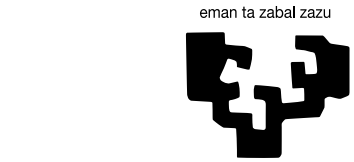


Documentación de Práctica 1

Leroy Deniz

September 30, 2020

Abstract

La aplicación permite mapear los puntos correspondientes a una imagen en una serie de triángulos que van a ir imprimiéndose en la pantalla. La imagen irá mostrándose en cada triángulo en función de los puntos correspondientes a cada lado del triángulo, lo que permitirá ver el trozo visible de imagen en diferentes posiciones, siempre enmarcadas en una figura.

1 Objetivos de nuestra aplicación

La aplicación tiene por objetivo mapear la textura de una imagen en formato *ppm* en una serie de triángulos que serán visibles de a uno en una ventana de 500x500 píxeles. La línea base de la imagen se corresponderá siempre con el segmento de la base del triángulo. Luego, el resto de la figura tendrá como textura su correspondiente píxel de la foto.

2 Requerimientos funcionales

Para poder compilar y ejecutar esta aplicación, hacen falta una serie de librerías que se deben instalar, para este caso utilizaremos como sistema oeprativo Debian 10, distribución gratis de Linux.

Instalación de las librerías de OpenGL.

```
$ aptitude install freeglut3 freeglut3-dev
```

Instalación de la suite *ImageMagik* para edición de imágenes en modo consola.

```
$ aptitude install imagemagik
```

Instalación del compilador *gcc* para linux

```
$ aptitude install gcc
```

Una vez instaladas todas las dependencias, la instrucción para compilar la aplicación, estando en el directorio donde se encuentran los archivos, es:

```
$ gcc *.c -lGL -lGLU -lglut -lm -o app
```

3 Interacción usuario-aplicación

La aplicación una vez ejecutada, interactúa con el usuario a través de las siguientes teclas:

1. Tecla *Esc*

   Cierra la ventana y finaliza la aplicación.

2. Tecla *Enter*

   Indica a la aplicación que elimine el triángulo que está mostrando e imprima el siguiente en la lista.

4 Estructura de ficheros

Para poder compilarse correctamente y poder ser ejecutada, la aplicación debe tener los siguientes ficheros:

1. dibujar-puntos.c

   Contiene el programa principal que permitirá mostrar la ventana de aplicación con su funcionalidad.

2. cargar-ppm.c

   Responsable de tomar cada pixel de la imagen en ppm y cargarlos en el buffer para, una vez iniciado el programa, poder acceder a los datos de la textura.

3. cargar-triangulos.c

   Toma del fichero *triangles.txt* los datos de cada triángulo para poner a disposición de la ventana de la aplicación.

4. triangulo.h

   Contiene la definición de la estructura *punto* e *hiruki*.

5. triangles.txt

   Contiene los datos de todos los triángulos a mostrar. Por cada fila tiene la información de las coordenandas de sus tres vértices y texturas (x, y, z, u, v);

6. foto.ppm

   Archivo de imagen en formato *ppm*.

5 Estructuras de datos

```
typedef struct punto
{
    float x, y, z, u,v;
} punto;
```

El tipo de estructura *punto* se constituye con cinco atributos float, cada uno representa lo siguiente:

1. x - posición del punto en el eje OX.
2. y - posición del punto en el eje OY.
3. z - posición del punto en el eje OZ. Como trabajamos en dos dimensiones, este punto no es relevante.
4. u - coordenada x de la textura de la imagen original.
5. v - coordenada y de la textura de la imagen original.

```
typedef struct hiruki
{
    punto p1,p2,p3;
} hiruki;
```

El tipo de estructura *hiruki* contiene los tres puntos que representan la posición de cada vértice del triángulo.

6 Funciones relevantes

```
unsigned char * color_textura(float u, float v)
```

Es la función responsable de tomar el pixel que corresponde de la imagen original y devolver un puntero al mismo. Este será utilizado en la función que pinte el triángulo.

```
void dibujar_triangulo(hiruki* triangulo_ptr)
```

La función recibe un puntero a una estructura de tipo triángulo que, mediante sus atributos, contiene las direcciones de sus vértices. En primer lugar identifica y posiciona cada uno de los puntos según su posición en el eje OY, para tomar el punto superior, medio e inferior.

Luego, calcula las pendientes de cada una de las rectas determinadas por el los vértices del triángulo, para luego calcular los puntos de corte de cada uno de los segmentos de recta. Dibuja todos los pixels de cada segmento entre la entrada y la salida con su textura correspondiente a la de la imagen, utilizando la función *segmentos*, (divide el triángulo en dos subtriángulos, cuya arista común es la línea horizontal que pasa por el punto medio del triángulo).

```
void segmentos(punto* primer_punto, punto* segundo_punto)
```

La función *segmentos* es quien pinta el contenido del triángulo, línea a línea por pixels. Recibe dos punteros, uno de la entrada al segmento y otro de la salida. Calcula la pendiente del segmento en la imagen original y trae toda la línea de píxels para dibujarla en el nuevo triángulo.

```
static void marraztu(void)
```

La función *marraztu* es quien dibuja el triángulo haciendo uso de la función *dibujar_triangulo*.

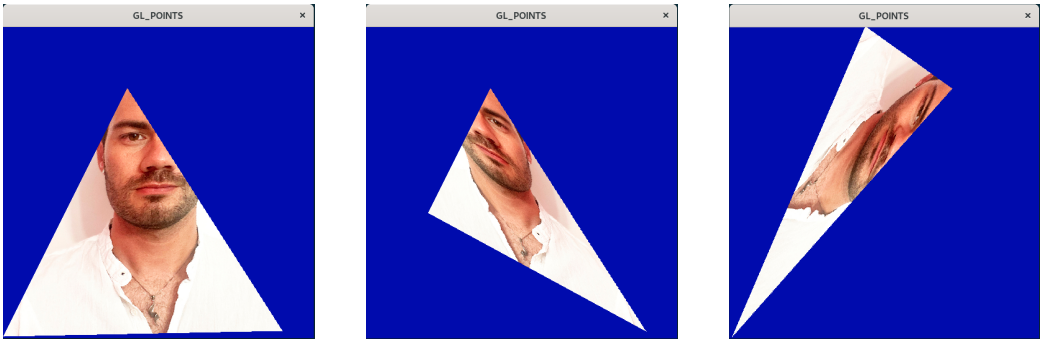
```
static void teklatua (unsigned char key, int x, int y)
```

La función *teklatua* recibe las órdenes del usuario; si se presiona Enter (cod: 23), se elimina el contenido de la ventana y se imprime el siguiente triángulo de la lista; si se presiona la tecla Esc (cod: 27), la aplicación finaliza.

7 Capturas de la aplicación

Imagen original que definirá la textura de los triángulos.

Ahora bien, una vez sea ejecutada la aplicación, adjuntamos aquí tres capturas de pantalla de lo que se verá en tres triángulos distintos.



(a) Triángulo 1 (b) Triángulo 2 (c) Triángulo 3

Figure 1: Vistas de la aplicación