eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Mapeo de Textura

Yara Diaz de Cerio & Jon Perez

October 1, 2019

1. Objetivos y herramientas

El objetivo de esta aplicacion es el dibujado de un area triangular de una imagen en un lienzo aparte mediante openGl. Deberemos tener una imagen en formato .ppm, la cual cargaremos usando el fichero "cargar-ppm.c". Los triangulos a dibujar vendran en el fichero "triangulos.txt". Las estructuras que vamos a usar las hemos guardado en el fichero "triangulo.h". Aqui se define lo que es un punto que tiene 3 variables dimensionales y dos de textura que hacen referencia a la imagen. Toda la aplicacion de dibujado esta en el fichero restante "dibujar-puntos.c".

2. Desarrollo y funcionalidades

Lo primero es cargar los triangulos en un puntero " triangulosptr", tambien contaremos cuantos triangulos habia en el fichero y el triangulo que vamos a dibujar.

```
// Indica en que triangulo estamos dibujando.
int indice=0;

// puntero hacia los triangulos cargados
hiruki *triangulosptr;

//numero de triangulos hayados en el .TXT
int num_triangles;
```

Para adaptar la textura de la imagen hemos usado el subprograma color_textura. Este se encargara de devolvernos los valores apropiados segun el tamaño de nuestra ventana y los valores dados.

```
//consigue los datos de la textura adaptados al tamaño de la ventana
unsigned char * color_textura(float u, float v)
{
    int col = u *dimx;
    int fila = dimy -v*dimy;
return(bufferra + (3*dimx*fila +3*col));
}
```

Por otro lado, tenemos el metodo teklatua. Este sera el que preste atencion al teclado esperando ordenes del usuario para saber que hacer. En nuestro caso tendra dos opciones, con la tecla enter pasara al siguiente triangulo (en bucle) y con el escape terminara la aplicacion.

```
// This function will be called whenever the user pushes one key
static void teklatua (unsigned char key, int x, int y)
{
switch(key)
    {
    case 13:
        printf ("ENTER: que hay que dibujar el siguiente triángulo.\n");
        /* hacer algo para que se dibuje el siguiente triangulo */
        /*
        indice ++;  // pero si es el último? hay que controlarlo!
        */
        // printf("%s %d \n", "el indice: ", indice );
        printf("%s %d \n", "num_triangles: ", num_triangles );
        //Avanzar al siguiente triangulo o al primero en caso de que sea el ultimo
            if(indice < num_triangles-1){
                indice++;
            }else{
                indice=0;
            }


        break;
    case 27:  // <ESC>
        exit( 0 );
        break;
    default:
        printf("%d %c\n", key, key );
    }
```

Hemos llegado a la funcion de dibujado, mas que dibujar por si misma, llama a otras funciones para que hagan el trabajo. El trabajo de este subprograma consiste en generar la ventana usando openGl con las dimensiones apropiadas y sacar por pantalla el resultado del dibujo.

```
static void marraztu(void)
{
float u,v;
float i,j;
unsigned char* colorv;
unsigned char r,g,b;

// borramos lo que haya...
glClear( GL_COLOR_BUFFER_BIT );

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 500.0, 0.0, 500.0,-250.0, 250.0);
printf("%s %d \n", "el indice es: ",indice +1);
dibujar_triangulo(triangulosptr + indice);
glFlush();
}
```

Para dibujar el triangulo primero necesitamos ordenar los puntos segun su altura, ya que vamos a dibujar de arriba a abajo. Una vez eso este establecido, nos aseguraremos de que si dos puntos tienen la misma altura no causen ningun problema de division por 0, para ello antes de calcular ninguna variacion nos aseguramos de que si se da el caso la variacion para esos puntos toma el valor predeterminado de 0. Si todos tienen la misma altura nos dira que es una linea y lo sacara por pantalla.

```
if( midPTR->y-supPTR->y == 0 && infPTR->y-supPTR->y== 0 &&  infPTR->y-midPTR->y ==0 ){
     printf("%s\n","No es un triangulo" );
     exit( 0 );

}
else if (midPTR->y-supPTR->y == 0){
    printf("%s\n", "apunta hacia abajo" );
    mSM = 0;
    uSM = 0;
    vSM = 0;


    p1.x = midPTR->x;
    p1.y = midPTR->y;
    p1.z = midPTR->z;
    p1.u = midPTR->u;
    p1.v = midPTR->v;


     mSI = (infPTR->x - supPTR->x) / (infPTR->y-supPTR->y);
     mMI = (infPTR->x - midPTR->x) / (infPTR->y-midPTR->y );


     uSI = (infPTR->u - supPTR->u) / (infPTR->y-supPTR->y);
     uMI = (infPTR->u - midPTR->u) / (infPTR->y-midPTR->y);


     vSI = (infPTR->v - supPTR->v) / (infPTR->y - supPTR->y);
     vMI = (infPTR->v - midPTR->v) / (infPTR->y - midPTR->y);

}
else if (infPTR->y-midPTR->y ==0 )
{
    printf("%s\n", "apunta hacia arriba" );
    mMI = 0;
    uMI = 0;
    vMI = 0;

     mSM = (midPTR->x - supPTR->x) / (midPTR->y-supPTR->y);
     mSI = (infPTR->x - supPTR->x) / (infPTR->y-supPTR->y);


     uSM = (midPTR->u - supPTR->u) / (midPTR->y-supPTR->y);
     uSI = (infPTR->u - supPTR->u) / (infPTR->y-supPTR->y);


     vSM = (midPTR->v - supPTR->v) / (midPTR->y - supPTR->y);
     vSI = (infPTR->v - supPTR->v) / (infPTR->y - supPTR->y);


}
else{

// Variaciones del resto de las variables respecto a la Y
     mSM = (midPTR->x - supPTR->x) / (midPTR->y-supPTR->y);
     mSI = (infPTR->x - supPTR->x) / (infPTR->y-supPTR->y);
     mMI = (infPTR->x - midPTR->x) / (infPTR->y-midPTR->y );

     uSM = (midPTR->u - supPTR->u) / (midPTR->y-supPTR->y);
     uSI = (infPTR->u - supPTR->u) / (infPTR->y-supPTR->y);
     uMI = (infPTR->u - midPTR->u) / (infPTR->y-midPTR->y);

     vSM = (midPTR->v - supPTR->v) / (midPTR->y - supPTR->y);
     vSI = (infPTR->v - supPTR->v) / (infPTR->y - supPTR->y);
     vMI = (infPTR->v - midPTR->v) / (infPTR->y - midPTR->y);
}
```

Una vez tenemos eso ordenado, cabe mencionar que usaremos puntos auxiliares para no modificar ningun valor original.

```
//puntos auxiliares para no modificar los valores originales.
punto p1,p2;
p1.x = supPTR->x;
p1.y = supPTR->y;
p1.z = supPTR->z;
p1.u = supPTR->u;
p1.v = supPTR->v;

p2.x = supPTR->x;
p2.y = supPTR->y;
p2.z = supPTR->z;
p2.u = supPTR->u;
p2.v = supPTR->v;

// Variaciones del resto de las variables respecto a la Y
float mSM;
float mSI;
float mMI ;

float uSM;
float uSI;
float uMI;

float vSM ;
float vSI;
float vMI ;
```

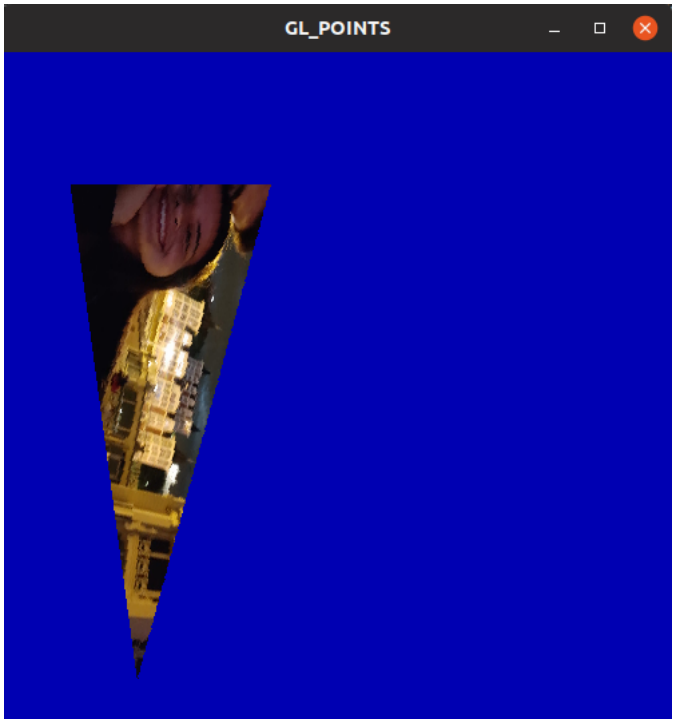
Nuestro patron de dibujado ira desde arriba del todo hasta el punto mas bajo. Para esto, utilizando interpolacion, variaremos los valores de los puntos auxiliares segun sus razones de cambio anteriormente mencionadas. Esto ocurrira de la siguiente manera, bajaremos de pixel en pixel el valor de la "Y", y como esas variaciones estan en funcion de la misma por cada valor que descendamos modificaremos el valor del resto de variables segun su razon de cambio.

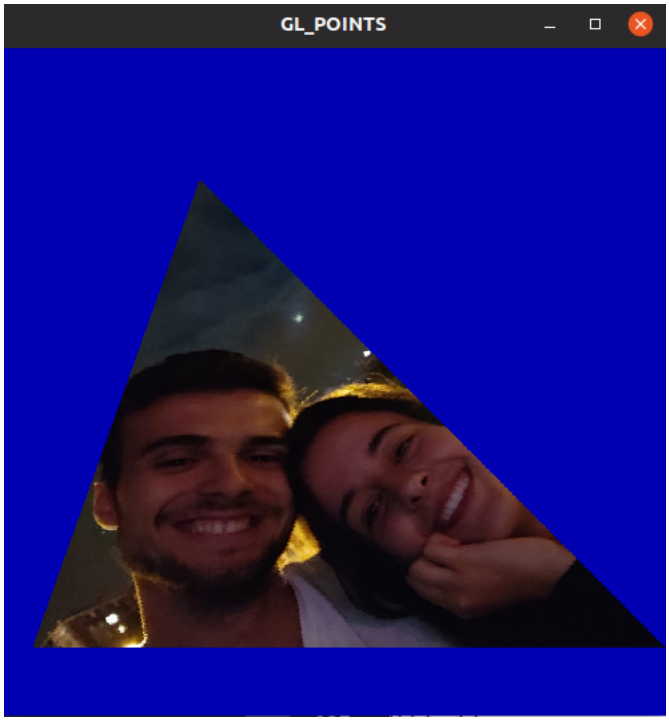
```
//Bucle para movernos desde el punto mas alto hasta el medio
    for(float i = supPTR->y; i > midPTR->y;i-- ){

//Resta de la variacion ya que nos movemos de arriba a abajo

        p1.x = p1.x - mSM;
        p2.x = p2.x - mSI;

        p1.u = p1.u - uSM;
        p2.u = p2.u - uSI;

        p1.v = p1.v - vSM;
        p2.v = p2.v - vSI;


        if(p1.x >= p2.x){
            dibujar_segmento(&p2,&p1,i);
        }else {dibujar_segmento(&p1,&p2,i);}


    }
```

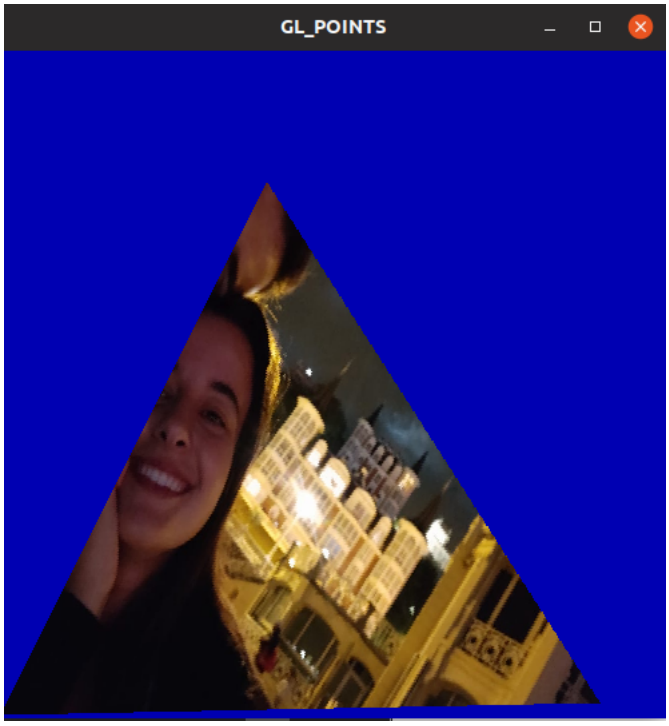
Despues, comprobando cual de los puntos esta mas a la izquierda, se los pasamos a otra funcion que se encargara de dibujar la linea que hay entre esos dos puntos. Al metodo de dibujar segmento le daremos los dos puntos interpolados y la altura a la que se encuentran. Una vez mas, moviendonos de uno en uno en la X y variando las texturas segun las razones de cambio para esa interpolacion concreta, dibujaremos pixel a pixel los puntos intermedios.

```
// Dada una altura y dos puntos, dibujar la linea que los une mediante interpolacion
void dibujar_segmento(punto *pc1PTR, punto *pc2PTR, float alt)
{
    unsigned char* colorv;
    unsigned char r,g,b;

// variables auxiliares para no modificar las originales
    float pU,pV;

    pU =pc1PTR->u;
    pV = pc1PTR->v;

// Variacion respecto de X
    float varU = (pc2PTR->u - pc1PTR->u)/ ( pc2PTR->x - pc1PTR->x);
    float varV = (pc2PTR->v - pc1PTR->v)/ ( pc2PTR->x - pc1PTR->x);

// bucle de interpolacion
    for( float i = pc1PTR->x; i < pc2PTR->x; ++i){



            pU = pU + varU;
            pV = pV + varV;

        //funcion de dibujado.
            colorv = color_textura(pU,pV);
             r= colorv[0];
             g =colorv[1];
             b = colorv[2];
             glBegin( GL_POINTS );
            glColor3ub(r,g,b);
            glVertex3f(i,alt,0.);
            glEnd();


            }
}
```

Estos seran los metodos principales usados para el dibujo de todos los triangulos. A continuacion, dejaremos varios ejemplos de los triangulos que hemos conseguido usando la siguiente imagen.


GL_POINTS


GL_POINTS



3. Manual de Usuario

El manual de Usuario requerido para esta aplicacion es realmente minimo, sin embargo, seran necesarias las librerias de OpenGl y sus dependencias junto a un terminal linux.
Para iniciar la aplicacion bastara con situar el terminal en la carpeta de la aplicacion y escribir " ./ejecutable ", esto inmediatamente dibujara el primer triangulo del fichero.
Una vez llegados a este punto solo habra dos opciones, pulsar enter para pasar al siguiente triangulo o pulsar escape para cerrar la aplicacion.