Práctica 1 Gráficos por Computador

Pau Soler Valadés

September 30, 2020

Abstract

Documentación de la práctica 1 de *Gráficos por Computador* donde se explica la implementación, funcionamiento y proceso de desarrollo de una aplicación en C la cual plasma una fotografía en un triangulo mediante la librería *OpenGL*.

1 Introducción

Esta aplicación implementa un sistema en C usando la librería *OpenGL* mediante la cual podremos dibujar los triangulos introducidos en el fichero *triangulos.txt* en una ventana de 500x500 píxeles. Sobre dicho tirangulo, tiene que aparecer una fotografía del bello rostro del autor de este documento, aplicando las transformaciones a la imagen requeridas según le dictemos al programa en *triangulos.txt*. Usaremos el botón ENTER para pasar al siguiente triangulo y ESC para salir de la aplicación.

Para hacerlo se han puesto a nuestra disposición los ficheros *cargar-triangulos.c* y *cargar-foto.c* que cargan el triangulo de *triangulo.txt* a un struct triangulo con tres struct puntos con 5 valores cada uno (x,y,z,u,v) y cargan los píxeles de la foto desde la esquina superior derecha hasta la inferior izquierda con el formato (r,g,b) respectivamente. En este documento no nos centraremos en el funcionamiento interno de esos programas, sinó exclusivamente en el de *dibujar-triangulos.c*.

Dicho esto, este último fichero está comentado en inglés con detalle qué recibe y qué hace cada función. Me disculpo de antemano si ha quedado algún comentario en Castellano o otro que no se entiende a primera vista: no es que no sepa escribir, sinó que están en Catalán.

2 Funcionamiento

Al llamar al *main* de la aplicación se inicializan tres variables globales:

- **indice**: un int que controla a qué triangulo de la aplicación estamos apuntando.
- **num_triangulos**: un int que dice cuántos triangulos hay en *triangulos.txt*
- **triangulosptr**: un puntero hiruki que apunta al primer elemento de una tabla dónde se han almacenado los hirukis leídos de *triangulos.txt*.

Después tenemos las funciones para inicializar openGL debidamente, de las que cabe destacar *glutDisplayFunc(marraztu)* y *glutKeyboardFunc(teklatua)* siendo la primera controlada por GLUT llamando a la función *marraztu* y la segunda reaccionado al evento de llamada por teclado, llamando a *teklatua*.

En *teklatukoa* controlamos que cuando se pulse ENTER indice augmente uno modulo *num_triangles*, así nunca pasará ese valor. En *marraztu* se llama a la función *dibujar_triangulo*, el tronco de la aplicación:

```
void dibujar_triangulo()
{

  int h;
  punto *Aptr, *Bptr, *Cptr;
  punto pin_left, pin_right;

  determinar_orden(&Aptr, &Bptr, &Cptr);

    //the first for goes from A-> (the nearest from 0) to B->y.
  for(h=Aptr->y; h<=Bptr->y; h++)
  {
    calcular_interseccion(Aptr, Cptr, &pin_left, h);
    calcular_interseccion(Aptr, Bptr, &pin_right, h);

    if(pin_left.x < pin_right.x)
      linea_triangulo(pin_left, pin_right, h);
    else
      linea_triangulo(pin_right, pin_left, h);
  }
  //the second goes from B to C, thus making the h all the way up to 500
  for(; h<=Cptr->y; h++)
  {

    calcular_interseccion(Cptr, Bptr, &pin_left, h);
    calcular_interseccion(Cptr, Aptr, &pin_right, h);

    if(pin_left.x < pin_right.x)
      linea_triangulo(pin_left, pin_right, h);
    else
      linea_triangulo(pin_right, pin_left, h);
  }
}
```

Empezamos con los tres punteros Aptr, Bptr, Cptr: estos van a apuntar a los tres vértices del triangulo, pero para dibujarlos debemos tenerlos en orden, así que llamamos a *determinar_orden* pasando por parámetro la dirección de memória del puntero para que la modificación en la nueva función cambie también los originales.

2.1 Determinar Orden

Esa función es una adaptación de la típica función de cómo ordenar tres números de menor a mayor. Primero assignamos cada puntero de manera aleatória a uno de los vértices del triangulo al que apunta *triangulosptr[indice]* (al ser la variable global, pasarla por parámetro solo la haría más lenta), después comprovamos:

1. Es A.y mayor que C.y? Si lo es cambia el puntero de A a C y viceversa. Si no déjalos igual.

2. Es A.y mayor que B.y? Si lo es cambia el puntero de A a B y viceversa. Si no déjalos igual.

3. Es B.y mayor que C.y? Si lo es cambia el puntero de B a C y viceverda. Si no déjalos igual.

Sé que hay otra manera más eficiente, que es directamente assignar los puntos a los valores a medida que los vayas necesitando (la proporcionada en clase) pero esta manera me parece más entendedora.

Una vez terminada la función disponemos de los vértices A, B y C del triangulo ordenados del más cercano al eje de las abscisas (A) al más lejano (C), siendo el restante (B) el otro puntero.

Una vez tenemos los vértices ordenados, podemos proceder a dibujar el triangulo. Para hacerlo lo partiremos en dos trozos: dibujaremos todas las líneas desde la altura del vértice A hasta la altura del vértice B (primer for) y después desde la altura del vértice B hasta la del C (segundo for). Para no ir a ciegas, para cada altura comprendida entre dos vértices, encontramos el valor más pequeño de x que pertenece al triangulo y el mayor que también lo hace para dibujar los píxeles de entremedio. Esto lo hacemos en *calular_interseccion*

2.2 Calculando Intersección

Recibimos como argumentos los dos puntos por los que pasa el lado del triangulo correspondiente (en la primera mitad son A y C con A y B y en la

segunda C con B y C con A) la altura del punto que queremos calcular y un puntero dónde guardarnos el valor. Para calcular el valor de x es tan sencillo como hacer interpolación, o una regla de tres. Como sabemos que entre A y B hay una distancia y en el eje de las abscisas, y que entre A y la altura del punto hay otra y' y sabemos que siguen la proporción del crecimiento del pendiente, sabiendo la distáncia en las las ordenadas entre A i B a la que llamaremos x podremos saber cuando vale la x' que sigue esa proporción. Para encontrar el valor exacto pues solo debemos sumarle al valor de A.x el de x', expresado de la siguiente manera:

$$x = x_B - x_A$$
$$\text{y} = \text{y}_B - y_A$$
$$\text{y'} = \text{y}_P - y_A$$

$$\frac{y'}{y} = \frac{x'}{x} \Leftrightarrow x' = \frac{xy'}{y}$$
$$\text{x}_P = x_A + x'$$

En el caso que $y' = 0$, asignamos a todas las variables el valor de A, ya que significa que A está donde el punto B.

Para terminar ya nos ponemos a dibujar los píxeles del triangulo de un punto al otro de una altura concreta del triangulo. Eso lo hacemos con la función *linea_triangulo* con un for que va del valor de pin_left.x a pin_right.x y se hace en la función línea triangulo. Ahora entraremos en cómo se consigue poner la foto dentro del triangulo

2.3 Dibujando los Píxeles

Para saber el valor correspondiente a cada u y v de cada punto del triangulo, en la función *calcular_interseccion* se interpolan de la misma manera que las x dado que su incremento es lineal. Pero eso solo nos da el valor del par (u, v) en las dos intersecciones, para eso tenemos que sumar a cada píxel que sumemos la diferencia de los valores de u entre los de x, es decir $inc_u = \frac{\Delta u}{\Delta v}$. Ahora que tenemos todos los valores de la línea, hablemos de la función *dibujar_píxel*.

dibujar_pixel recibe de argumentos el par de coordenadas x e y y el par u y v. Con el par (x,y) ubicamos en la ventana el píxel que debe ser pintado con las funciones de openGL y con el par u y v se llama a la función *color_pixel*. Esta retorna un puntero al buffer donde *cargar-foto.c* ha metido todos los valores de todos los píxeles de la foto.

2.4 Los valores de u i v

Tanto u como v son dos floats entre 0 y 1. Estos representan en qué posición de una matriz se encuntra el píxel que queremos: si u es $1/4$ y $v = 3/4$nos indica que el los colores que corresponden al píxel (x,y) de encuntran en la posición de la foto $(1/4 * 500, 3/4 * 500)$ dado que la ventana es 500x500. Para apuntar exactamente para cualquier valor de u y v usamos la siguiente fórmula:

$$[i, j] = i * dimx + j; i = u * dimx; j = dimy(1 - v)$$

donde i,j representa el píxel de la foto y la expresión sirve para una matriz puesta en fila.

Con esto ya hemos cubierto todo lo más relevante de la aplicación, excepto las dos siguientes excepciones:

2.5 Puntos en mismas coordenadas

Si tenemos que A o C están alineadas tanto en las abscisas como en las ordenadas, el calcular_interseccion no se llama con los parámetros adequados, Para comprovar esto se ha añadido este código antes del ya dicho en *dibujar_triangulo*:

```
  //if A and C are aligned, we draw the pixels between them
  if((Aptr->y==Cptr->y))
  {
    calcular_interseccion(Aptr, Cptr, &pin_left, h);
    calcular_interseccion(Cptr, Aptr, &pin_right, h);

    linea_triangulo(pin_left, pin_right, Aptr->y);
  }
  //in the same way, if A and B are aligned, we draw the pixels between them
  if(Aptr->y==Bptr->y)
  {
    calcular_interseccion(Aptr, Bptr, &pin_left, h);
    calcular_interseccion(Bptr, Aptr, &pin_right, h);

    linea_triangulo(pin_left, pin_right, Aptr->y);
  }
```

Y en *determinar_orden* en el caso que A.y sea igual que C.y, el puntero canviará al más cercano al eje de ordenadas y abscisas (puédase ver con el punto que tiene el menor módulo eucídeo desde el (0,0)) con el siguiente código:

```
if((*Cptrptr)->x == (*Aptrptr)->x)
  {
    if((*Aptrptr)->y > (*Cptrptr)->y)
    {
      aux = *Aptrptr;
```

```
      *Aptrptr = *Cptrptr;
      *Cptrptr = aux;
      printf("A>C\n");
    }
    else
    {
      aux = *Cptrptr;
      *Cptrptr = *Aptrptr;
      *Cptrptr = aux;
      printf("A<C\n");
    }

}
```