas lenta y si esta asustado su velocidad será 40 veces mas rápida.

```

// *** Actualiza movimiento de der a izq ***
if (pezAsustado == False )
    if ( foto[i].rotaActz != 180 || fabs(foto[i].x) > 80)
        // *** Pez normal ***
        foto[i].x = foto[i].x + foto[i].veloActx;
    else
        // *** Pez muerto ***
        foto[i].x = foto[i].x + (foto[i].veloActx / 4.0f);
else
    // *** Pez asustado ***
    if ( fabs(foto[i].x) < 100 )
        foto[i].x = foto[i].x + foto[i].veloActx*40;

```

### 3.2.4. Código completo de la función

Mostramos el código integro de la función que se encarga del movimiento de todos los elementos de la pecera.

```

// *** Mueve los peces ***
void muevePeces(int valorSinUso){

    static int pezAsustadoContador;
    static int webCamContador;
    static int comerContador;

    int x;

```

```

// *** Inicia el generador de números aleatorios ***
srand((unsigned int) time(NULL));

// *****
// *** Desactiva el susto de peces, después unos seg ***
// *****
if (pezAsustado == False)
    pezAsustadoContador = 40;
else
{
    pezAsustadoContador--;
    if (pezAsustadoContador <= 0)
    {
        // *** Fin del susto ***
        pezAsustado = False;

        // *** Peces que mueren del susto ***
        for (int i = FotosPez; i < FotosMax; i++)

            if (rand() % 100 < 80) foto[i].rotaActz = 180;

        printf("Fin del susto, automático (algunos peces se han muerto).\n");
    }
}

// *****
// ***** Detecta Caras (con webCam activa) *****
// *****
if (webCamActiva == True)
{
    webCamContador = (webCamContador + 1) % 4;
    if (webCamContador == 0)
    {
        webCamDetCara();
        foto[2].rotax = 80 - (caraX * 80); // en la webcam x esta invertida
        foto[2].rotay = 40 - (caraY * 40);
        foto[2].invDir = True; // activa el pintado
    }
}

// *****
// ***** Mueve las burbujas *****
// *****
for (int i = FotosBru; i < FotosPez; i++)
{
    // *** Llega al limite superior ***
    if (foto[i].y > 20)
        foto[i].y = -20;
    else
        // *** Actualiza movimiento ***
        foto[i].y = foto[i].y + foto[i].veloy;
}

// *****
// ***** Mueve la comida *****
// *****
if (estanComiendo == False)
    comerContador = 300; // tiempo de espera para comer.

```

```

else
    if (comerContador <= 0)
    {
        // *** Fin de la comida ***
        estanComiendo = False;
        printf("Fin de la comida.\n");

    } else {

        // *** Están comiendo ***
        comerContador--;

        for (int i=0; i < PapelesMax; i++)
            if ( comida[i].sePinta == True)
            {
                comida[i].x += comida[i].velox;
                comida[i].y += comida[i].veloy;
                comida[i].z += comida[i].veloz;

                if (comida[i].y < -15 || drand48() < ( (comerContador > 100) ? 0.01
: 0.03 ) )

                    comida[i].sePinta = False;
            }
        }

// *****
// ***** Mueve la sirena *****
// *****
x=2;
if ( foto[x].invDir == True)
{
    // Posición real en pantalla:      x, y, z (positivas o negativas);
    // Posición a la que quiere ir:    rotax, rotay, rotaz (positivas siempre);
    // Posición actual:                veloActx, veloActy, veloActz (positivas siempre);
    // Velocidad:                      velox, veloy, veloz;
    // Se pinta:                       invDir;
    // Rotación:                       rotaActx, rotaActy, rotaActz

    double distancia;

    // ##### Control x #####

    // *** La sinera se esconde (1º se va y después se esconde) ***
    if (foto[x].rotax == -1.0 && foto[x].rotay == -1.0)
    {
        foto[x].rotax = 140;
        foto[x].rotay = 10;
        if (foto[x].velox < 0) foto[x].rotax *= -1;
    }
    if (fabs(foto[x].veloActx) > 120 ) foto[x].invDir = False;

    // *** Busca la posición (ajusta la dir de la velo) ***
    if (foto[x].veloActx > foto[x].rotax && foto[x].velox > 0)
        foto[x].velox = foto[x].velox * -1;

    if (foto[x].veloActx < foto[x].rotax && foto[x].velox < 0)
        foto[x].velox = foto[x].velox * -1;

    // *** Controla la rotación según su dirección ***

```

```

if (foto[x].velox > 0) { foto[x].rotaActx=0; foto[x].rotaActy=180; }
if (foto[x].velox <= 0) { foto[x].rotaActx=0; foto[x].rotaActy=0; }

```

```

// *** Suaviza el avance ***
distancia = (fabs(foto[x].veloActx - foto[x].rotax) / 80) * 100;
if (distancia > 20 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox;
else if (distancia > 10 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox / 1.5;
else if (distancia > 6 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox / 2;
else if (distancia > 4 )
    foto[x].veloActx = foto[x].veloActx + foto[x].velox / 4;

```

```

// *** Actualiza la posi real **
// x va de -40 a 40 y tiene una anchura de 80.
foto[x].x=foto[x].veloActx - 40;

```

```

// ##### Control y #####

```

```

// *** Busca la posición (ajusta la dir de la velo) ***
if (foto[x].veloActy > foto[x].rotay && foto[x].veloy > 0)
    foto[x].veloy = foto[x].veloy * -1;

```

```

if (foto[x].veloActy < foto[x].rotay && foto[x].veloy < 0)
    foto[x].veloy = foto[x].veloy * -1;

```

```

// *** Suaviza el avance ***
distancia = (fabs(foto[x].veloActy - foto[x].rotay) / 40) * 100;
if (distancia > 20 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy;
else if (distancia > 10 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy / 1.5;
else if (distancia > 6 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy / 2;
else if (distancia > 4 )
    foto[x].veloActy = foto[x].veloActy + foto[x].veloy / 4;

```

```

// *** Actualiza la posi real **
// y va de -20 a 20 y tiene una anchura de 40.
foto[x].y=foto[x].veloActy - 20;

```

```

}

```

```

// *****
// ***** Mueve los peces *****
// *****

```

```

for (int i = FotosPez; i < FotosComida; i++)
{

```

```

    // *** Sale si el pez no se pinta ***
    if ( i >= FotosPez + numPeces ) break;

```

```

// ##### Control x #####

```

```

// *** Llega al limite derecho ***
if (foto[i].x > 40 -1 && foto[i].velox > 0 && pezAsustado == False)
    foto[i].velox = foto[i].velox * -1;

```

```

// *** Llega al limite izquierdo ***
if (foto[i].x < -40 && foto[i].velox < 0 && pezAsustado == False)
    foto[i].velox = foto[i].velox * -1;

// *** Cambio de sentido al azar (evita cambios continuados) ***
if
(
    rand() % 100 < 0.2 && foto[i].invDir == True && pezAsustado == False &&
    (foto[i].velox > 0 && foto[i].veloActx > foto[i].velox * 0.8 ||
    foto[i].velox < 0 && foto[i].veloActx < foto[i].velox * 0.8)
)
    foto[i].velox = foto[i].velox * -1;

// *** Cambia la dirección con suavidad ***
// ref: http://www.cplusplus.com/reference/cmath/abs/
if ( foto[i].veloActx > foto[i].velox )
    foto[i].veloActx = foto[i].veloActx - fabs(foto[i].velox/40);

if ( foto[i].veloActx < foto[i].velox )
    foto[i].veloActx = foto[i].veloActx + fabs(foto[i].velox/40);

// *** Controla la rotación según su dirección ***
if (foto[i].veloActx > 0) { foto[i].rotax=0; foto[i].rotay=180; }
if (foto[i].veloActx <= 0) { foto[i].rotax=0; foto[i].rotay=0; }

// *** Cambia la rotación con suavidad ***
if ( foto[i].rotaActy > foto[i].rotay )
    foto[i].rotaActy = foto[i].rotaActy - 45;

if ( foto[i].rotaActy < foto[i].rotay )
    foto[i].rotaActy = foto[i].rotaActy + 45;

// *** Actualiza movimiento de der a izq ***
if (pezAsustado == False )
    if ( foto[i].rotaActz != 180 || fabs(foto[i].x) > 80)
        // *** Pez normal ***
        foto[i].x = foto[i].x + foto[i].veloActx;
    else
        // *** Pez muerto ***
        foto[i].x = foto[i].x + (foto[i].veloActx / 4.0f);
else
    // *** Pez asustado ***
    if ( fabs(foto[i].x) < 100 )
        foto[i].x = foto[i].x + foto[i].veloActx*40;

// ##### Control y #####

// *** llega al limite superior ***
if (foto[i].y > 15 && foto[i].veloy > 0 && pezAsustado == False)
    foto[i].veloy = foto[i].veloy * -1;

// *** Llega al limite inferior ***

if (foto[i].y < ( estanComiendo == False ) ? -15 : 5) && foto[i].veloy < 0 &&
pezAsustado == False)
    foto[i].veloy = foto[i].veloy * -1;

```

```

// *** Cambio de sentido al azar (evita cambios continuados) ***
if
(
    rand() % 100 < 0.6 && foto[i].invDir == True &&
    pezAsustado == False && estanComiendo == False &&
    (foto[i].veloy > 0 && foto[i].veloActy > foto[i].veloy * 0.8 ||
    foto[i].veloy < 0 && foto[i].veloActy < foto[i].veloy * 0.8)
)
    foto[i].veloy = foto[i].veloy * -1;

// *** Cambia la dirección con suavidad ***
if ( foto[i].veloActy > foto[i].veloy )
    foto[i].veloActy = foto[i].veloActy - fabs(foto[i].veloy/20);

if ( foto[i].veloActy < foto[i].veloy )
    foto[i].veloActy = foto[i].veloActy + fabs(foto[i].veloy/20);

// *** Actualiza movimiento superior e inferior ***
if (pezAsustado == False )
    if ( foto[i].rotaActz != 180 || fabs(foto[i].y) > 40)
        // *** Pez normal ***
        foto[i].y = foto[i].y + foto[i].veloActy;
    else
        // *** Pez muerto ***
        foto[i].y = foto[i].y + (foto[i].veloActy / 4.0f);
else
    // *** Pez asustado ***
    if ( fabs(foto[i].y) < 50 )
        foto[i].y = foto[i].y + foto[i].veloActy*40;

// ##### Control z #####

// *** Llega al limite superior ***
// No pongas valores positivos o los peces atravesaran el cristal.
if (foto[i].z > -4 && foto[i].veloz > 0 && pezAsustado == False)
    foto[i].veloz = foto[i].veloz * -1;

// *** Llega al limite inferior ***
if (foto[i].z < -20 && foto[i].veloz < 0 && pezAsustado == False)
    foto[i].veloz = foto[i].veloz * -1;

// *** Cambio de sentido al azar (evita cambios continuados) ***
if
(
    rand() % 100 < 0.6 && foto[i].invDir == True && pezAsustado == False &&
    (foto[i].veloz > 0 && foto[i].veloActz > foto[i].veloz * 0.8 ||
    foto[i].veloz < 0 && foto[i].veloActz < foto[i].veloz * 0.8)
)
    foto[i].veloz = foto[i].veloz * -1;

// *** Cambia la dirección con suavidad ***
if ( foto[i].veloActz > foto[i].veloz )
    foto[i].veloActz = foto[i].veloActz - fabs(foto[i].veloz/20);

if ( foto[i].veloActz < foto[i].veloz )
    foto[i].veloActz = foto[i].veloActz + fabs(foto[i].veloz/20);

```

```

// *** Actualiza movimiento hacia adelante o hacia el fondo ***
if (pezAsustado == False )
    if ( foto[i].rotaActz != 180 || fabs(foto[i].z) > 75)
        // *** pez normal ***
        foto[i].z = foto[i].z + foto[i].veloActz;
    else
        // *** pez muerto ***
        foto[i].z = foto[i].z + (foto[i].veloActz / 4.0f);
else
    // *** pez asustado ***
    if ( fabs(foto[i].z) < 100 )
        foto[i].z = foto[i].z - fabs(foto[i].veloActz * 80);

} // Fin del for para todos los peces.






// *** Pide el repintado de pantalla ***
glutPostRedisplay();

// *** Prepara la siguiente actualización ***
glutTimerFunc(ActMseg, muevePeces, 0);
}

```

#### 4. Peces con texturas, nadando en profundidad ( $z \neq 0$ ).

Todos los peces pueden nadar en profundidad, es en la carga de datos con la función “IniciaFotos” donde se indican las características de cada vez.

- Puede observarse que hay peces como el azul de ojos saltones  tiene sus velocidades “y” y “z” a cero, lo que significa que nada de derecha a izquierda y viceversa, coordenada “x”. En el se puede ver con claridad como desacelera al llegar a un extremo, gira y como vuelve a acelerar.
- El pez dormilón o mala cara  su única velocidad esta en el eje “y” por lo que esta constantemente subiendo y bajando en la pecera.
- Y nuestro simpático pez globo  tiene preferencia por irse del frente al fondo de la pecera, eje z.
- Nuestro pez billy el rápido es el azul  este se mueve a mas velocidad que el resto de peces, esto se consigue dando valores mas altos a las velocidades de sus 3 ejes.
- Estos tres peces  son amigos y donde va uno van los otros, esto es gracias a que controlamos con total exactitud su dirección y velocidad. Abriendo la posibilidad de crear bancos de peces, solo hay que ponerlo unos cerca de otros, con las mismas velocidades y desactivar la inversión de dirección aleatoria. Todo esto se puede hacer desde la función “IniciaFotos”. Cuidado hay 6 peces payaso en la pecera, 3 de ellos son amigos los otros no.

Con todo lo indicado anteriormente puedes hacer que reine la anarquía en la pecera haciendo que cada pez en una dirección y velocidades distintas, creando el caos. O construir patrones de navegación. Otra alternativa se usan una solución mixta como la que he elegido yo, en la que algunos peces nadan al azar y otros siguen algún patrón.



## 5. Añadir elementos dentro de la pecera

Aquí presentamos los objetos o elementos que se han ido añadiendo a la pecera para completarla y hacer la más atractiva.

### 5.1. Añadir objetos en 3D

Inicialmente se planteo la posibilidad de añadir objetos 3D, como bien comento Manolo, la pecera es un prototipo por lo que se puede añadir objetos como una tetera en 3D. Seria como tener una pecera con peces tetera (tetera 3D), peces espada (espada 3D) y peces globo (globo 3D).

Pero fue descargado categóricamente. El motivo fue que durante la creación de la comida intente pintar triángulos de colores en la pecera usando primitivas de dibujo OpenGL. Fue imposible, los triángulos no aparecían por pantalla, prueba de ello es la función “pintaPapel” que esta desechada. Al final tuve que hacer la comida también con texturas, al final quedo bastante bien.

### 5.2. Añadir peces con texturas

Aparte de los 3 peces que ya disponía la pecera y que presentamos:

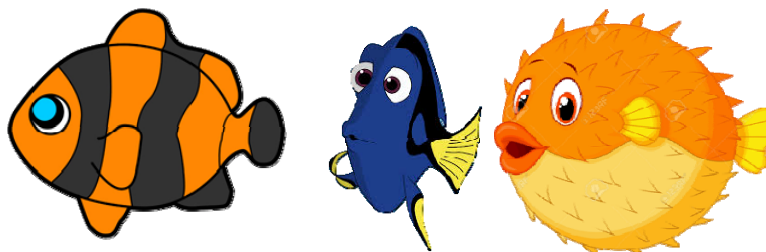


Ilustración 3-Peces originales

Se han añadido un gran numero de peces, algunos ya tenían transparencias (canal alfa) otro no. Y tuve que retocarlos con “Adobe Photoshop” para que incluyeran todos transparencias.

Y casi la totalidad de ellos hubo que reescalarlos, no tiene sentido tener peces en alta definición cuando en la pecera son muy pequeños. Además esto provocaba una sobrecarga en la pecera, especialmente en la carga de texturas. Algunos imágenes se reescalaros al 10% de su tamaño original.

También tuve que rotar algunos peces para que todos miren hacia la derecha, para evitar que algunos nadaran hacia atrás. Y poder tratar todos los peces por igual, sin tener que usar ángulos de rotación distintos según el pez.

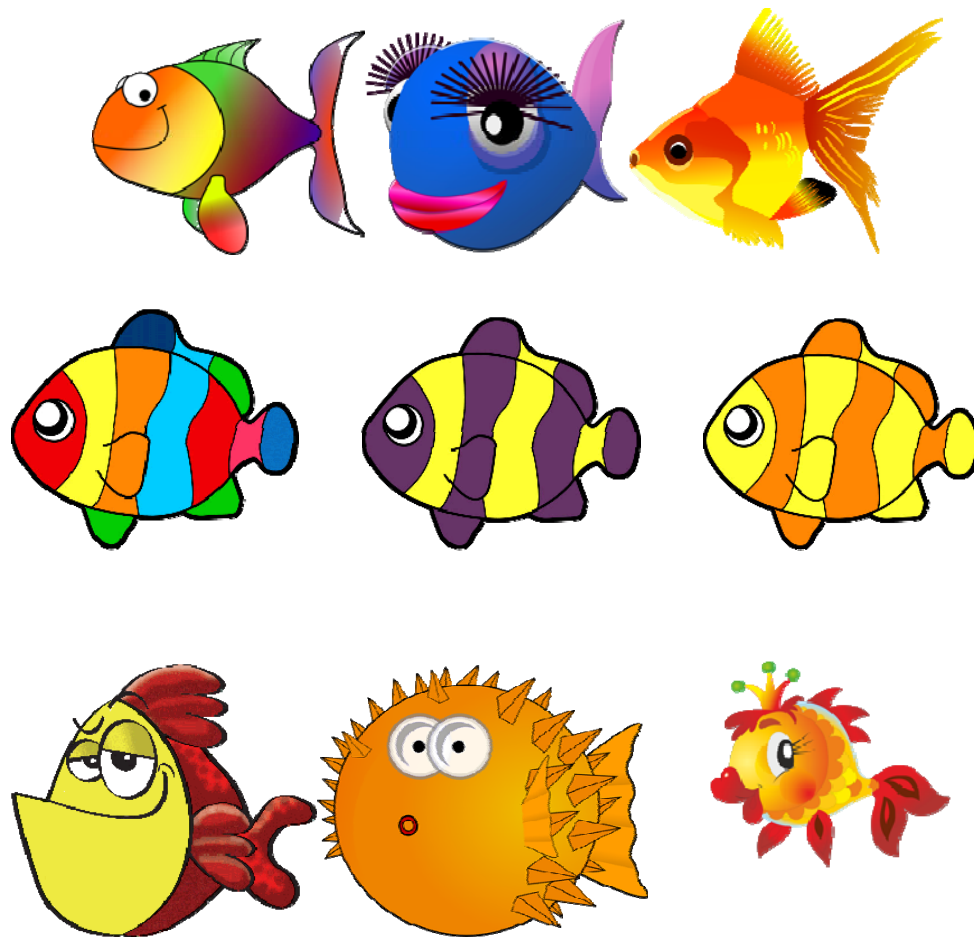


Ilustración 4-Peces añadidos

#### 5.2.1. Captura, pecera con peces

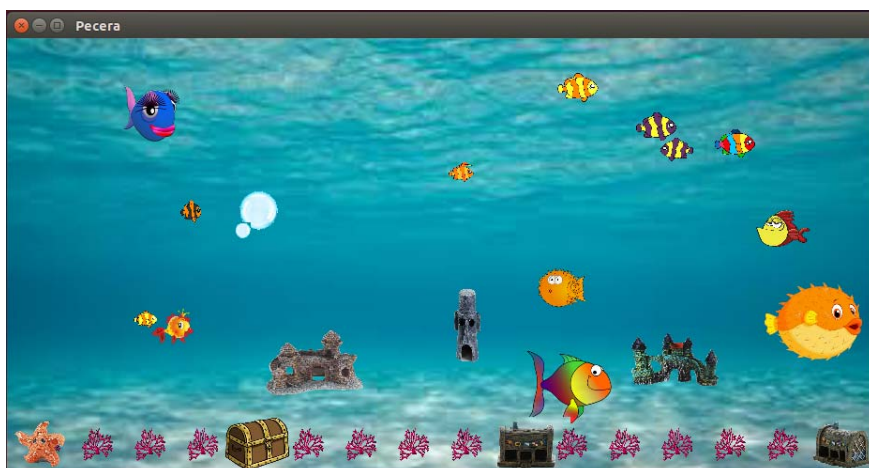


Ilustración 5-Pecera, con peces

### 5.3. Añadir sirena, dibujo vectorial

Mención especial hay que darle a la sirena, ella no es una imagen de mapa de bits sino que es un dibujo vectorial.

El “Adobe Photoshop” no podía abrirla pero si pudo “Adobe Illustrator”, es la primera vez que manejo este tipo de imágenes y resulto ser muy fácil su edición, ya que todos elementos de la imagen pueden tratarse como objetos o agrupaciones.

Por desgracia para cargarla como una textura en la pecera la tuve que convertir a una imagen “png”. La sirena vectorial y muchas otras imágenes también vectoriales, puede encontrarse en la referencia [[Sirena Vectorial](#)].



Ilustración 6-La sirena

#### 5.3.1. Captura, pecera con sirena

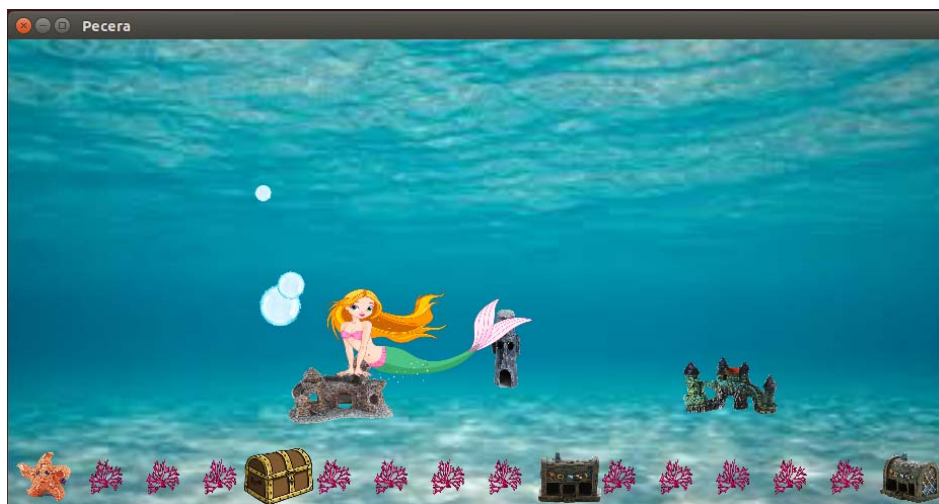


Ilustración 7-Pecera, con sirena

## 5.4. Añadir adornos al fondo

La pecera venia ya dotada con dos, un pequeño cobre y un coral.

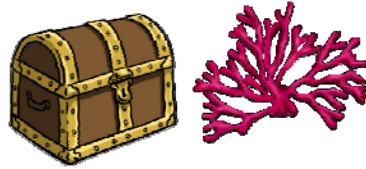


Ilustración 8-Adornos originales

He añadido unos cuantos adornos, pero el objetivo no era sobrecargar la pecera de adornos. Ya que el elemento que más debe destacar son los propios peces.

Por eso he añadido 6 nuevos elementos, en tiendas de complementos para peceras reales se pueden encontrar un gran repertorio de estos artículos. A tal efecto dejo esta referencia de la cual he sacado algunos de estos objetos [[Adornos Fondo](#)].

Al igual que con los peces, ha sido necesario editar los adornos con el “Adobe Photoshop” para dotarlos de transparencias, recortar lo que no se necesita y reescalar las imágenes para que no sea demasiados grandes.

Los dos castillos, disponen de zonas transparentes en un interior, lo que provoca un efecto curioso ya que es como si el agua pasara a través de ellos.



Ilustración 9-Adornos castillos



Ilustración 10-Adornos del fondo

#### 5.4.1. Captura, pecera con adornos

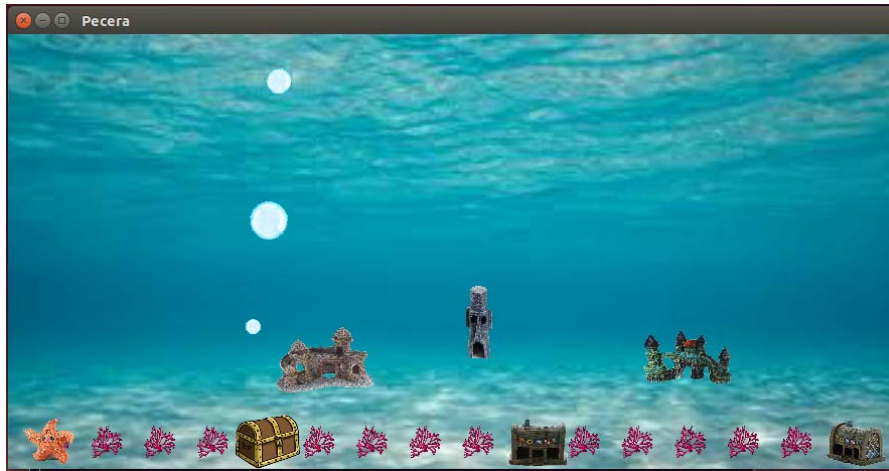


Ilustración 11-Pecera, con adornos

## 6. Mover burbujas y comida

Comentamos como se realizan los movimientos de las burbujas y de la comida de los peces.

### 6.1. Mover burbujas

Los compañeros del año pasado querían que los peces reventasen las burbujas cuando se las cruzasen.

En su lugar yo he decidido se sean un simple adorno para ambientar la pecera y darle mas personalidad. Pero no hay interacción entre ellas y los peces.

Lo que ha simplificado su realización, usa la misma textura pero con zoom distintos, su movimiento es lineal en el eje “y” y tiene forma circular, cuando salen por arriba entran otra vez por debajo. Pero se les ha dado velocidades distintas las burbujas mas pequeñas mas rápidas y las mas grandes mas lentas.

```
strcpy(foto[10].archivo, "foto/burbuja.png");
strcpy(foto[11].archivo, "foto/burbuja.png");
strcpy(foto[12].archivo, "foto/burbuja.png");

...

// *** Pone coordenadas de burbujal ***
x=10;
foto[x].x = -20; foto[x].y = -20; foto[x].z = -1;
foto[x].zoomx = 0.15; foto[x].zoomy = 0.15;
foto[x].velox= 0.0; foto[x].veloy= 0.6; foto[x].veloz= 0.0;
```



```

// *** Pone coordenadas de burbuja2 ***
x=11;
foto[x].x = -21; foto[x].y = -20; foto[x].z = -2.5;
foto[x].zoomx = 0.4; foto[x].zoomy = 0.4;
foto[x].velox= 0.0; foto[x].veloy= 0.2; foto[x].veloz= 0.0;

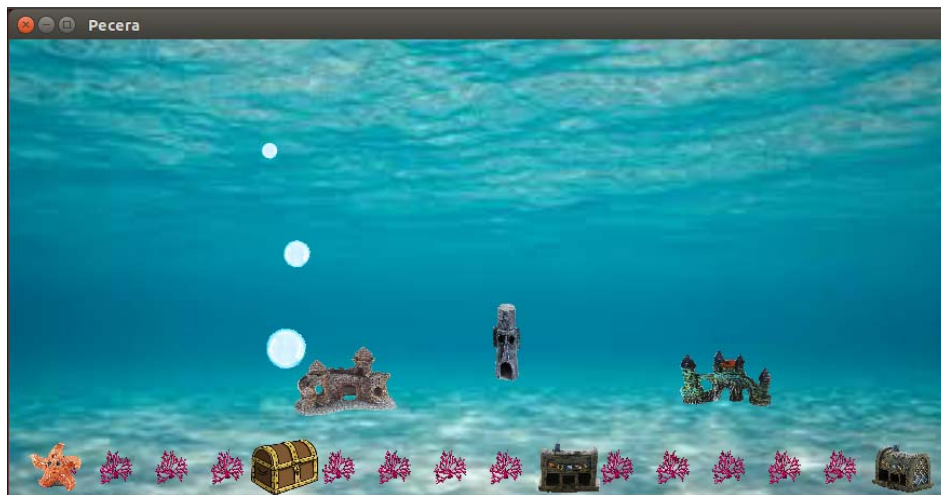
// *** Pone coordenadas de burbuja3 ***
x=12;
foto[x].x = -18; foto[x].y = -20; foto[x].z = -1;
foto[x].zoomx = 0.25; foto[x].zoomy = 0.25;
foto[x].velox= 0.0; foto[x].veloy= 0.4; foto[x].veloz= 0.0;

...

// ***** Mueve las burbujas *****
for (int i = FotosBru; i < FotosPez; i++)
{
    // *** llega al limite superior ***
    if (foto[i].y > 20 )
        foto[i].y = -20;
    else
        // *** Actualiza movimiento ***
        foto[i].y = foto[i].y + foto[i].veloy;
}

```

- **Captura de la pecera con burbujas.**



**Ilustración 12-Pecera, con burbujas**

## 6.2. Mover comida de peces

En cuando a la comida son 4 texturas que se reutiliza para cubrir los 200 trozos de comida que aparecen en la pecera.

```

strcpy(foto[29].archivo, "foto/Comida-Amarilla.png");
strcpy(foto[30].archivo, "foto/Comida-Azul.png");
strcpy(foto[31].archivo, "foto/Comida-Roja.png");
strcpy(foto[32].archivo, "foto/Comida-Verde.png");

```

La velocidad de “x” varía muy poco y puede ser positiva o negativa para que vaya a la derecha o izquierda. La velocidad “y” siempre es negativa para que vayan hacia el fondo.

Tanto las coordenadas como las velocidades se obtienen de forma aleatoria inicialmente se usaba rand, pero gracias a la referencia [[Num Aleatorios](#)] la sustituí por drand48 que me pareció mas sencilla y adecuada. Sin olvidarnos de la iniciación de la semilla aleatoria.

```
// *** Inicia la comida de los peces ***
void IniComida()
{
    estanComiendo = True;

    // rand() da valores entre el 0 y RAND_MAX (32767).
    // se ini semilla con "srand (time(NULL));" o "srand (getpid());"

    // drand48() da valores decimales entre el 0 y el 1, sin llegar al 1.
    // para iniciar semilla "srand48(time(NULL));" o "srand48(getpid());"
    // si no ini semilla repe los nº en cada nueva ejecución.
    // ref: http://www.chuidiang.com/clinix/funciones/rand.php
    srand48(time(NULL));

    for (int i=0; i < PapelesMax; i++)
    {
        comida[i].x = (drand48() * 80) - 40;
        comida[i].y = (drand48() * 15) + 21;
        comida[i].z = -1.5;
        comida[i].velox = -0.05 + (drand48() / 10); // da + y -
        comida[i].veloy = -0.10 - (drand48() / 10); // da - solo
        comida[i].veloz = 0;
        comida[i].zoomx = 0.2;
        comida[i].zoomy = 0.2;
        comida[i].sePinta = True;
    }
    printf("Damos de comer a los peces.\n");
}
```

Conforme va cayendo la comida esta va desapareciendo, simulando que los peces se la comen, lo hace un una probabilidad del 1%. Cuando se esta agotando el tiempo de comida esta probabilidad se incrementa hasta el 3%.

La comida desaparece del todo cuando se acota el tiempo de comida o llegan al fondo de la pecera.

```
// ***** Mueve la comida *****
if (estanComiendo == False)
...
    comida[i].x += comida[i].velox;
    comida[i].y += comida[i].veloy;
    comida[i].z += comida[i].veloz;

    if (comida[i].y < -15 || drand48() < ( (comerContador > 100) ? 0.01 : 0.03 ) )
        comida[i].sePinta = False;
```

El pintado de la comida es el habitual, solo que hay que transferir los datos del vector comida al vector foto para el pintado.

```
// *** Pinta la comida ***
if (estanComiendo == True)
...
    a = FotosComida + contaComida;

    foto[a].x = comida[i].x;
    foto[a].y = comida[i].y;
    foto[a].z = comida[i].z;

    foto[a].zoomx = comida[i].zoomx;
    foto[a].zoomy = comida[i].zoomy;
...
    pintaFoto (a);
}
contaComida = (contaComida + 1) % 4;
```

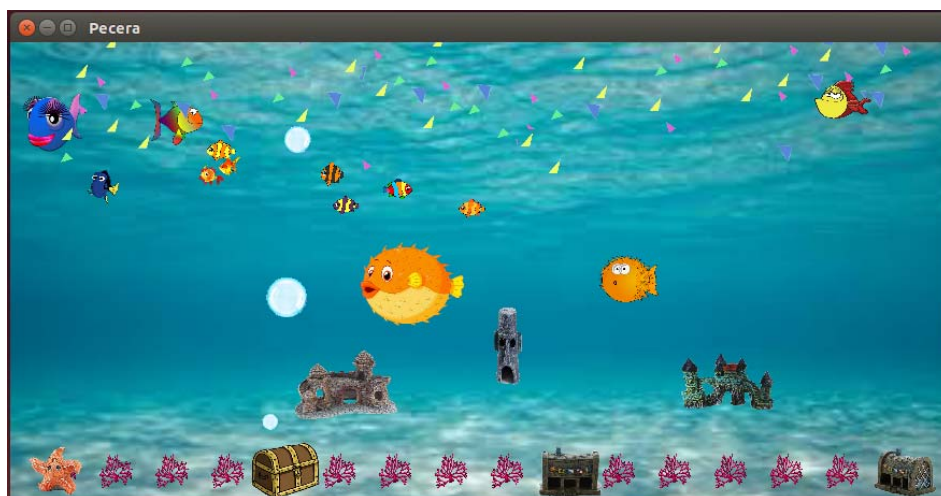
Recuerdo que ya se ha comentado que los peces suben a la mitad superior de la pantalla durante la comida.

```
// ##### Control y #####

// *** Llega al limite superior ***
if (foto[i].y > 15 && foto[i].veloy > 0 && pezAsustado == False)
    foto[i].veloy = foto[i].veloy * -1;

// *** Llega al limite inferior ***
if (foto[i].y < ( estanComiendo == False) ? -15 : 5) && foto[i].veloy < 0 && pezAsustado ==
False)
    foto[i].veloy = foto[i].veloy * -1;
```

- **Captura de la pecera con peces comiendo.**



**Ilustración 13-Pecera, con peces comiendo**



## 7. Incorporar interacción con usuario

Indicamos que técnicas avanzadas de interacción con el usuario se usan en la pecera, excluyendo las habituales como teclado y ratón.

### 7.1. Detección de cara

Consiste en realizar técnicas de visión por computador para detectar la posición y su seguimiento de objetos, como caras, manos, cuadrados, etc.

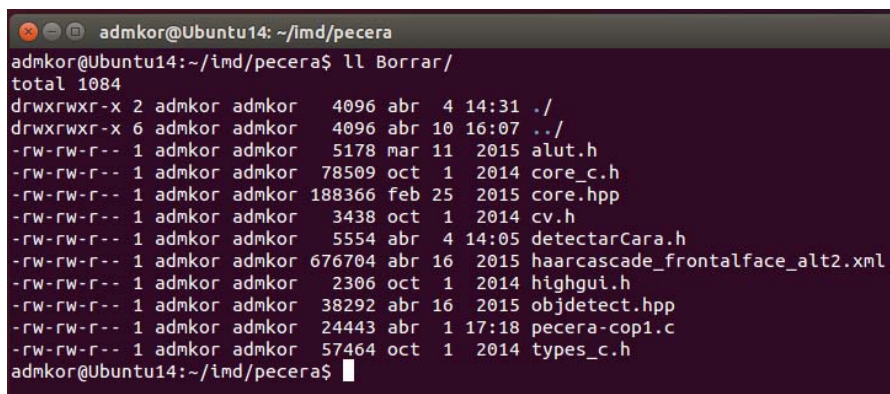
#### 7.1.1. Trabajo previo.

Inicialmente se planteo realizar una “detección de movimiento en la parte superior de la pecera para dar de comer a los peces” o “detectar la cercanía de la cara para asustar a los peces”.

En su lugar **decidí incluir** un objeto destacado **“una sirena” que te valla siguiendo por la pantalla**, en función de donde se detecte tu cara en al Webcams. Esto permite una **interacción mucho más atractiva** que las mencionadas anteriormente.

Por lo que no vi adecuado usar el planteamiento inicial. Para evitar que durante el seguimiento (tracking) de la sirena, los peces huyeran o empiecen a comer.

Para abordar este problema, fue necesario eliminar todo el detector de caras del año pasado, ya que producía muchos errores. Esto junto con la eliminación de los archivos de cabecera, innecesarios en Ubuntu. Ha permitido la eliminación de un gran número de archivos y la consecuente simplificación de la pecera.



```
admkor@Ubuntu14: ~/lmd/pecera
admkor@Ubuntu14:~/lmd/pecera$ ll
total 1084
drwxrwxr-x 2 admkor admkor 4096 abr  4 14:31 ./
drwxrwxr-x 6 admkor admkor 4096 abr 10 16:07 ../
-rw-rw-r-- 1 admkor admkor 5178 mar 11 2015 alut.h
-rw-rw-r-- 1 admkor admkor 78509 oct  1 2014 core_c.h
-rw-rw-r-- 1 admkor admkor 188366 feb 25 2015 core.hpp
-rw-rw-r-- 1 admkor admkor 3438 oct  1 2014 cv.h
-rw-rw-r-- 1 admkor admkor 5554 abr  4 14:05 detectarCara.h
-rw-rw-r-- 1 admkor admkor 676704 abr 16 2015 haarcascade_frontalface_alt2.xml
-rw-rw-r-- 1 admkor admkor 2306 oct  1 2014 highgui.h
-rw-rw-r-- 1 admkor admkor 38292 abr 16 2015 objdetect.hpp
-rw-rw-r-- 1 admkor admkor 24443 abr  1 17:18 pecera-cop1.c
-rw-rw-r-- 1 admkor admkor 57464 oct  1 2014 types_c.h
admkor@Ubuntu14:~/lmd/pecera$
```

Ilustración 14-Archivos borrados

#### 7.1.2. Ejemplo del detector de caras

Gracias a un ejemplo que encontré por Internet pude realizar satisfactoriamente el detector de caras. Este ejemplo funciono bien y a la primera por lo que pude adaptar lo para la pecera. El ejemplo puede localizarse en la referencia [[Ej FaceDetect](#)].

Esto permitió que con un solo archivo se dispusiera de un detector de caras operativo “facedetect.h”. También fue necesario descargar un nuevo archivo “haarcascade\_frontalface\_alt.xml” ya que el antiguo parecía estar corrupto o dañado.

Para que este detector de caras se integre con la pecera se le ha dotado de 3 nuevas funciones. Una para activar la Webcams, otra para desactivarla y una tercera para realizar la detección de caras.

### 7.1.3. Función “webCamOn”

La función “webCamOn” se encarga de activar la WebCam, inicia los datos y carga el esquema de la cara frontal.

```
// *** Activa la webCam ***
int webCamOn(){
    // *** Carga la caraFrontal **
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
    if( !cascade ) {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
        return False;
    }

    // *** Reserva memoria ***
    storage = cvCreateMemStorage(0);

    // *** Crea una ventana OpenCV ***
    cvNamedWindow( VentanaCV, CV_WINDOW_AUTOSIZE );
    cvMoveWindow( VentanaCV, 600, 0);

    // *** Abre la webcam ***
    capture = cvCaptureFromCAM(0);
    if (!capture) {
        fprintf(stderr, "Error: No se puede abrir la webcam.\n");
        return False;
    }
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 320);
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 240);

    printf("WebCam abierta.\n");
    return True;
}
```

### 7.1.4. Función “webCamOff”

La función “webCamOff” se encarga de cerrar la webCam y la ventana OpenCV.

```
// *** Desactiva la webCam ***
void webCamOff(){
    // *** Cierra ventana ***
    cvDestroyWindow(VentanaCV);
    cvWaitKey(25);

    // *** Cierra la cámara ***
    cvReleaseCapture( &capture );

    printf("WebCam cerrada.\n");
}
```

### 7.1.5. Función “webCamDetCara”

Y la última función “webCamDetCara” se encarga de recuperar una foto de la webCam y realizar la detección de caras. Las coordenadas del centro de la primera cara detectada se dejan en la variables globales “caraX” y “caraY”

```
// *** Desactiva la webCam ***
void webCamDetCara(){
    // *** Carga la foto ***
    //image = cvLoadImage( input_name , 1); // Saca foto desde archivo
    image = cvQueryFrame(capture); // Saca foto desde webCam
    if (!image) fprintf(stderr, "Error: no se carga la foto %s\n", input_name);

    // *** Detecta cara en foto ***
    detect_and_draw( image );
}
```

La variables “caraX” y “caraY” no tienen coordenadas como tales, tienen un tanto por uno, es decir un valor de 0 a 1, que representa en que porcentaje de la pantalla se encuentra la cara. Esto se realiza así ya que las resoluciones de las pantallas OpenCV y OpenGL son distintas.

```
int fotoAncho = img->width;
int fotoAlto = img->height;
...
if (i==0) {
    caraX = ((r->x + r->width*0.5)*scale) / fotoAncho;
    caraY = ((r->y + r->height*0.5)*scale) / fotoAlto;
}
```

### 7.1.6. Captura de la WebCam

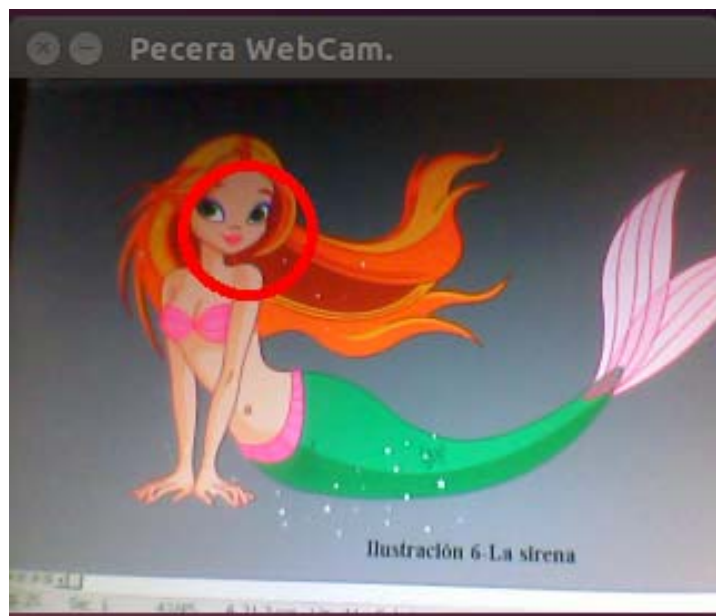


Ilustración 15-WebCam de pecera

Como se ha obtenido esta captura, apuntando con la webCam del portátil al monitor del equipo de sobremesa, con la memoria abierta. Así la sirena se sigue a si misma.

## **7.2. Un sonido fuerte asusta los peces**

Me hubiera gustado añadir este tipo de interacción.

Pero por falta de ejemplos que capturen sonido o lo usen para interacción. No me ha sido posible completar lo.

## **8. Ampliaciones**

Debido a que en el objetivo “Incorporar la interacción con el usuario” no se especificaron con exactitud que se iba a realizar. Procedemos a realizar una serie de ampliaciones para concretar los y ampliar la capacidad de interacción de la pecera, a través de técnicas de visión por computador.

- **Gestión inteligente de comida y sustos.**

Al dar de comer o asustar a los peces, se verifica que al menos este un pez nadando. Sino se emite un mensaje por consola. Y si se quitan los peces durante la comida, la comida se desactiva automáticamente.

- **Interacción por proximidad de la cara.**

He usado el radio del círculo del detector de caras para detectar la proximidad de la cara y asustar a los peces. Siempre que hallan peces nadando, en caso contrario no hace nada.

- **Interacción con movimiento en la parte superior.**

He añadido un detector de movimiento, basado en la actividad 4 de prácticas. Así se puede detectar movimiento en la parte superior de la pantalla e iniciar la comida de los peces. Si hay peces nadando, en caso contrario no hace nada. Tiene una ventana independiente y permite controlar su umbral.

## 9. Conclusiones y trabajos futuros

### 9.1. ¿Cuánto de los objetivos se ha alcanzado?

- Análisis del código anterior.  
Se ha realizado por completo y satisfactoriamente.
- Crear una estructura de datos para los "peces" y elementos visuales.  
Se ha realizado por completo y satisfactoriamente.
- Tener peces con texturas (fotos) nadando en 2D ( $z=0$ ), libre de errores.  
Se ha realizado por completo y satisfactoriamente.
- Añadir elementos a la pecera. (adornos, peces distintos, objetos 3D)  
Se ha realizado satisfactoriamente. Los objetos 3D no se han incluido, por problemas técnicos.
- Dotar de movimiento en profundidad a los peces ( $z \neq 0$ ).  
Se ha realizado por completo y satisfactoriamente.
- Incorporar la interacción con el usuario.  
Se ha realizado satisfactoriamente. La interacción por sonido no esta disponible.

### 9.2. ¿Qué se podría hacer a partir de aquí?

- Añadir interacción por sonido para asustar los peces con un grito.
- Dar de comer a los peces acercándoles algo de color verde.
- Sustituir la sirena por un avatar (chica o sirena en 3D).
- Hacer cantar a la sirena.
- Incluir un pescador que quiera pescar a la sirena o es la sirena la que pesca al pecador, no lo tengo muy claro.
- Introducir la pecera en un recipiente (cubo) de cristal y poner girar la pecera entera o cambiar su alguno de visión.

## 10. Bibliografía y referencias.

- [Instalar OpenAL] [Como instalar OpenAL](#). Última consulta abril 2016.
- [Doc OpenGL] [Documentación de OpenGL](#). Última consulta abril 2016.
- [Doc OpenCV] [Documentación de OpenCV](#). Última consulta abril 2016.
- [Sirena Vectorial] [Cartoon Mermaid 02 - Vector](#). Consulta abril 2016.
- [Adornos Fondo] [Sumérgete en la realidad](#). Última consulta abril 2016.
- [Num Aleatorios] [Números aleatorios en C](#). Última consulta abril 2016.
- [EjFaceDetect] [Ejemplo FaceDetect](#). Última consulta abril 2016.