eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Mapeo de textura

Silvia Arenales Muñoz

October 1, 2022

Abstract

Este escrito describe la implementación de una aplicación implementada en lenguaje de programación C y ha sido documentada mediante Latex.

1 Objetivos de nuestra aplicación

El objetivo de esta primera práctica de la asignatura de Gráficos por Computador, sobre el primer tema dado, discretización, es desarrollar una aplicación que mapee una textura sobre un triángulo. Es decir, dados tres puntos, la aplicación dibujará un triángulo con dichos puntos y mapeará en él un triángulo equivalente de una foto del desarrollador, en formato *.ppm*. Para la realización de este se realizará mediante la librería OpenGL de C.

2 Sobre este documento

Este documento explica el desarrollo de la aplicación implementada, el cual es el mapeo de una textura sobre un triángulo. Asimismo, este especifica los problemas surgidos y conocimientos adquiridos a la hora de realizarlo.

3 Interacción con la usuaria

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

1. **ENTER**
   Una vez iniciada nuestra aplicación se mostrará el primer dibujo del triángulo leído de nuestro fichero *triangulos.txt*. Para que el programa dibuje el siguiente triángulo se deberá de pulsar la tecla **ENTER**. En caso de que ya se hayan dibujado todos los triángulos del fichero el programa volverá de nuevo al primer triángulo.

2. **ESC** Para detener la ejecución de nuestro programa, bastaría con pulsar la tecla **ESC**.

4 Ficheros de objetos tridimensionales

Primeramente, explicaré conceptos básicos del trabajo para asegurar una correcta comprensión.

Los triángulos se definen mediante 3 puntos tridimensionales (x, y, z), donde $x, y \in$(0,500)$yz \in (-250, 250)$. A cada punto hay que asignarle las coordenadas de textura que vienen dadas mediante los valores (u, v) donde $u, v \in (0, 1)$.

Además para poder determinar el triangulo a dibujar y la foto que se debe mapear recibimos los dos siguiente elementos.

4.1 triangulos.txt

Disponemos de un fichero **triangles.txt**, en la cual tenemos definidos los triángulos. Estos están definidos de la siguiente manera: cada línea del fichero que comienza con el carácter **t** define un triángulo mediante 15 números, cinco para cada vértice del triángulo; los tres primeros representan las coordenadas (x, y, z), coordenadas espaciales del triángulo, y los otros dos representan las coordenadas de textura (u, v) que definen la parte de la foto que hay que mapear sobre el triángulo.

Listing 1: ejemplo traingulos.txt

```
t 1 1 1 0.1 0.1 200 400 0 0.4 0.8 450 10 0 0.9 0.1
t 0 0 1 0.1 0.1 0 250 0 0.4 0.8 500 500 0 0.9 0.1
t 0 0 1 0.1 0.1 250 0 0 0.4 0.8 500 500 0 0.9 0.1
```

4.2 foto.ppm

La textura a utilizar es una foto en formato **ppm**, que se cargará en la aplicación desde el fichero **foto.ppm**.

Figure 1: Foto utilizada para mapear la textura.

5 Código C

Para la codificación de este programa estamos utilizando la API de gráficos **OpenGL** que especifica una interfaz de software estándar para hardware de procesamiento de gráficos 2D y 3D.

Primeramente si nos fijamos en el main() de nuestro programa, tenemos varias funciones de la librería mencionada que inicializan la ventana.

```
glutInit(&argc,argv);
glutInitDisplayMode ( GLUT_RGB );
glutInitWindowSize ( 500, 500 );
glutInitWindowPosition ( 100, 100 );
glutCreateWindow( "GL_POINTS" );
```

Luego tenemos dos funciones importantes **glutDisplayFunc(marraztu)** y **glutKeyboardFunc(teklatua)**. La primera llama a la función *marraztu* la cual es la encargada dibujar los triángulos y la segunda responde a las peticiones de teclado.

5.1 Función Marraztu

Esta función inicializa la ventana y llama a la función *dibujar_triangulo* la cual irá leyendo del fichero los triángulos y dibujándolas.

```
   dibujar_triangulo(&triangulosptr[indice]);
```

5.1.1 Dibujar_triangulo

Esta función llama a diferentes funciones importantes para dibujar el triángulo correctamente.

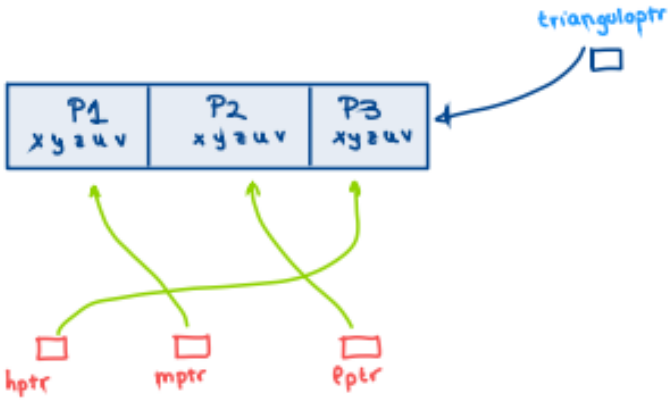
Primero, llama a *establecer_orden_altura*.

Establecer orden altura

Establecer orden altura como dice su nombre ordena en función de la altura los tres puntos que entran como parámetro. Para ello, irá comparando las alturas de los tres puntos; exactamente la coordenada y. Aquel que tenga coordenada más alta se le asignará al puntero *hptr (high)*, el segundo más alto *mptr (medium)* y al último *lptr (low)*.

```
static void establecer_orden_altura(punto *hptrptr, punto *lptrptr, punto *mptrptr)
```

Figure 2: Ilustración de la función establecer orden.



Una vez establecido el orden, si las coordenadas de cada punto son diferentes, se llama a *calcular_corte*

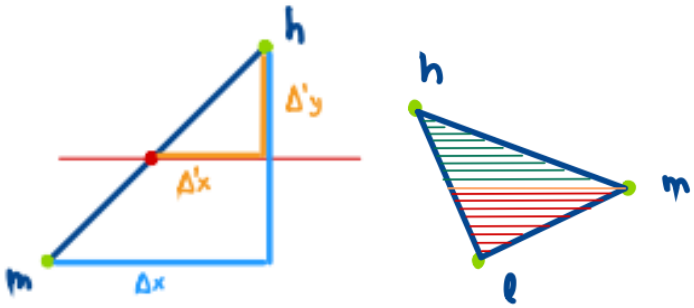
Calcular punto de corte mediante Interpolación Lineal

```
static void calcular_corte(int alt, punto *p1ptr, punto *p2ptr, punto *corteptr)
```

Para dibujar un triángulo realizamos ristas de lineas horizontales, con lo cual comenzamos a dibujando desde el punto más alto y vamos disminuyendo la coordenada de *altura* en uno hasta llegar al punto más bajo del plano.

Con lo cual disponiendo de tres vectores que unen los puntos del triángulo y conociendo siempre la coordenada de altura solo nos queda una incógnita que se puede hallar por interpolación lineal, con la siguiente fórmula.

Figure 3: Interpolación lineal.



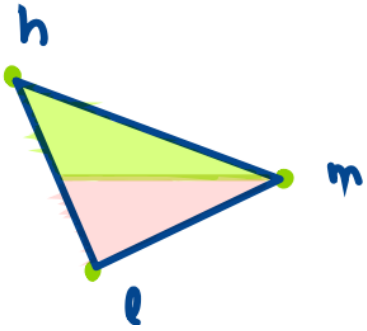
$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} \tag{1}$$

Luego, despejamos la x, el punto de corte:

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

Es de mencionar de que antes de nada debemos dividir el triángulo en dos partes, de la parte más alta hasta la altura del punto medio y desde este último hasta el final.

Figure 4: Ilustración división del triángulo.



Una vez calculado el punto de corte se llama a *dibujar_seccion*.

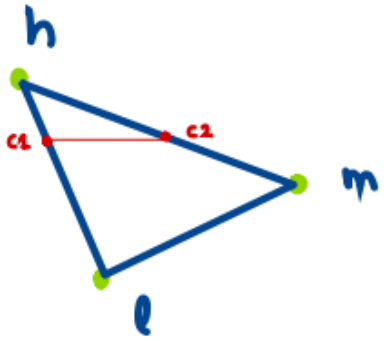
Dibujar seccion

```
static void dibujar_seccion(punto *corte1, punto *corte2,
                            punto *h, punto *m, punto *l)
```

Se trata de una parte fundamental del programa. Se encarga de pintar poco a poco el triángulo.

Como hemos dicho anteriormente, estamos dibujando ristas, por tanto, lo que se encarga esta función es de dibujar cada rista. Para esto es necesario de dos puntos, dos puntos de corte anteriormente calculados (c1 y c2). Iremos pintando desde la coordenada del primer punto desplazándonos hacia la derecha hasta el segundo punto.

Figure 5: Para dibujar la rista es necesario dos puntos.



Para esto llamamos a las siguientes funciones.

```
calcularbaricentro(h,m,l,j,corte2->y,&alfa,&beta,&gama);
calcularUV(h,m,l,&u,&v);
colorv = color_textura(u,v);
```

Calcular coordenadas baricentricas

```
void calcularbaricentro(punto *p1ptr,punto *p2ptr,punto *p3ptr,
                        float i,float j,float *alfa,float *beta,float *gama)
```

Las coordenadas baricéntricas permiten verificar si un punto se encuentra dentro o fuera de un triángulo. Entonces, sea $\tau untriánguloconvértices$p$_0$,$p_1$,$p_2$, y un punto p cualquiera en el plano donde yace τ, $elpuntopuedeserexpresadocomo$:

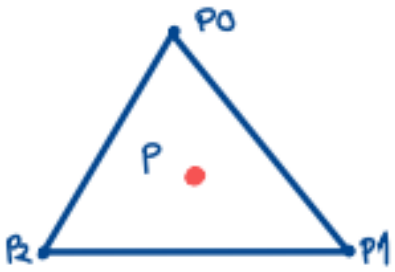
$$p = p_0 + \beta(p_1 - p_0) + \gamma(p_2 - p_0) \tag{3}$$

$$p = (1 - \beta - \gamma)p_0 + \beta p_1 + \gamma p_2 \tag{4}$$

$$p = \alpha p_0 + \beta p_1 + \gamma p_2 \tag{5}$$

$$\alpha + \beta + \gamma = 1; \alpha, \beta, \gamma \in \mathbb{R} \tag{6}$$

Entonces, si $\alpha, \beta, \gamma \in [0, 1]$ entonces p se encuentra dentro del triángulo. Así, es posible *colorear* todos los píxeles que se encuentren dentro de τ.$Aestostresvalores$, $selescon$



Para calcularlas, asumamos que $\tau está formado por p_0 = (x_0, y_0), p_1 = (x_1, y_1) y p2 = (x_2, y_2)$. Entonces, para un punto $p = (x, y)$ sus coordenadas pueden ser calculas como:

$$\alpha = \frac{f_{12}(x,y)}{f_{12}(x_0,y_0)}; \beta = \frac{f_{20}(x,y)}{f_{20}(x_0,y_0}; \gamma = \frac{f_{01}(x,y)}{f_{01}(x_0,y_0} \tag{7}$$

donde podemos agrupar de la siguiente manera:

$$f_a b(x,y) = (y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_a y_a; a, b \in 0, 1, 2 \tag{8}$$

Por lo tanto, para hacer el trazado mediante este método aumentaremos los parámetros que multiplican los tres puntos de nuestro triángulo poco a poco para cubrir toda su área interior. Con este además de poder obtener los valores x e y de los puntos medios también obtenemos los valores u y v con la misma fórmula.

CalcularUV

```
void calcularUV (punto *p1ptr, punto *p2ptr, punto *p3ptr, float *u, float *v)
```

Esta función se encarga de calcular el color de textura teniendo en cuenta las coordenadas de la textura.

Color textura

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

Las variables *dimx* y *dimy* guardan las dimensiones de la foto y el puntero *bufferra* apunta al primer píxel de la foto. Por 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 hay que analizar la estructura de la imagen. Sabemos que cada fila tiene un número de píxeles *dimx*, 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 fórmulas para obtener la fila y la columnas del píxel con coordenadas u_o y v_0.

$$fila = (1 - v_0) * dimy * 3u_0 * dimx \tag{9}$$

$$columna = 3u_0 dimx \tag{10}$$

Una vez calculados estos valores solo habrá que sumárselos al buffer y devolver el valor obtenido.

5.2 Función Teklatua

Las dos opciones para interactuar con nuestra aplicación es tecleando con las teclas **ENTER** y **ESC**, las cuales ya han sido descritas. Ambas funcionalidades están implementadas en esta función. La función es una de las que se nos dieron inicialmente, pero hemos tenido que modificar el caso en el que se pulsa la tecla **ENTER** para que pase al siguiente triángulo de la lista.

```
static void teklatua (unsigned char key, int x, int y){
  switch(key){
    case 13:// <ENTER>
      printf ("ENTER: siguiente tri ngulo.\n");
      if(indice < num_triangles-1){
        indice++;
      }else{
        printf ("Ya hemos leido todo el fichero, empezamos de nuevo.\n");
        indice = 0;
      }
      marraztu;
      break;
```

Como podemos observar, controlamos el índice de la lista de triángulos. Cada vez que pulsemos **ENTER**, el índice aumenta, por lo que pasamos al siguiente triángulo. Si la lista llega al final, volvemos a empezar desde el principio, asignando al número 0 al índice.

6 Resultado

Podemos encontrar cinco diferentes situaciones:

6.1 Caso normal

Este caso es el más común donde se puede apreciar un triángulo bien definido con tres puntos distribuidos sobre el plano.

6.2 Línea horizontal

Los puntos están alineados en el eje y , es decir toman los mismos valores en y, y forman una línea horizontal.

6.3 Línea vertical

Los puntos están alineados en el eje x , es decir toman los mismos valores en x, y forman una línea vertical.

6.4 Puntos superpuestos

Los puntos están alineados en el eje x , es decir toman los mismos valores en x, y forman una línea vertical.

6.5 Línea oblícua

Dos de los puntos están superpuertos y forman con el otro restante una línea oblícua.

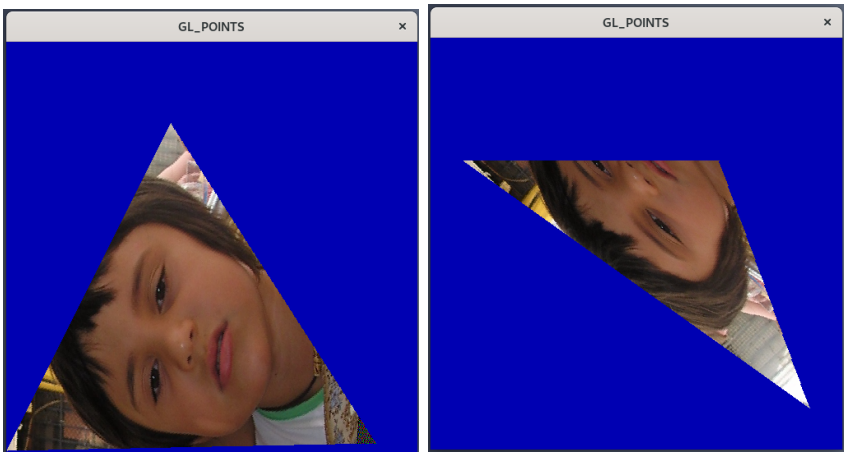


Figure 6: Caso normal

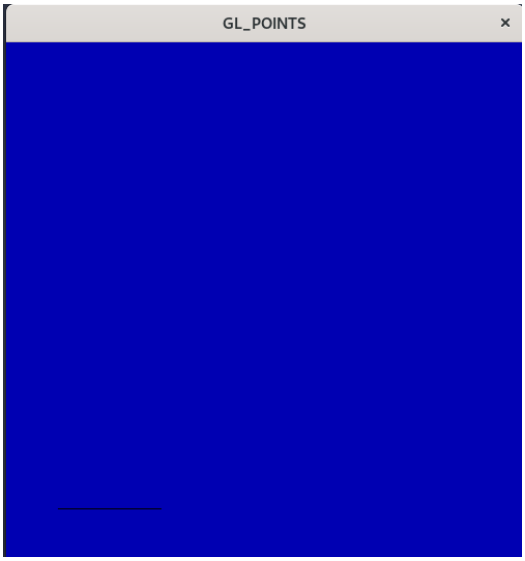


Figure 7: Caso línea horizontal

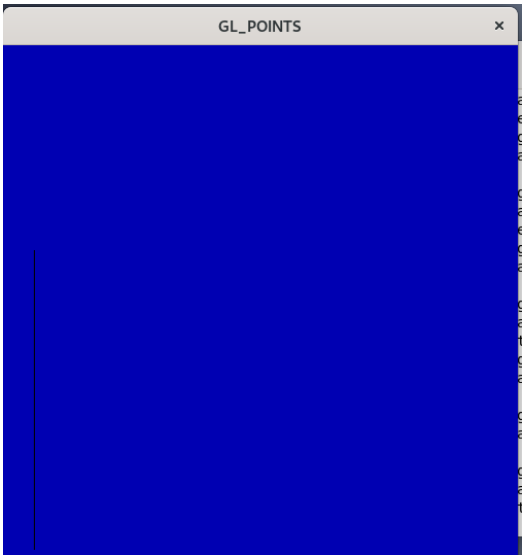


Figure 8: Caso línea horizontal

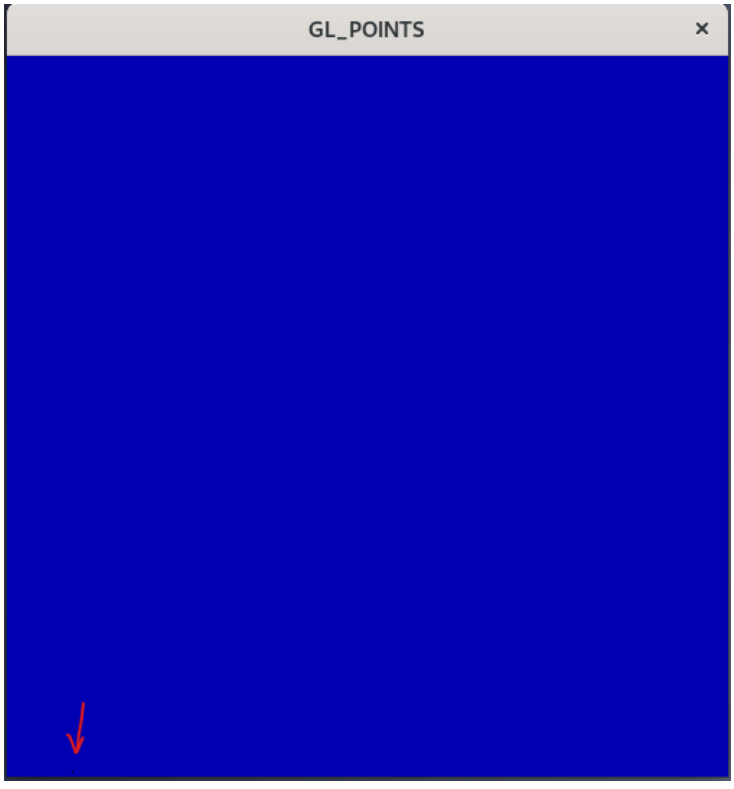


Figure 9: Caso puntos superpuestos

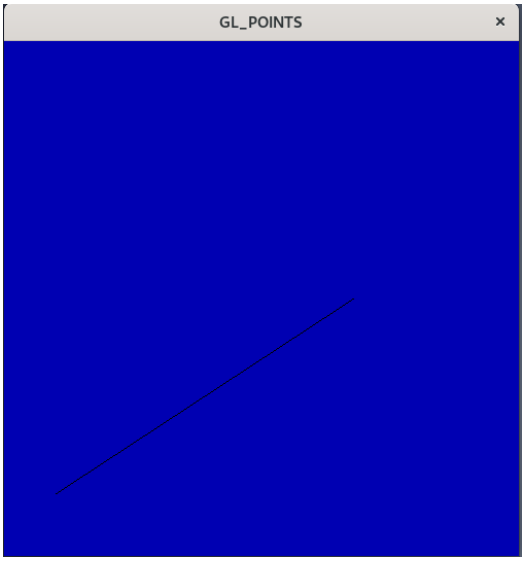


Figure 10: Caso línea oblícua

7 Trabajo futuro

El proyecto, a pesar de no ser muy complejo en cuanto a cálculo y resultados, es una buena introducción para conocer las librerías que ofrece OpelGL de C y así poder experimentar dibujando gráficos en 2D. En un futuro uno de nuestro objetivo podría ser el cálculo dinámico de coordenadas, que nos permitiría simular una cámara moviéndose por el escenario.

References

[1] Transparencias de Egela de la asignatura Gráficos por Computador, *Transparencias-discretización*