German Telmo Eizaguirre Suarez

Doble Grado en Biotecnología e Ingeniería Informática

23-10-2019

Ejercicio 2. Rellenar un polígono

Modelización y Visualización, URV (Tarragona)

Introducción

Denominamos rasterización de áreas a rellenar los píxeles del interior de un polígono mostrado por pantalla. Existen dos tipos principales de algoritmos para este propósito.

Los algoritmos de barrido recorren el polígono de su cota superior a la inferior mediante rectas horizontales. Dibujan aquellos fragmentos de cada recta que se incluyan en el interior del polígono. Los algoritmos de barrido son comúnmente conocidos como *Scanline algorithms*.

Los algoritmos de inundación toman un píxel inicial en el interior del polígono. A partir de dicho píxel, pintan todos los píxeles conectados por cierta ruta continua al píxel inicial e incluidos en el interior del polígono. Los algoritmos de inundación son comúnmente conocidos como *Flood Fill algorithms.*

Para esta práctica hemos implementado el algoritmo Sirve como continuación de la práctica anterior de la asignatura, en la que implementamos la rasterización de línea rectas por Needleman-Wunsch.

Decisiones de diseño

Hemos implementado un algoritmo que dibuja el contorno de un polígono y un algoritmo de relleno Scan-Line que cubre los diferentes tipos de polígonos propuestos en el enunciado. El algoritmo de contornos básicamente dibuja una recta entre cada uno de los dos puntos que recibe como argumento. El último punto lo une con el primero. A grandes rasgos, el algoritmo de relleno sigue el siguiente procedimiento.

1. Define el máximo absoluto y el mínimo absoluto de las coordenadas y del polígono.
2. Recorre con líneas horizontales el polígono de su máximo absoluto a su mínimo absoluto.
3. Compara cada línea horizontal con todas las rectas que forman el área del polígono y establece los puntos de corte mediante la fórmula.

   $$x_{corte} = x_1 + (\partial x / \partial y) \times (x_{linea} - y_1)$$

4. Ordena todos los puntos de corte de cada línea horizontal por valor creciente de la x y por cada pareja de puntos pinta los píxeles en medio de ambos puntos. Por ejemplo, si los puntos son {u, v, w, z}, pinta las rectas [u, v] y [w, z].

Aunque la anterior es la visión general del algoritmo que sirve para rellenar polígonos básicos, existen polígonos con topologías más complejas que requieren refinamientos en el código.

- Controlamos los bordes horizontales del polígono, de modo que si alguna línea de barrido coincide con un borde horizontal lo ignora. De otro modo detectaría todos los puntos del borde como puntos de corte.
- Para polígonos cóncavos, controlamos los vértices en el centro de la concavidad, donde contactan dos bordes del polígono. Son puntos críticos ya que su tratamiento es diferente dependiendo de si la concavidad es horizontal o vertical. En el primer caso, el vértice debe interpretarse como dos puntos de corte, ya que se pintarán fragmentos de línea a su izquierda y derecha. En el segundo caso, el vértice debe interpretarse como un solo punto de corte, ya que solo se pintan fragmentos de línea a su costado interior. Este hecho se ejemplifica en la figura 1.

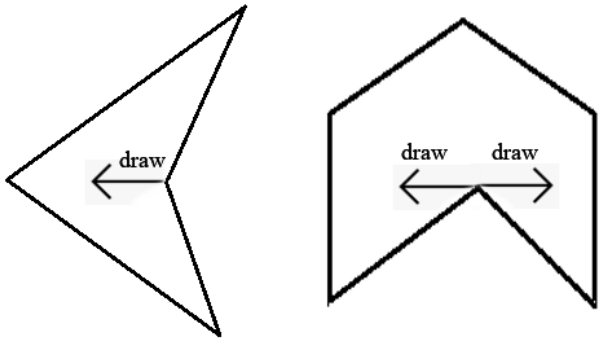

Figura 1: Tratamiento de vértices cóncavos.

Para abordar este problema, medimos la diferencia de las ys de los bordes que conforman el vértice cóncavo desde la x del vértice. Si la diferencia de las ys va en el mismo sentido, añadimos dos veces el vértice (concavidad vertical). Si no, solo se añade en su primera aparición (concavidad horizontal).

- Hemos modificado levemente el algoritmo de contorneado de polígonos para que pueda dibujar polígonos con agujeros. Para ello hay que expresar los puntos que conforman el polígono con agujeros en el formato explicado a continuación.

  Supongamos que el polígono esta formado por un aro externo compuesto por los vértices{E1, …, En}. El polígono interno lo forman los vértices {I1, …, In}. Para contornear un polígono con agujeros el algoritmo deberá recibir la totalidad de puntos en el formato (E1, E2, ..., En, E1, I1, I2, ..., In, I1). Del mismo modo, para dibujar varios

agujeros, (E1, E2, ..., En, E1, I1, I2, ..., In, I1, E1, O1, O2, …, On). El algoritmo dibuja las rectas de manera normal y genera un “pasillo virtual” no dibujado hacia el primer nodo del agujero. Para no pintar las líneas del pasillo se guarda un registro de los puntos de origen de cada una de las líneas, y no se dibujan aquellas cuyo punto de origen ya está dibujado. El “pasillo virtual” generará dos puntos de corte en el mismo lugar y por lo tanto será rellenado a ambos lados, pasillo incluido.

Juego de pruebas.

En el juego de pruebas hemos comprobado la correcta representación de todos los polígonos definidos en el enunciado y en nuestra descripción del diseño.

Caso	**Color**
Rectángulo. Líneas horizontales y verticales.	Azul claro. <0.674 , 0.905, 0.862>
Triángulo.	Verde. <0.482, 0.886, 0.094>
Polígono convexo sin agujeros, 7 lados.	Amarillo. <0.95, 0.95, 0.08>
Polígono con concavidad horizontal.	Naranja. <1.0, 0.2, 0.0>
Polígono con concavidad vertical.	Azul marino. <1.0, 0.5, 0.0>
Triángulo con agujero.	Índigo. <0.5, 0.0, 1.0>
Polígono con dos agujeros.	Marrón. <0.6, 0.2, 0.0>
Polígono de ejemplo.	Rojo. <1.0, 0.0, 0.0>