eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Gráficos por computador

I. Hidalgo

September 26, 2021

Abstract

Desarrollo de una aplicación que dados tres puntos dibuja un triángulo y mapea en él un triángulo equivalente de una foto elegida en formato .ppm. Para calcular cada punto a dibujar y su respectivo color en la foto se utilizan coordenadas polares. El triangulo que se dibuja no debe corresponder en forma y tamaño al triangulo que extraemos de la foto, es decir se puede aumentar y distorsionar.

1 Objetivos de nuestra aplicación

El objetivo de la aplicación es mapear una foto mediante las herramientas de dibujado que nos da la librería OpenGL de C. Para ello habrá que determinar cual es la área que cubre el triangulo en nuestra pantalla y además cual es su equivalente en la foto para poder extraer el color RGB de cada pixel y copiarlo en nuestro punto del triangulo.

2 Datos recibidos

Para poder determinar el triangulo a dibujar y la foto que debemos mapear recibimos diferentes datos.

2.1 triangles.txt

En primer lugar recibimos un documento de texto que el programa procesará para determinar las coordenadas de los tres puntos que forman el triangulo, tanto en nuestra pantalla como en la foto.

En el fichero cada linea que comienza por la letra 't' representa un triángulo. La letra 't' va seguida de 15 números separados por espacios que son los parámetros que definen el triángulo. Cada punto tiene asociados 5 valores:

- (x,y,z) → coordenadas cartesianas en el espacio, aunque para esta practica no será necesario tener en cuenta la tercera dimensión z ya que únicamente dibujaremos un polígono en dos dimensiones.
- (u,v) → las coordenadas u y v representan la posición de un píxel en la imagen .ppm. Toman valores reales desde 0 hasta 1 dependiendo de la posición del píxel tal y como se muestra en la Figura 1.

Figure 1: Coordenadas u y v

2.2 foto.ppm

Se recibe también una imagen en formato ppm, que leeremos mediante la función load_ppm(). En dicha función cargaremos todos los píxeles de la imagen y apuntaremos al primero con la variable *bufferra*, además se guardarán las dimensiones de la imagen en píxeles en las variables *dimx* y *dimy*.

3 Funciones de teclado

La aplicación será sencilla e intuitiva, solo dispondrá de dos teclas para poder interactuar con ella.

- ENTER. Dado que leemos los triángulos de un fichero podemos cargar varios en una sola ejecución, sin embargo en pantalla únicamente se mostrará el primero que se ha cargado. Para que el programa dibuje el siguiente triangulo especificado en el fichero se deberá pulsar ENTER. En caso de que ya se hayan dibujado todos los triángulos del fichero el programa volverá de nuevo al primer triangulo.
- ESC. Para detener la ejecución del programa el usuario simplemente deberá pulsar ESC.

4 Calculo de coordenadas

Para realizar el calculo de coordenadas de los puntos que forman el triangulo hemos planteado dos opciones principales:

4.1 Puntos de corte mediante Interpolación Lineal

Para dibujar un triángulo mediante computador la opción mas sencilla es hacerlo mediante ristras de lineas horizontales, con lo cual deberíamos comenzar a dibujar desde el punto más alto y disminuir la coordenada de altura en uno hasta llegar al punto más bajo en el plano. Con lo cual, disponiendo de los tres vectores que unen los puntos del triangulo y conociendo siempre la coordenada de la altura solo nos queda una incógnita que se puede hallar mediante interpolación lineal, con la siguiente fórmula.

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} \tag{1}$$

Despejamos la x:

$$x = x_1 + \frac{y - y_1}{y_2 - y_1}(x_2 - x_1) \tag{2}$$

Cabe destacar que deberemos dividir el dibujado del triangulo en dos bucles ya que cuando se pasa por el punto medio uno de los vectores de referencia para calcular las x cambia.

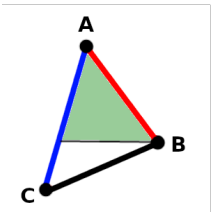

Figure 2:
Vectores de referencia: $\overrightarrow{AB}$ y $\overrightarrow{AC}$

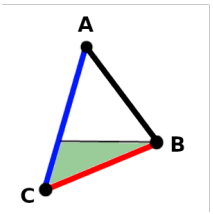

Figure 3:
Vectores de referencia: $\overrightarrow{AC}$ y $\overrightarrow{BC}$

Por lo tanto mediante ese método tendríamos el siguiente código para dibujar un triangulo.

```
static void dibujar_triangulo(hiruki h){
  punto a,b,c;

  a = h.p1; b = h.p2; c = h.p3;
  /*Se ordenan los puntos respecto a y de manera descendente en las
  variables a, b y c*/
  ordenar_puntos(&a,&b,&c);

  int xa = a.x, ya = a.y;
  int xb = b.x, yb = b.y;
  int xc = c.x, yc = c.y;

  dibujar_recta(xa,ya,xa,ya);
  int y1 = ya-1;
  int x1,x2;
  while ( y1 > yb ){

    x1 = xa + (y1-ya) * (xb-xa) / (yb-ya);
    x2 = xa + (y1-ya) * (xc-xa) / (yc-ya);

    if(x1 > x2){
      int aux = x1;
      x1 = x2;
      x2 = aux;
    }
    dibujar_recta(x1,y1,x2,y1);
    y1--;
  }

  while ( y1 > yc ){

    x1 = xb + (y1-yb) * (xb-xc) / (yb-yc);
    x2 = xa + (y1-ya) * (xa-xc) / (ya-yc);

    if(x1 > x2){
      int aux = x1;
      x1 = x2;
      x2 = aux;
    }
    dibujar_recta(x1,y1,x2,y1);
    y1--;
  }

}
```

Aún así el programa final no contiene este método ya que además de dibujar el triangulo era necesario mapear la foto mediante las coordenadas u y v, y el método de la interpolación lineal complicaba más el calculo de dichas coordenadas en el punto interpolado.

4.2 Coordenadas Baricéntricas

El método de las coordenadas baricéntricas es algo menos intuitivo pero es mucho más sencillo de implementar. Para entenderlo es muy sencillo imaginar una linea entre dos puntos cualesquiera. Para conocer un punto cualquiera de esa linea podemos multiplicar un parámetro a cada extremo y sumar los valores para obtener el valor del punto. La suma de los parámetros que se multiplican a los extremos debe ser exactamente 1, cuando tenemos dos extremos $\alpha_2 = 1 - \alpha_1$.

Las coordenadas baricéntricas se pueden expandir hasta n-simplex con n+1 parámetros, siempre que se cumplan las condiciones:

- $\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n \in [0, 1]$
- $\alpha_1 + \alpha_2 + \alpha_3 + \ldots \alpha_n = 1$

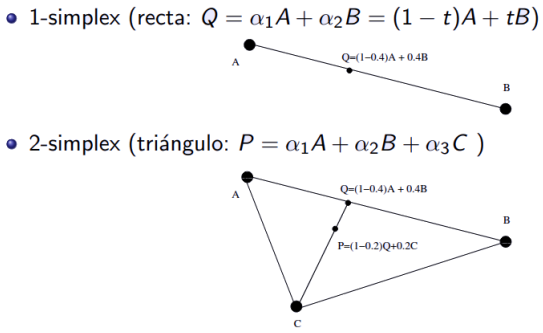

Figure 4: Ejemplo de coordenadas baricéntricas para 1-simplex y 2-simplex

Por lo tanto para hacer el trazado mediante coordenadas baricéntricas aumentaremos los parámetros que multiplican los tres puntos de nuestro triangulo poco a poco para cubrir toda su área interior. Con este método además de poder obtener los valores de x e y de los puntos medios también se pueden obtener los valores de u y de v con la misma formula, lo que facilita el mapeo de la imagen.

Este es el método que dibuja un triangulo con la formula de las coordenadas baricéntricas, en este caso se dibujan punto a punto en lugar de ristras de puntos horizontales.

```
static void dibujar_triangulo(hiruki h){
  punto a,b,c,aux;

  a = h.p1;
  b = h.p2;
  c = h.p3;
  ordenar_puntos(&a,&b,&c);

  int xa = a.x, ya = a.y;
  float ua = a.u, va = a.v;
  int xb = b.x, yb = b.y;
  float ub = b.u, vb = b.v;
  int xc = c.x, yc = c.y;
  float uc = c.u, vc = c.v;

  int x,y;
  float u,v;
  for ( float alfa = 0; alfa <= 1; alfa += 0.001){
    for ( float beta = 0; beta <= 1; beta += 0.001){

      float gamma = 1 - beta - alfa;
      if(gamma >= 0){
        x = alfa * xa + beta * xb + gamma * xc;
        y = alfa * ya + beta * yb + gamma * yc;
        u = alfa * ua + beta * ub + gamma * uc;
        v = alfa * va + beta * vb + gamma * vc;

        dibujar_punto(x,y,u,v);

      }

    }
  }

}
```

5 Mapeado de la foto

Ya sabemos como obtener las coordenadas que definen la posición de un píxel en la foto, tenemos las dimensiones de la foto guardadas en las variables *dimx* y *dimy* y tenemos el puntero *bufferra* apuntando al primer píxel de la foto. Por lo tanto debemos saber en que medida aumentar el puntero para que nos devuelva el píxel que necesitamos en función de las coordenadas u y v y las dimensiones de la imagen.

Para ello debemos analizar la estructura de la imagen. Sabemos que cada fila tiene un numero *dimx* de píxeles, que a su vez tienen un triplete RGB, y las filas simplemente tienen una dimensión igual a *dimy*. También hay que tener en cuenta que v es igual a 1 en la parte superior de la foto y va disminuyendo su valor a medida que bajamos.

Con todos estos datos podemos concluir las siguientes formulas para obtener la fila y la columna del píxel con las coordenadas u_0 y v_0

$$fila = (1 - v_0) \cdot dimy \cdot \underbrace{3u_0 \cdot dimx}_{\text{Longitud de una fila}} \tag{3}$$

$$columna = 3u_0 \cdot dimx \tag{4}$$

Una vez calculados estos valores solo habrá que sumárselos al buffer y devolver el valor obtenido. Implementado en C tendríamos el siguiente código.

```
unsigned char* color_textura(float u, float v)
{
  int fila = dimy * (1-v);
  int columna = dimx * u;
  int total = (fila * dimx  + columna) * 3;
  return(bufferra + total);
}
```

6 Resultados

Tras compilar todos los módulos y ejecutar en el terminal esta es la imagen que aparece.

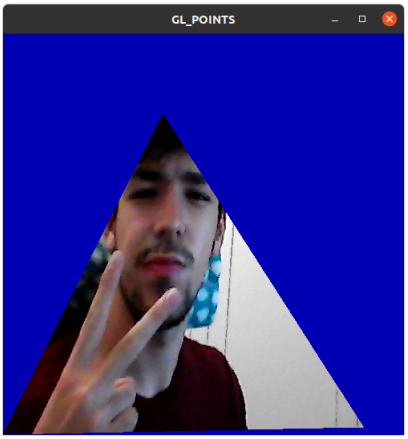

Figure 5: Output del mapeado de la foto sobre el primer triangulo

Si pulsamos ENTER repetidas veces cambiará el triangulo al siguiente que estÉ en el archivo triangles.txt

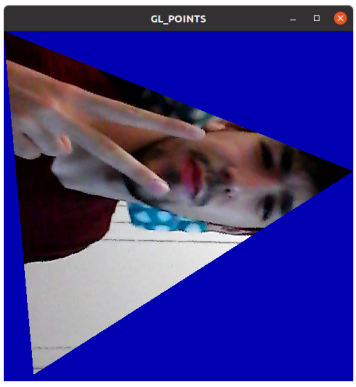

Figure 6:
Triangulo 2

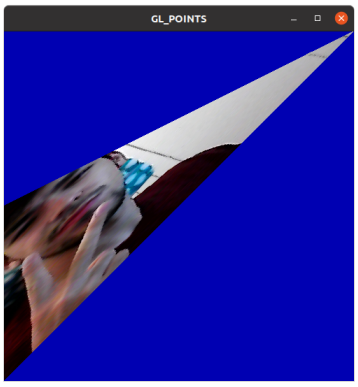

Figure 7:
Triangulo 3

También se puede dibujar y mapear un triangulo que tenga una parte fuera de nuestra pantalla.

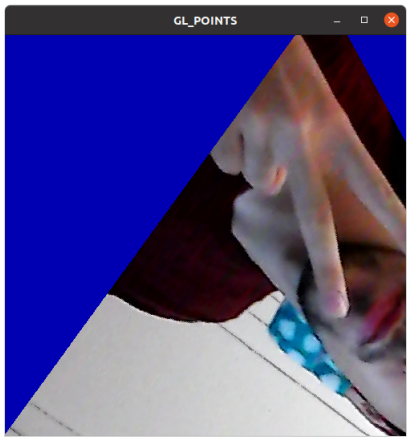

Figure 8: Triangulo con puntos fuera de la vision de la pantalla

7 Conclusiones

El proyecto, a pesar de no ser muy complejo en cuanto a calculo y resultados, es una buena introduccion para conocer las posibilidades y limitaciones del modulo OpenGL de C.

La librería en cuestión es una herramienta muy potente de trazado de polígonos, tanto en 2D como en 3D, aunque en esta práctica no hayamos podido experimentar la segunda alternativa. Mediante el cálculo de coordenadas se pueden dibujar diversos objetos, aunque ese no es nuestro objetivo principal dado que ya existen otros software de modelado mediante polígonos.

Algo más relacionado con nuestro objetivo es el caso del calculo dinámico de coordenadas, que nos permitiría simular una cámara moviéndose por el escenario a medida que los objetos se reposicionan con nuestro movimiento, como veremos en siguientes practicas.