Reporte Trabajo Final Computacion Grafica
Renderizador CG

Jesus Erick Vera Callme

En el siguiente reporte incluye lo que se hizo para el trabajo del Renderizador de Computación Gráfica.

- Se uso la libreria FLTK.

Se implementó los siguiente algoritmos vistos en clases para el trabajo

- Brensenham para el dibujado de líneas
- Scanline para el relleno de polígonos
- Clipping
- Interpolación

Bresenham:

El algoritmo básico utiliza la ecuación de la recta expresada como

$$y = mx + b$$

Si la recta se dibuja desde el punto ($x0$, $y0$) hasta el punto ($x1$, $y1$), el algoritmo varía x desde $x0$ hasta $x1$ en incrementos de una unidad. Bresenham hace calculos con numeros enteros evitando los cálculos flotantes.
Este algoritmo trabaja con 2 puntos p1, p2. Calcula diferenciales de x e y.

Este algoritmo recibe 2 vectores de puntos, un **color_buffer**, un w, y h que son las coordenadas de viewport.

Luego la función calculará las diferenciales del punto inicial __ ppvv1 y __ppvv2

```
int __dx = fabs(__ppvv1[0] - __ppvv2[0]);
int __dy = fabs(__ppvv1[1] - __ppvv2[1]);
```

Lo anterior lo hace para llamar a una función que calcule cuando la diferencial de x es mayor o menor que la diferencial de y.

```
if (__dx >= __dy)
else if (__dy > __dx)
```

Si la diferencial de x _ dx es mayor que la diferencial de y __dy, entonces la acumulacion se dará.
Lo anterior solo inicializa __i, ___j y __k,

- i es igual al doble de la diferencial de y menos la diferencial de x
- j es el doble de la diferencial de y, o el incremento incE
- k es el doble de la diferencia de las diferenciales x e y o el incremento incNE.

un swap si el x del punto inicial es menor que el punto final.
Si sucede así, entonces procederé a dibujar el primer punto inicial.

```
draw_point(__ppvv1, __w, __h, __color_buffer);
```

Lo siguiente que hare sera ir incrementando en una unidad desde el punto mínimo al punto casi final.

Preguntare mientras el x del primer punto sea menor al final, hago el incremento,
si es que __i es negativo entonces, el incremento de __i será __j (2*dy)
si no sucede así, __i es mayor que cero, y compruebo que el punto inicial en y es menor que el punto final, sino es el caso entonces, disminuyó al punto inicial en y.
Luego incremento __i en k (2*(dx-dy)). Y después de calcular el punto a dibujar, lo mando a draw_point.

El caso contrario sucede cuando la diferencial de y es mayor que la diferencial de x.
Para continuar inicializar los valores de los incrementos:

```
__i = 2 * __dx - __dy;
__j = 2 * __dx;          // incE
__k = 2 * (__dx - __dy);         //incNE
```

Hare lo mismo que el paso anterior, pero preguntando si la y del punto inicial es menor que la del punto final. Si sucede eso hago un swap de puntos.
Una vez pasó lo anterior, dibujo el punto draw point.

Luego iterar mientras que el y inicial sea menor que el y final.

```
while (__ppvv1[1] < __ppvv2[1]) {
        if (__i < 0)
           __i += __j;
        else {
           if (__ppvv1[0] > __ppvv2[0])
              --__ppvv1[0];
           else
              ++__ppvv1[0];
           __i += __k;
        }
        ++__ppvv1[1];
        interpolacion_triangle(__ppvv1[0], __ppvv1[1]);
        draw_point(__ppvv1, __w, __h, __color_buffer);
     }
```

En la iteración preguntare si el incremento de __i es negativo, si sucede así este se incrementa en __j

Relleno de polígonos (Scanline)

El algoritmo Scanline trabaja utilizando 2 tablas una ActiveEdgeTable y EdgeTable.
La segunda estructura se usa para dependiendo a su coordenada y identificar que edge nace desde la coordenada y.
Una vez construida EdgeTable, se procede con la ActiveEdgeTable, esta tabla almacenará toda la iteración y operación que se realiza en cada edge.

En nuestra implementación hacemos uso de 3 funciones

```
int __number_of_vertex=3;
set_points(__pv1, __pv2, __pv3, __number_of_vertex);
find_max_min(); start_fill_table_x_polygon(__w,__h,__color_buffer);
start_fill_table_x_polygon(__w,__h,__color_buffer);
```

Primero indicamos de que figura será, como se está trabajando con triángulo vertex=3
Luego se llama a la función set_points que es la que inicializará todo Scanline,
Luego llamamos a find_max_min, dicha función primero empieza determinando x e y maximos y minimos.

```
   for(int __i = 0; __i < __v; __i++){
      if(__x_min > int(__points_to_fill[__i].__p_x_y[0]) )
         __x_min = int(__points_to_fill[__i].__p_x_y[0]);
      if(__x_max < int(__points_to_fill[__i].__p_x_y[0]) )
         __x_max = int(__points_to_fill[__i].__p_x_y[0]);
      if(__y_min > int(__points_to_fill[__i].__p_x_y[1]) )
         __y_min = int(__points_to_fill[__i].__p_x_y[1]);
      if(__y_max < int(__points_to_fill[__i].__p_x_y[1]) )
         __y_max = int(__points_to_fill[__i].__p_x_y[1]);
}
```

Para continuar se sigue con star_fill_table_x_polygon, esta función iterara sobre toda la estructura mientras que el valor min (y_min+0.00001) sea menor que __y_max.

```
   float __min_value;
   __min_value = __y_min + 0.0001;

   while(__min_value <= __y_max){
      line_process_fill_table_x(__min_value);
      fill_polygon_table_x(__min_value,__w, __h, __color_buffer);
      __min_value++;
   }
```

Esta funcion, llama a line_process_fill_table_x

```
   for (int __i = 0; __i < 6; __i++){ // setting array
```

```
    __table_x[__i]=0;
  }
  int __x1, __x2, __y1, __y2, __temp;
  __counter = 0;
  for(int __i = 0; __i < __v; __i++){
    __x1=int(__points_to_fill[__i].__p_x_y[0]);
    __y1=int(__points_to_fill[__i].__p_x_y[1]);
    __x2=int(__points_to_fill[__i+1].__p_x_y[0]);
    __y2=int(__points_to_fill[__i+1].__p_x_y[1]);
    if(__y2 < __y1){ // swap
      __temp = __x1;
      __x1 = __x2;
      __x2 = __temp;
      __temp = __y1;
      __y1 = __y2;
      __y2 = __temp;
    }
    if(__z <= __y2 && __z >= __y1){ //z between y max, and y min
      if( (__y1 - __y2) == 0 ) // horizontal line
        __x = __x1;
      else { // se utiliza para realizar cambios en x de modo que podemos llenar nuestro polígono después de cierta distancia
        __x = ( (__x2 - __x1) * (__z - __y1) ) / (__y2 - __y1);   // setting x
        __x = __x + __x1; //
      }
      if(__x <= __x_max && __x >= __x_min) //
        __table_x[__counter++]=__x;
    }
  }
```

La anterior función lo que hará es iterar por todos los vértices, en donde inicia el FOR, dentro del bucle lo que evaluaremos es si estos puntos son mayores o menores que sus siguientes, si es asi se hace un swap.
Luego preguntaremos si el valor mínimo Z está entre nuestros 'y's , si sucede así preguntaremos como siguiente paso si se trata de una línea horizontal en cuyo caso se igualan las x. Sino sucede así, incrementamos a x con

x = x+ dx / dy

Luego preguntaremos si x está entre min y max, si es asi, llenamos la tabla de lo que pintaremos con el valor actual de x.

La otra función que se manda a llamar es

```
  int __i;
  for(__i = 0; __i < __counter; __i += 2)       {
    vector<2> __p;
    __p[0] = __table_x[__i];      // x1
    __p[1]=z;

    vector<2> __p2;
    __p2[0] = __table_x[__i+1];   // x2
    __p2[1]=z;          // y2

    bresenham_algorithm(__p, __p2, __w, __h, __color_buffer);
}
```

El anterior algoritmo solo iterara por toda la tabla de puntos de x e y, y para poder pintar una línea horizontal.

Clipping

Este algoritmo me dirá si un punto dibujado está dentro del viewport que he considerado evaluar.

Hay un enfoque conocido que es el Cohen-Sutherland, este enfoque hace uso de valores binarios o TBLR para determinar si un punto está tanto Arriba, Abajo, Izquierda o Derecha.

- T = Top = 1000 = 8
- B = Bottom = 0100 = 4
- L = Left = 0010 = 2
- R = Right = 0001 = 1

Si el punto está dentro del viewport, entonces su binario es 0000.

Las operaciones se realizan con un operador OR que me daran el codigo que indica si el punto está en top, bottom, right, o left

En el algoritmo procedemos primero a calcular los tblr codes, usando el algoritmo descrito anteriormente.
Necesitaremos una pendiendo, por lo cual la calculamos, dicha pendiendo nos servirá para determinar nuestros nuevos puntos x e y si fuera necesario en nuestro viewport.
Haremos un while seguidamente para calcular hasta dónde es necesario iterar desde el punto inicial.

Dentro del bucle preguntaremos

- si el código que obtenemos está dentro
- si lo anterior no sucede verificamos si esta en top
    - si esta en top se calcula la intersección
        - y=ymax;
        - x=xi+ 1.0/m*(ymax-yi);
- si lo anterior no sucede verificamos si está en bottom
    - si está en bottom se calcula la intersección
        - y=ymin;
        - x=xi+1.0/m*(ymin-yi);
- si lo anterior no sucede verificamos si está en right
    - si está en right se calcula la intersección
        - x=xmax;
        - y=yi+m*(xmax-xi);
- si lo anterior no sucede verificamos si está en left
    - si esta en left se calcula la intersección
        - x=xmin;
        - y=yi+m*(xmin-xi);

Luego preguntaremos

```
if(c==c1){ // clipping
   xd1=x;
   yd1=y;
   c1=evaluate_TBLR(xd1,yd1);
}
if(c==c2){
   xd2=x;
   yd2=y;
   c2=evaluate_TBLR(xd2,yd2);
}
```

Si el código es igual al primero o segundo codigo o c2, si esto sucede así indica que todavía no está dentro, entonces volveremos a calcular el código.

Una vez lo anterior este hecho, retornamos las verdaderas coordenadas de acorde al viewport.

Interpolación

Para la interpolación de colores, usaremos coordenadas baricéntricas, que consisten en teniendo los colores de un triángulo (tres vértices con posiciones y color), haremos una operación entre los vértices para obtener una proporción de cuanto de color de RGB debe ir en cada punto dentro del triángulo.

La forma en como se calculó la coordenada baricéntrica fue determinar:

- El área general de los tres puntos del triángulo

- Luego desde el nuevo punto que se busca a evaluar hacia los otros vértices
- Del punto hacia los otros vértices calcular el area (seran 3 áreas menores)

Hecho lo anterior podemos ahora hallar constantes alfa, beta, y gamma que nos indicarán la proporción de cada color en cada vértice.

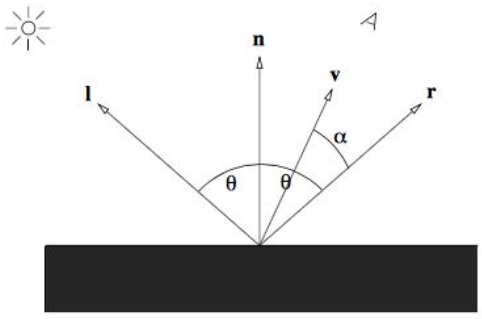
- Alfa se calcula dividiendo el área pequeña 1 que forma un triángulo pequeño sobre el gran triángulo.
- Beta se calcula dividiendo el área pequeña 2 que forma un triángulo pequeño sobre el gran triángulo.
- Gamma se calcula dividiendo el área pequeña 3 que forma un triángulo pequeño sobre el gran triángulo.

```
alfa=
        area_triangle_3_points(x,y, __pv_2_cb, __pv_3_cb) /
        area_triangle_3_points(__pv_1_cb[0], __pv_1_cb[1], __pv_2_cb, __pv_3_cb);
beta=
        area_triangle_3_points(x,y, __pv_3_cb, __pv_1_cb) /
         area_triangle_3_points(__pv_2_cb[0], __pv_2_cb[1], __pv_3_cb, __pv_1_cb);
gamma=
        1.0-alfa-beta;
```

Luego para darle el color se setea en _material_color

Iluminacion:

Para la iluminacion se utilizo el enfoque de Iluminacion de Phong visto en clase:



- Normal vector: vector que es perpendicular a superficie y dirigido hacia afuera de la superficie
- View vector: vector que apunta hacia la direccion del espectador.
- Light vector: vector que hacia la fuente de luz
- Reflection vector: vector que indica la direccion de la refleccion del vector ligth.

Luego se procede a hacer el calculos de los vectores.

- Del vector normal calculando (normales por cruz):
    - $n = (P1 - P0) \times (P2 - P0)$
- Luego el vector de refleccion (producto punto vector n por l)
    - $r = 2 (n * l) n - l$
- l es la obtencion de un vector (light - pos)
- v es la obtencion de un vector(ligth - cam)

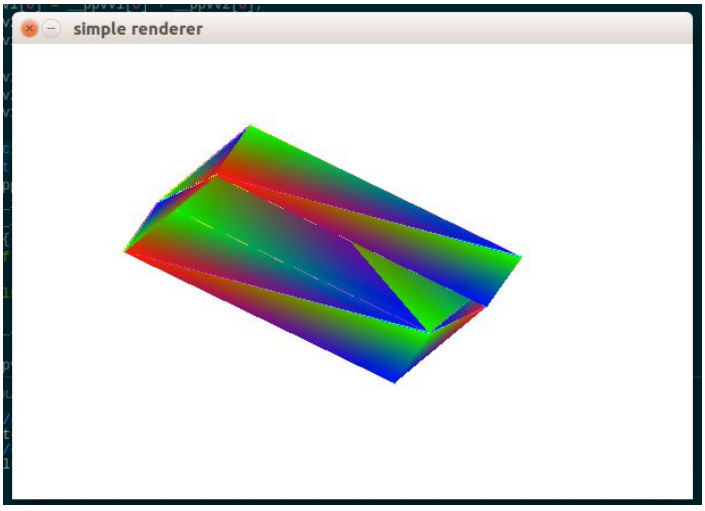
Calculos de componentes:

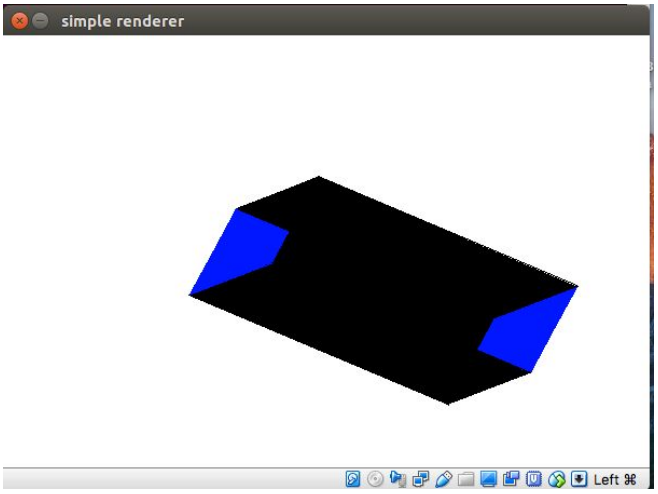
- Ia = ka*La
- Id = kd* max(0, n · l)*Ld producto punto
- Is = ks(r · v)Ls

Luego se calcula la intensidad total sumando.
Después del cálculo general se hace la asignacion a material_color del renderizador

Ejecución.

-Bresenham, Clipping, ScanLine, Interpolación en acción





Reconocedor de .obj

Se hizo el reconocimento del objeto .obj wavefront, usando una libreria del ogl.

La idea es:

- Cargar el .obj en un vector de glm de tipo 3
  - vector de vertices
  - vector de normales
  - vector de texturas
- Luego hacer el parseo de vertices hacia un .scn
- Para luego dibujar el .obj

Problemas:

1. Se detectaron algunos problemas al momento de ejecutar algunos algoritmos.
2. Primero el problema
   a. X_ChangeProperty: BadValue (integer parameter out of range for operation) 0x0
   b. Que se soluciono en engine.cpp cambiando
      i. __main_win.show();
      ii. en vez de __main_win.show(0,0);

3. Luego hubo un problema con el clipping, porque este al ser solo para lineas, no se abordó el tema de polígonos tantos convexos como cóncavos, por eso cuando se acerca a los bordes se distorsiona algunos.

4. Se identificó otro problema y es cuando la imagen se mueve, esta tiene a rotarse, por tanto cuando se hace clipping, no es factible calcular bordes.