TRABAJO IMD: PECERA
FECHA: 2014/2015

Alumnos:
Nombre: Xavier Calaf García **Correo:** xacagar@inf.upv.es
Nombre: Isabel Garví López **Correo:** igarvilo@gmail.com

Índice:

1. Ideas clave y objetivos

- Pasar la pecera a lenguaje C.
- Dibujarla en OpenGL.
- Añadir profundidad a la pecera.
- Dar de comer a los peces cuando se entra a la pecera en la parte superior.
- Intentar que los peces sean en 3D (Se complicó y nos ha sido imposible hacerlo).
- Dibujar las burbujas en la pecera simulando respiración peces.
- Dibujar corales, estrellas de mar, cofres y elementos típicos de una pecera.

2. Introducción

Al comienzo nos planteamos realizar y modificar una simulación de pecera que veríamos en la pantalla del ordenador. Ésta ya estaba hecha de años anteriores con unas ciertas características que nosotros modificamos añadiendo todas aquellas posibilidades que indicamos en los objetivos.

En este trabajo íbamos a usar el lenguaje OpenGL y C y añadir las características como darles de comer a los peces, añadir las burbujas como si respirasen los peces y una función que nos motivaba bastante pero nos fue complicado de realizar, que era que cuando la webcam nos detectara la cara los peces se alejasen de la pantalla como si se asustasen al ver nuestra presencia.

3. Desarrollo.

Comenzamos este proyecto diseñando la pecera en OpenGL, pues anteriormente esta pecera estaba diseñada con OpenCV.
Primero importamos las librerías necesarias para que funcione:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <GLUT/glut.h>
#include <OpenGL/gl.h>
```

3.1 Estructura y jerarquía del texto

Comenzamos construyendo el main, en el que inicializamos la glut, los buffers (frame buffer y z-buffer, que añade profundidad).

Después de esto, creamos la ventana llamada pecera a tamaño ventana completa.

También en el main, registramos los callbacks:

```
// Registro de callbacks
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutSpecialFunc(onSpecialUp);
   glutKeyboardFunc(onKey);
   glutMouseFunc(mouse);
   glutIdleFunc(update);
```

Y creamos un menú asociado al botón derecho del ratón. Las opciones de este menú son: Añadir pez (funcionalidad no implementada totalmente), añadir burbuja (funcionalidad no implementada totalmente), usar opencv y visualizar cámara (OpenCV no funciona, anotado en el menú), y salir.

Después de esto, llamamos a Init para iniciar el programa.

Para el fondo de la pecera hemos usado una imagen, que cargamos mediante la librería 'freeimage'. Para ello, inicializamos freeimage en el main.
En el método cargarTexturas(), cargamos todas las imágenes, tanto del fondo como de los peces, burbujas, cofre y coral. Todo lo hacemos del mismo modo:

```
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);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

Previamente se han declarado las variables fish_1, fish_2, fish_3, coral, fondo y cofre

como GLuint.
En el método reshape, tenemos el código para cuando redimensionamos la ventana. También aquí está la cámara, que tiene un ángulo de 90º, lo más cerca que se ve es 0.1 y lo más lejos 1000.

Con esto tenemos la pecera diseñada. Ahora añadimos cosas.
Para dibujar los peces, el coral y el cofre, se ha dibujado un cuadrado (para cada elemento, en una posición) y se ha 'bindeado' la imagen cargada.

```
glBindTexture(GL_TEXTURE_2D, fish_1);
  glPushAttrib(GL_COLOR_BUFFER_BIT);
  glBegin(GL_QUADS);
    glTexCoord2d(0,0);
    glVertex3f(-6, -8, -1.0);
    glTexCoord2d(0, 1);
    glVertex3f(-6, -5, -1.0);
    glTexCoord2d(1,1);
    glVertex3f(-2, -5, -1.0);
    glTexCoord2d(1,0);
    glVertex3f(-2, -8, -1.0);
  glEnd();
  glPopAttrib();
```

Inicialmente se tuvo un problema, no salían las texturas. Esto se solucionó añadiendo la función glTexCoor2d(), (solución proporcionada por Ignacio Ballester Tester). Lo que hace esta función es poner la coordenada 0,0 de la imagen en la esquina inferior izquierda del quad, la coordenada 0,1 en la esquina superior izquierda del quad, la 1,1 a la esquina superior derecha y la 1,0 a la inferior derecha. (esto puede variar según la imagen, pero el principio es el mismo, poner cada coordenada de la imagen en una esquina del quad). En un principio, cometí un error y repetí la coordenada 1,0, haciendo que el pez no se viera correctamente, se ha corregido en la versión 2 de la pecera.

Se han añadido dos métodos para dibujar peces y burbujas aleatoriamente, pero todavía no funcionan correctamente.

Para dar movimiento a los peces se ha utilizado el método update, igual que para realizar la interacción con las burbujas.
Para el movimiento, se han utilizado nuevas variables, declaradas previamente, en la que si esta variable (en el caso de los peces que se mueven hacia la izquierda) es mayor que -60, se decremento en 0.4 para el pez 1 y en 0.2 para el pez 3. Para el pez que va hacia la derecha, si la variable es menor que 60, se incrementa en 0.2.
Las burbujas se desplazan ambas hacia arriba, pero la grande más despacio que la pequeña. Si la variable es menor que 40, se incrementa en 0.2 la grande y 0.4 la pequeña.

Esto se le pasa al método display con la función **glutPostRedisplay()**

```
if(drawPez){
    glPushMatrix();
        glTranslatef(x_fish1, 0, 0);
        drawFish1();
    glPopMatrix();

    glPushMatrix();
        glTranslatef(x_fish2, 0, 0);
        drawFish2();
    glPopMatrix();

    glPushMatrix();
        glTranslatef(x_fish3, 0, 0);
        drawFish3();
    glPopMatrix();
}

drawChest();
drawCoral();
```

Con esto se consigue que se traslade el pez y se redibuje, y por tanto da la sensación que el pez se desplaza.
El if(drawPez), es para cuando usamos OpenCV y nos acercamos a la cámara, los peces se asustan y desaparecen. La variable drawPez está inicializada a 1.

Como hemos dicho antes, en el método update también está implementada la interacción entre peces y burbujas. Hay que decir que para detectar la colisión no se ha tenido en cuenta la profundidad de peces y burbujas, pues todos ellos se mueven en una profundidad fija y no colisionarían nunca.

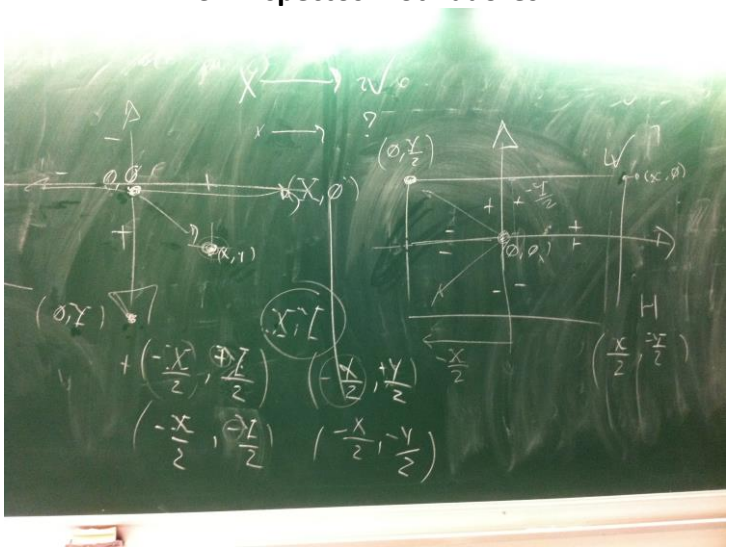
En general, la idea es, cuando el pez está dentro de las coordenadas x e y de las burbujas, la burbuja 'explota'. En realidad lo que tenemos es una variable drawBubblem = 1 y una variable drawBubbles = 1 que ponemos a 0 cuando pez y burbuja colisionan, también ponemos la y de las burbujas a 0 para que cuando se vuelvan a dibujar, salgan desde abajo y no desde donde se ha borrado. Esto lo pasamos al método display donde tenemos un if(drawBubblem) y un if(drawBubbles) que cuando las variables están a 0, no se cumple y por tanto la burbuja no se dibuja.
Después se ha añadido al método update dos if's para volver a poner las variables drawBubblem y drawBubbles a 1.

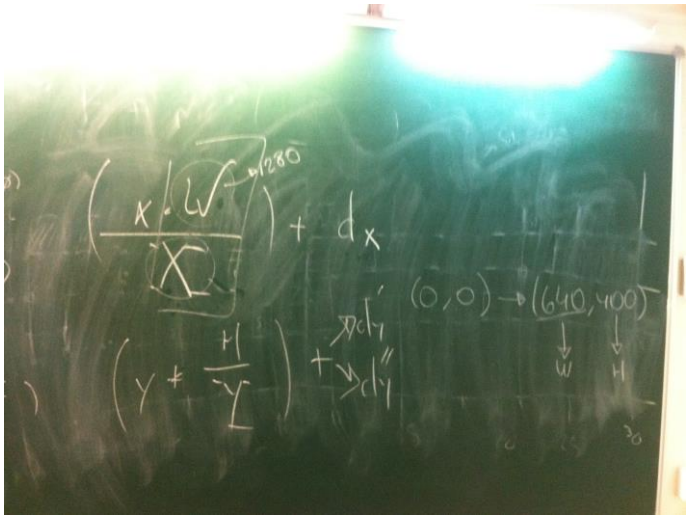
```
if(drawSmaBubble == 0){
    drawSmaBubble = 1;
}

if(drawMedBubble == 0){
    drawMedBubble = 1;
}
```

Una vez explicado esto, pasamos a explicar nuestra interacción con la pecera.
Como se ha explicado antes, tenemos un menú que aparece cuando hacemos click con el botón derecho del ratón.
Siguiendo con el ratón, tenemos un método mouse(button, state, x, y) en el que registramos cuando se clica con el botón izquierdo del ratón. Cuando esto sucede, pueden pasar dos cosas:
Si el click que se registra es en la parte superior de la pantalla (y < 0.05*alto), los peces deben subir a esa parte de la pantalla y comer. Si esto no sucede, los peces deben acudir a las coordenadas x e y de la ventana de dibujo.
Con este método tuvimos problemas, porque OpenGL trabaja con dos ventanas, la real y la de dibujo. Nuestra ventana real es de 1280*800, mientras que la de dibujo trabaja con las x de -38.5 a 38.5 y las y entre -26 y 26. Preguntamos al profesor por este problema y esta fue su solución:

3.2 Aspectos motivadores





Implementamos esas ecuaciones, pero no funcionaba correctamente, pues nos daba una x muy alta.
Finalmente, y después de preguntar a un compañero dimos con la solución (proporcionada por Ignacio Ballester Tester).

```
x1 = ((x- (ancho/2)) / X1);
y_nueva = -((y- (alto/2)) / Y1);
```

Siendo X1 = ancho/77 e Y1 = alto/52.

Estos dos datos se lo pasamos a las funciones pecesComen y pecesCerca.
La función pecesComen no está implementada porque no hemos conseguido implementar correctamente la función pecesComen, y ambas comparten el mismo principio.

```
void pecesCerca(int x, int y){
    // los peces se acerca a la zona donde se ha clickado
    mover = 1;
    drawPez = 0;
    if(u+x_fish1-2 != x){
        if(u < x){
            u = u + 1;
        }else{
            u = u - 1;
        }
    }
    if(w != y){
        if(w < y){
            w = w + 1;
        }else{
            w = w - 1;
        }
    }
    printf("u: %f, w: %f\n", u, w);
    glutPostRedisplay();
}
```

Lo que hemos intentado hacer es: si la variable nueva = 0 + x del pez no es igual a la coordenada x proporcionada por la función mouse, la variable incrementa o decrementa dependiendo de si la variable es menor o mayor que x.
(Si da tiempo a implementarla correctamente antes de la presentación, me gustaría tenerla hecha).
Esto se pasa al **display** con mover = 1, y da la sensación que el pez se desplaza. De momento, con cada click que hacemos, el pez se desplaza ese incremento, pero no lo hace de continuo como debería ser.

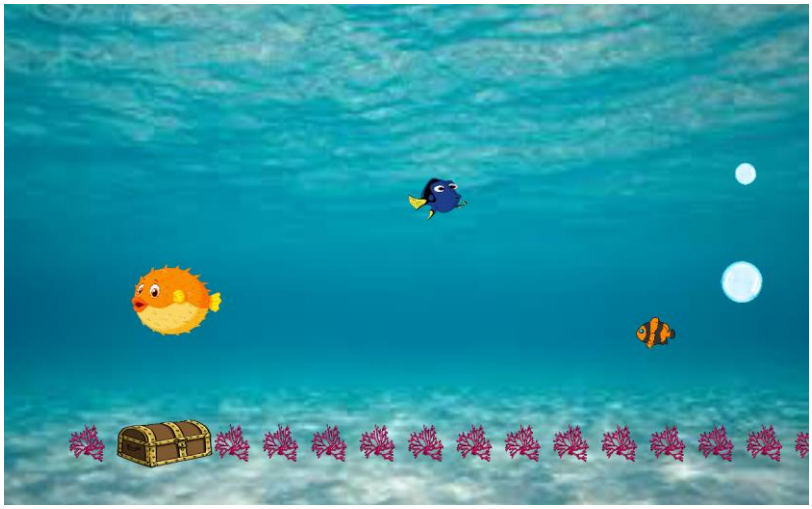
Siguiendo con la interacción persona-pecera, tenemos el método **onKey**, que registras pulsaciones en el teclado. En nuestro caso, tenemos 2 opciones:
Si pulsamos **ESC**, **Q** o **q**, salimos de la ejecución.
Si pulsamos **a** o **A**, nos lleva a la función drawFish para dibujar un pez al azar.

También tenemos el método **onSpecialUp,** que registra cuando pulsamos alguna flecha del teclado. En este caso, si pulsamos la flecha hacia arriba, nos lleva a la función drawFish para dibujar un pez al azar.

Además, para interactuar con la pecera, contamos con OpenCV. Para poderlo utilizar, hemos construido un cabecera llamada '**detectarCara.h**' que incluimos al principio del código.
En esta cabecera hay un método llamado **detectarCara**, en el que iniciamos la webcam y, si el usuario quiere y lo indica desde el menú, se muestra lo que la webcam ve. A partir de aquí, se identifica la cara que está más cerca y sacamos la proximidad. De vuelta en **pecera.c** comprobamos si la proximidad es mayor a 500, es que la persona está muy cerca y los peces se asustan (llamamos a pecesAsustados, donde los peces deben desaparecer).

4. Conclusión

Para finalizar aquí podemos ver una captura de pantalla de lo que la pecera es; como se puede ver hemos dibujado un fondo de pantalla simulando el fondo de una pecera, tres tipos de peces distintos tanto en forma como en tamaño, que es la parte dinámica de ésta, al igual que las burbujas que sales del lateral que simula la respiración de los peces. Por otro lado en la parte inferior de la misma hemos colocado unos corales y un cofre como elementos típicos que podemos ver en algunas peceras.



Como conclusión al trabajo y problemas que hemos tenido a lo largo de su creación es que no hemos podido hacer que OpenCV funcione, pues compilamos pero al ejecutar obtenemos el siguiente error:
OpenCV Error: Unspecified error (size node is not a valid sequence.) in icvReadHaarClassifier, file /opt/local/var/macports/build/_opt_mports_dports_graphics_opencv/opencv/work/opencv-2.4.11/modules/objdetect/src/haar.cpp, line 2072
libc++abi.dylib: terminating with uncaught exception of type cv::Exception: /opt/local/var/macports/build/_opt_mports_dports_graphics_opencv/opencv/work/opencv-2.4.11/modules/objdetect/src/haar.cpp:2072: error: (-2) size node is not a valid sequence. in function icvReadHaarClassifier

Esto proviene de la función que usamos para leer el archivo 'Haarcascade_frontalface_alt.xml', pero no hemos sabido solucionar el problema (Preguntamos al profesor).
Además, si se muestra la pantalla con la imagen que ve la webcam, la pecera va muy despacio y se acaba parando.

Ahora pasamos al sonido.
En una cabecera llamada '**sonidoBubble.h**' incluida al principio del código se ha implementado el código necesario para que suene un burbujeo y no pare hasta que finalizamos la ejecución.
Se ha intentado compilar de varias maneras:

g++ pecera.c -o pecera_IMD -framework OpenGL -framework GLUT -lfreeimage -I /opt/local/include `pkg-config opencv --libs` -I /usr/local/include -L/usr/local/lib -lalut

g++ pecera.c -o pecera_IMD -framework OpenGL -framework GLUT -lfreeimage -I /opt/local/include `pkg-config opencv --libs` -I /usr/local/include `freealut-config --libs`

g++ pecera.c -o pecera_IMD -framework OpenGL -framework GLUT -lfreeimage -I /opt/local/include `pkg-config opencv --libs` -I /usr/local/include `freealut-config --libs --cflags`

g++ pecera.c -o pecera_IMD -framework OpenGL -framework GLUT -lfreeimage -I /opt/local/include `pkg-config opencv --libs` $(freealut-config --cflags) -lalut

g++ pecera.c -o pecera_IMD -framework OpenGL -framework GLUT -lfreeimage -I /opt/local/include `pkg-config opencv --libs` -I /usr/local/include -lalut

g++ pecera.c -o pecera_IMD -framework OpenGL -framework GLUT -lfreeimage -I /opt/local/include `pkg-config opencv --libs` -I /usr/local/lib `pkg-config freealut --libs` -lfreealut

Pero no hemos conseguido compilar y nos da el siguiente error:

Undefined symbols for architecture x86_64:
"_alDeleteBuffers", referenced from:
bubbleSound() in pecera-7082d1.o
"_alDeleteSources", referenced from:
bubbleSound() in pecera-7082d1.o
"_alGenSources", referenced from:
bubbleSound() in pecera-7082d1.o
"_alGetSourcei", referenced from:
bubbleSound() in pecera-7082d1.o
"_alSourcePlay", referenced from:
bubbleSound() in pecera-7082d1.o
"_alSourcei", referenced from:
bubbleSound() in pecera-7082d1.o
"_alutCreateBufferFromFile", referenced from:
bubbleSound() in pecera-7082d1.o
"_alutExit", referenced from:
bubbleSound() in pecera-7082d1.o
ld: symbol(s) not found for architecture x86_64

Tenemos una cabecera llamada '**pecera.h**' en la que tenemos dos métodos. El método texturarFondo sirve para poner la imagen cargada como fondo.
El método loadImageFile, sirve para cargar la imagen para posteriormente usarla como textura.

La orden de compilación de este trabajo es
g++ pecera.c -o pecera_IMD -framework OpenGL -framework GLUT -lfreeimage -I /opt/local/include `pkg-config opencv --libs`

5. Bibliografía

5.1 Referencias de fuentes electrónicas:

5.1.1 https://www.opengl.org
https://github.com/vancegroup/freealut

5.1.2 http://opencv.org

5.1.3 https://www.opengl.org
https://github.com/vancegroup/freealut

5.2 Compañeros de clase:

5.2.1 José Juan Preciado.

5.2.2 Giovanni

5.2.3 Ignacio Ballester Tester