Scan Conversion

Tarea 03

Computación Gráfica

UNAM 2022-2

Gibran Zazueta Cruz

03/marzo/2022

1. Introducción

Se programa el algoritmo de scan conversion para rellenar las caras de un cuboide. El objeto y estructura de la escena es la misma que se utilizó para la tarea 2.

1.1. Scan conversion

Scan Conversion es el proceso de convertir una figura geométrica o primitiva representada en el plano del dispositivo en un set de pixeles que describen su geometría. Esto incluye la representación de puntos, líneas, áres y figuras.

El nombre viene de que se 'escanea' la pantalla de arriba hacia abajo y de izquierda a derecha, para obtener la información de la primitiva.

1.1.1. Rellenado de polígono

Para lograr rellenar un polígono primero se almacena la información de sus aristas en un *Buffer*.

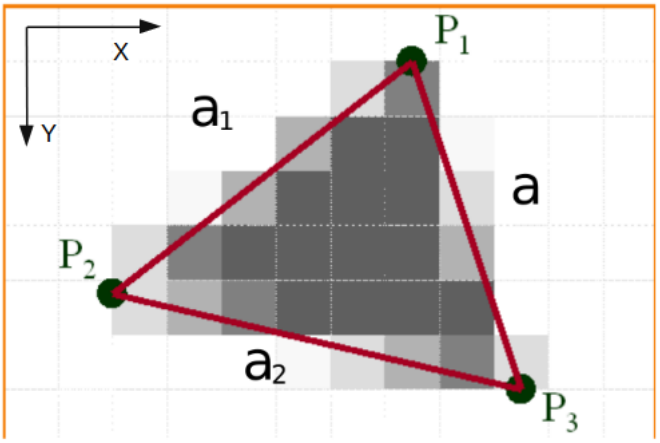

Figura 1: Algortimo rellenado de poligonos con Scan Conversion (imagen obtenida de [2])

Se siguen las aristas de la figura, una por una. Se calcula la pendiente de la recta y se va avanzando una unidad (de pixel) por y, calculando el valor correspondiente para x. Este proceso se debe realizar en orden, es decir, las aristas se siguen en orden horario o antihorario.

Tomando como ejemplo la figura 2, un procesamiento antihorario sería: $a_1 \rightarrow a_2 \rightarrow a_3$.

La idea es dividir las aristas del polígono entre un 'lado izquierdo' y un 'lado derecho' que permita colorear los pixeles desde un borde hasta el otro, por cada coordenada y. Con un procesamiento de este tipo y para un polígono convexo el buffer almacenará 2 coordendas de x por cada coordenada de y.

La manera en que se clasifican los bordes, entra parte derecha e izquierda, es tomando en cuenta el valor menor entre los dos vértices que forman la arista (por ejemplo P_{1_y} y P_{2_y} para la arista a_1).

Para un recorrido antihorario, mientras el valor del primer punto de la arista sea menor al del segundo se considera que esta pertenece al lado izquierdo. Cuando el valor del segundo punto de la arista sea menor al primero, la recta pertenece al lado derecho.

La información de las coordendas x (derecha e izquierda) correspondientes a cada coordenada y se almacena en el buffer.

Una vez se tiene la información del buffer, se recorre la figura de arriba hacia abajo (tomando en cuenta el eje de coordenadas del dispostivo, esto corresponde a un incremento en y), donde por cada valor de y se colorean los pixeles desde la coordenada x derecha hasta la x izquierda.

Esta implementación´ está basada en la información presentada en [3]

1.1.2. DDA

Se utilizó el algortimo de Digital Differential analyzer para dibujar las lineas de las aristas del cubo. Este algoritmo necesita dos puntos (vértices) como entrada. Calcula los incrementos ΔX, ΔY . Finalmente su avanzan iterativamente las coordendas en incrementos Δ, desde un punto inicial a un punto final, seleccionando en cada incremento el pixel a dibujar.

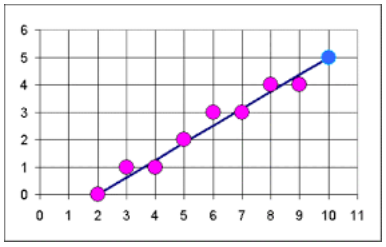

Figura 2: Algortimo DDA (imagen obtenida de [1])

2. Estructura del código

El programa tiene como base el código de la tarea 2.

En la clase CamProjection se programa el método de scan conversion para rellenar el poligono.

Después de proyectar y de pasar las coordenadas al plano del dispositivo, la función *fillCubeFace()* recibe las coordenadas de cada una de las 6 caras del cubo (representadas por 4 vértices). Esta función llama a *scanFillPoly()* que realiza el scan conversion, siguiendo cada una de las aristas y llenando el *Buffer*, representado por un arreglo multidimensional.

Una vez completado el *Buffer* se utiliza la información de este para ingresar los puntos a dibujar a la lista rasterPoints. Esta QList es la que finalmente se dibuja sobre el canvas.

En renderwindow se realizó una implementación de DDA en la función *lideDDA()*

3. Ejecutar el programa

En la carpeta de build se puede ejecutar el programa con el archivo ScanConversion-Run. Desde la consola de comandos de linux:

bash

```
./ScanConversion-Run
```

En la carpeta principal está el código fuente. Para generar el ejecutable primero se genera el Makefile con

bash

```
qmake ScanConversion.pro
```

Después se construye el proyecto con *make*

4. Instrucciones de uso

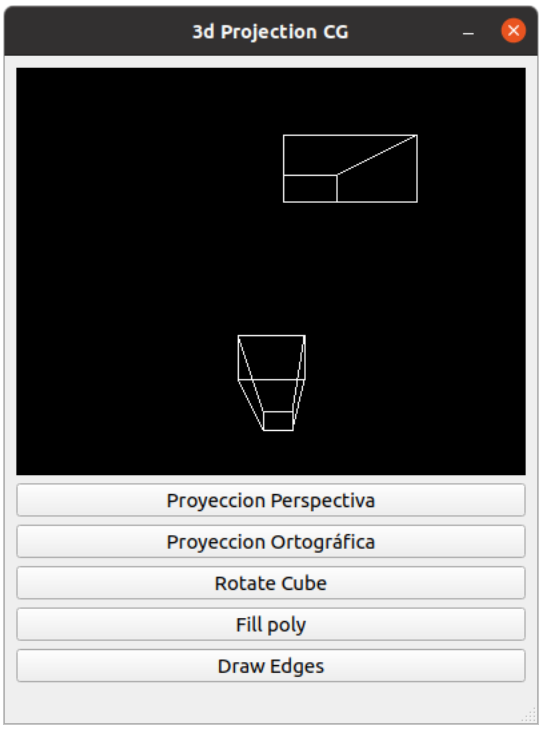

Figura 3: Interfaz gráfica del programa

Se tienen los botones *Fill poly* para rellenar las caras del cuboide y *Draw edges* para dibujar sus aristas

El programa inicia sólo dibujando bordes. Al presionar *Fill poly* se rellenarán las caras del polígono, al presionar una segunda vez se borrarán las caras. *Draw Edges* funciona con un comportamiento similar

El boton *Rotate cube* hace rotar el cubo sobre su eje x, de manera constante. Los botones perspectiva y ortografica cambian a la proyeccion correspondiente.

5. Programa en ejecución

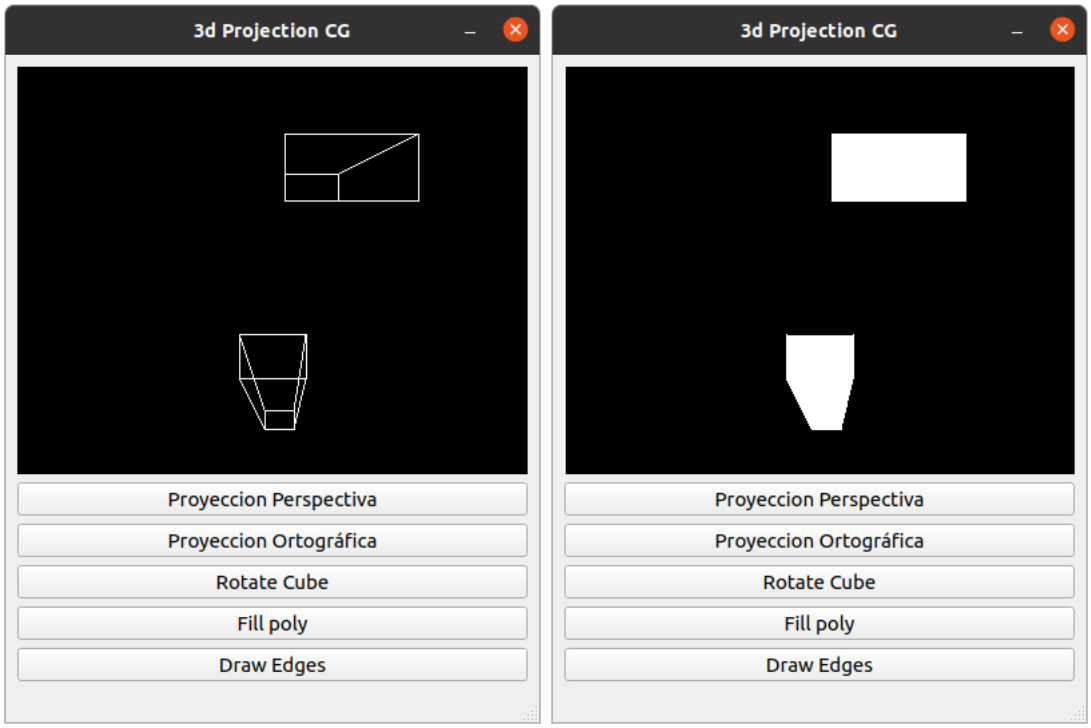

Figura 4: Cuboide con bordes y con rellenado. Proyectado en perspectiva

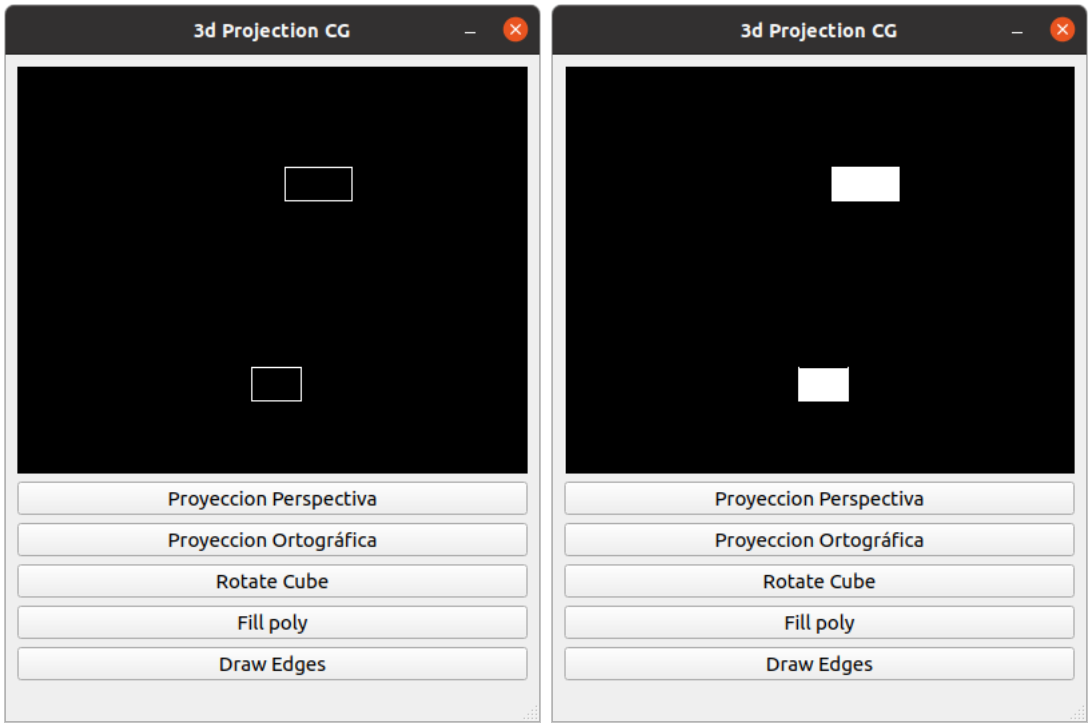

Figura 5: Cuboide con bordes y con rellenado. Proyección Ortográfica

Referencias

[1] Marchese. Image creation and Graphic primitives (*https* : *//csis.pace.edu/ marchese/CG*$_R$*ev/L*4*/cg*$_l$4*.htm*)

[2] Funkhouser. 2D rendering Pipeline (*https* : *//www.cs.princeton.edu/courses/ archive/fall*99*/cs*426*/lectures/pipeline*)

[3] Upssala Universit. Introduction to polygons. (*http* : *//www.it.uu.se/edu/course/homepage /grafik*1*/ht*06*/Lectures/L*02*/LinePolygon/x*$_p$*olyd.htm*)