Rellenado de rectángulos y polígonos

Dr. Luis Gerardo de la Fraga

8 de septiembre de 2009

Resumen

Esta es una traducción parcial de las secciones 3.5 y 3.6 del libro de texto (Folel, van Dam, Feiner and Hughes, Computer Graphics: Principles and Practice, 2nd Ed. in C. Addison Wesley).

1. Rellenado de rectángulos

La tarea de rellenar primitivas puede ser dividida en dos partes: la decisión de cuáles pixeles se tienen que rellenar (que depende de la forma de la primitiva modificada por el recortado), y la decisión más sencilla de cuál valor rellenarlos. En general, la determinación de qué pixeles rellenar consiste en tomar sucesivos líneas de escaneo que intersecan la primitiva y rellenarla en *tramos* de pixeles adyacentes que yacen en la primitiva de izquierda a derecha.

Para rellenar un rectángulo, la conversión a un algoritmo de escaneo es simplemente un ciclo **for** anidado:

for y desde $y_{\text{mín}}$ a $y_{\text{máx}}$ del rectángulo **do** {Cada línea de escaneo}
  for x desde $x_{\text{mín}}$ a $x_{\text{máx}}$ del rectángulo **do** {Cada pixel en el tramo}
    WritePixel(x, y, *valor*);

Un problema que resulta es como tratar las líneas que comparten pixeles. Consideremos el caso de dos rectángulos que comparten una arisata común. Si se trazan los rectángulos con al algoritmo anterior, los pixeles en la arista se trazarán dos veces, lo cual no es deseable como se explicará más adelante. Este es una manifestación de un problema más grande de primitivas que definen un área, el problema es definir cuales pixeles pertenecen a la primitiva y cuales no. Claramente, aquellos pixeles que yacen en el interior matemático

de una primitiva que define un área pertenecen a la primitiva. ¿Pero que pasa con los pixeles sobre las aristas?

Una regla simple es decir que un pixel en la frontera –esto es, un pixel sobre una arista– se considera parte de la primitiva si el semiplano definido por la arista y conteniendo la primitiva yace abajo o a la izquierda de la arista. Esto es, los pixeles en las aristas a la izquierda o abajo serán trazados, pero los pixeles que yacen en las aristas superiores y a la derecha no serán trazados. Una arista vertical compartida, por lo tanto, "pertenece" al rectángulo más a la derecha, de los dos que lo comparten.

2. Rellenado de polígonos

El algoritmo general para escanerar polígonos que se describirá, maneja tanto polígonos convexos como cóncavos, y aún funciona para los polígonos que se autointersecan o tienen huecos interiores. El algoritmo opera calculando tramos que yacen en las aristas izquierda y derecha del polígono. Los extremos de los tramos son calculados por un algoritmo incremental que calcula una línea de escaneo/intersección de la arista a partir de la intersección con la línea de escaneo anterior. La figura 1 ilustra el proceso básico de conversión a un escaneo para un polígono; la figura muestra un polígono y una línea de escaneo que pasa a través de él. La intersección de la línea de escaneo 8 con las aristas FA y CD yacen en coordenadas enteras, mientras que para EF y DE no es así; las intersecciones están marcadas en la figura con líneas pequeñas verticales etiquetadas desde la *a* a la *d*.

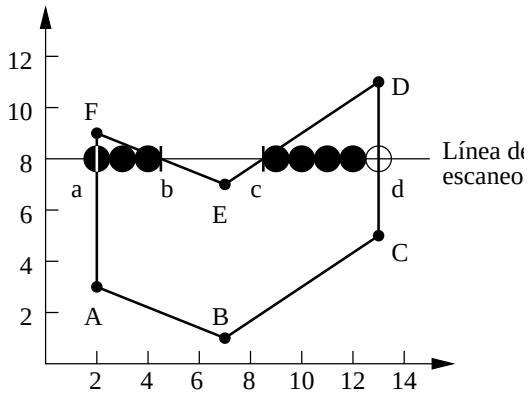

Figura 1: Un polígono y una línea de escaneo en $y = 8$

Se debe de determinar cuáles pixeles en cada línea de escaneo están dentro del polígono y se deben de poner los pixeles correspondientes (en este caso, los tramos desde $x = 2$ hasta 4 y de 9 a 13) a los valores apropiados. Se repite este proceso para cada línea de escaneo que intersecan al polígono, así se escanea al polígono entero.

Los tramos pueden ser rellenados en un proceso de tres pasos:

1. Encontrar la intersección de la línea de escaneo con todas las aristas del polígono.

2. Ordenar los intersecciones de forma incremental por la coordenada x.

3. Rellenar todos los pares de intersecciones que yacen en el interior del polígono, usando la regla de paridad impar para determinar si un punto está dentro de una región: la paridad inicialmente es par, y cada intersección que se encuentre invierte el bit de paridad –se dibuja cuando la paridad es impar, no se dibuja cuando la paridad es par.

Los primeros dos pasos del proceso, encontrar las intersecciones y ordenarlas, serán tratadas en la siguiente sección. En la Fig. 1, la lista ordenada de coordenadas x es $(2, 4{,}5, 8{,}5, 13)$. El paso 3 requiere cuatro puntos más:

3.1. Dada una intersección con un valor x fraccional, arbitrario ¿cómo determinar cual pixel es interior en cualquier lado de la intersección?

3.2. ¿Cómo tratar el caso especial de intersección en coordenadas de pixel enteras?

3.3. ¿Cómo tratar el caso especial en 3.2 para vértices compartidos?

3.4. ¿Cómo tratar el caso especial en 3.2 en el cual los vértices definen una arista horizontal?

Para manejar el caso 3.1, se dice que, si se está aproximando a un intersección fraccional a la derecha y se está dentro del polígono, se redondea hacia abajo la coordenada x de la intersección para definir el pixel como interior; si se está fuera del polígono, se redondea hacia arriba para estar dentro. El caso 3.2 se maneja aplicando el criterio que se manejó para las aristas compartidos para el caso de los rectángulos: si el pixel más a la izquierda en un tramo tiene la coordenada x entera, se define como un pixel interior; si el pixel más

a la derecha tiene la coordenada x entera, éste se define exterior. Para el caso 3.3, se cuenta el $y_{\text{mín}}$ de una arista en el cálculo de paridad pero no se cuenta el vértice $y_{\text{máx}}$; por lo tanto un vértice $y_{\text{máx}}$ se dibuja si y solo si éste es el vértice $y_{\text{mín}}$ de la arista adyacente. El vértice A en la Fig. 1, por ejemplo, se cuenta una sola vez en la cálculo de paridad porque este es el vértice $y_{\text{mín}}$ para la arista FA pero es el vértice $y_{\text{máx}}$ para la arista AB. Por tanto, ambas aristas y tramos son tratados como intervalos que son cerrados en su valor mínimo y abiertos en su valor máximo. Claramente, la regla contraria podría servir, pero la regla explicada parece más natural ya que se trata los puntos terminales mínimos como un punto de entrad y el máximo como un punto de salida. Para el caso 3.4, las aristas horizontales, el efecto deseado es que, como con los rectángulos, las aristas inferiores se dibujan pero no las aristas superiores. Como se verá, esto sucede de forma automática si no se cuentan los vértices de las aristas, ya que no son vértices ni $y_{\text{mín}}$ ni $y_{\text{máx}}$.

2.1. Aristas horizontales

Se tratan de forma correcta las aristas horizontales si no se toman en cuentra sus vértices, como se verá examinando varios casos con la ayuda de la Fig. 2. Considerando la arista inferior AB. El vértice A es un vértice $y_{\text{mín}}$ para la arista JA y AB no contribuye. Por lo tanto, la paridad es impar y el tramo AB se dibuja. La arista vertical BC tiene su $y_{\text{mín}}$ en B, pero de nuevo AB no contribuye. La paridad se vuelve par y el tramo se termina. En el vértice J, la arista IJ tiene un vértice $y_{\text{mín}}$ pero la arista JA no lo tiene, de forma que la paridad se vuelve impar y el tramo dibuja a la arista BC. El tramo que comienza en la arista IJ y choca con C no cambia en C parur es un vértice $y_{\text{máx}}$ para BC, de forma que el tramo continúa a lo largo de la arista inferior CD; en D, sin embargo, la arista DE tiene un vértice $y_{\text{mín}}$, de forma que la paridad inicia par y el tramo termina. En I, la arista IJ tiene su vértice $y_{\text{máx}}$ y la arista HI también no contribuye, de forma que la paridad comienza par y la arista superior IH no es dibujado. En H, sin embargo, la arista GH tiene un vértice $y_{\text{mín}}$, la paridad de vuelve impar, el tramo se dibuja desde H al pixel a la izquierda de la intersección con la arista EF. Finalmente, no existe vértice $y_{\text{mín}}$ en G, ni en F, de forma que la arista FG no se dibuja.

El algoritmo anterior resuelve los vértices compartidos en un polígono, las aristas compartidos por dos polígonos adyacentes y las aristas horizontales. También permite polígonos autointersectados.

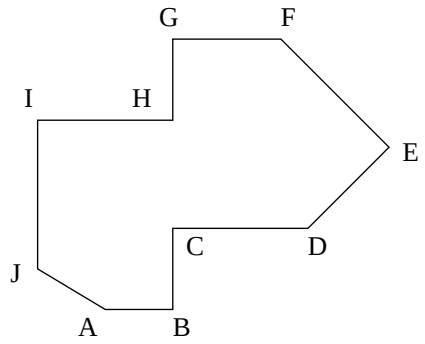

Figura 2: Aristas horizontales en un polígono

2.2. Coherencia de arista y el algoritmo de la línea de escaneo

El paso 1 en el procedimiento –el cálculo de las intersecciones– debe pensarse de forma clara porque si nó será lento. Se debe evitar el uso de técnicas de fuerza bruta para probar si cada arista del polígono intersecta una línea de escaneo dada. Aún más, se nota que muchas aristas intersectados por la línea de escaneo i también son intersectadas por la línea de escaneo $i + 1$. Esta *coherencia de arista* ocurre a lo largo de una arista para tantas líneas de escaneo intersecan a la arista. Cuando se mueve de una línea de escaneo a la siguiente, se puede calcular la nueva intersección x de la arista sobre la base de la intersección x anterior, de la misma forma como se calculo el pixel siguiente a partir del pixel actual en el algoritmo del trazo de la línea, usando:

$$x_{i+1} = x_i + 1/m$$

donde m es la pendiente de la arista.

Se puede evitar el uso de fracciones manteniendo solamente el numerador de la fracción y observando que la parte fraccional es más grande que uno cuando el numerador es más grande que el denominador. Esta técnica es realizada en el algoritmo de la Fig. 3, usando la variable *increment* para mantener las adiciones sucesivas para el numerador hasta que este tiene un "sobreflujo" pasando el denominador, es entonces cuando el denominador se decrementa por el denominador y x se incrementa.

Ahora se desarrollará el *algoritmo para líneas de escaneo* que toma ventaja de la coherencia de aristas y, para cada línea de escaneo, mantiene el conjunto de aristas que ésta intersecta y los puntos de intersección en una

```
Require: xmin, ymin, xmax, ymax, value
  x ← xmin
  numerator ← xmax − xmin
  denominator ← ymax − ymin
  increment ← denominator
  for y ← ymin; y ≤ ymax; y++ do
    WritePixel( x, y, value )
    increment+ = numerator
    if increment > denominator then
      {Sobreflujo, entonces se redondea el siguiente pixel}
      x++
      increment− = denominator
```

Figura 3: Algoritmo para escanear una arista izquierda de un polígono

estructura de datos llamada *tabla activa de aristas* (TAA). Las aristas en la TAA están ordenados en sus valores de intersección en x de forma que para rellenar los tramos definidos por pares de valores de intersección (ya redondeados) –esto es, los extremos de los tramos. Cuando se mueve a la siguiente línea de escaneo en $y + 1$, la TAA se pone al día. Primero, las aristas actuales en la TAA que no son intersectados por esta siguiente línea de escaneo (i.e. aquellas aristas cuya $y_{\text{máx}} = y$) se borran. Segundo, cualesquiera aristas nuevas intersectadas por esta siguiente línea de escaneo (i.e. aquellas aristas cuya $y_{\text{mín}} = y + 1$ se adicionan a la TAA. Finalmente, nuevas intersecciones en x se calculan, usando el algoritmo incremental ya presentado, para las aristas que estuvieron en la TAA pero que aún no son completados.

Para realizar la adición de aristas a la TAA de forma eficiente, inicialmente se crea una *tabla de aristas* (TA) global, conteniendo todos las aristas ordenadas por su coordenada y más pequeña. la TA se construye típicamente usando un ordenamiento de cubeta, con tantas cubetas como haya líneas de escaneo. Dentro de cada cubeta, las aristas se mantienen en orden incremental según la coordenada x del punto más bajo. Cada entrada en la TA contiene la coordenada $y_{\text{máx}}$ de la arista, la coordenada x de punto más bajo ($x_{\text{mín}}$) y el incremento en x usado en el paso de una línea de escaneo a la siguiente, $1/m$. La Fig. 4 como se deberían de ordenar las seis aristas del polígono de la Fig. 1. La Fig. 4 muestra la TAA para la línea de escaneo 9 y 10 para ese polígono (para la realización tal vez sea necesaria una bandera para indicar si es una arista derecho o izquierdo).

Una vez que la TA se ha realizado, los pasos de procesamiento para el algoritmo de líneas de escaneo son las siguientes:

3.1. Poner y al valor más pequeño de la coordenada y que esté en la TA; i.e., y para la primera cubeta no vacía.

3.2. Inicializar la TAA a vacío

3.3. Repetir hasta que la TAA y la TA estén vacíos

a) Mover de la TA la cubeta y a la TAA aquellas aristas cuya $y_{\text{mín}} = y$ (aristas de entrada).

b) Quitar de la TAA aquellas entradas para las cuales $y = y_{\text{máx}}$ (las aristas no involucrados en la siguiente línea de escaneo), entonces se ordena la TAA en x (es fácil dado que la TA está preordenada)

c) Rellenar los pixeles deseados sobre la línea de escaneo y usando pares de coordenadas x de la TAA.

d) Incrementa y en 1 (a la coordenada de la siguiente línea de escaneo.

e) Para cada arista no vertical que quede en la TAA, poner al día x para la nueva y.

Este algoritmo usa tanto la coherencia de arista para calcular las intersecciones en x, como la coherencia en la línea de escaneo (junto con el ordenamiento) para calcular los tramos. Dado que el ordenamiento trabaja sobre un número pequeño de aristas y dado que el reordenamiento del paso $3b$ es aplicada a una lista mayormente, o completamente, ordenada, pueden usarse tanto el ordenamiento por inserción o aún el ordenamiento de burbuja que es $O(N)$ en este caso.

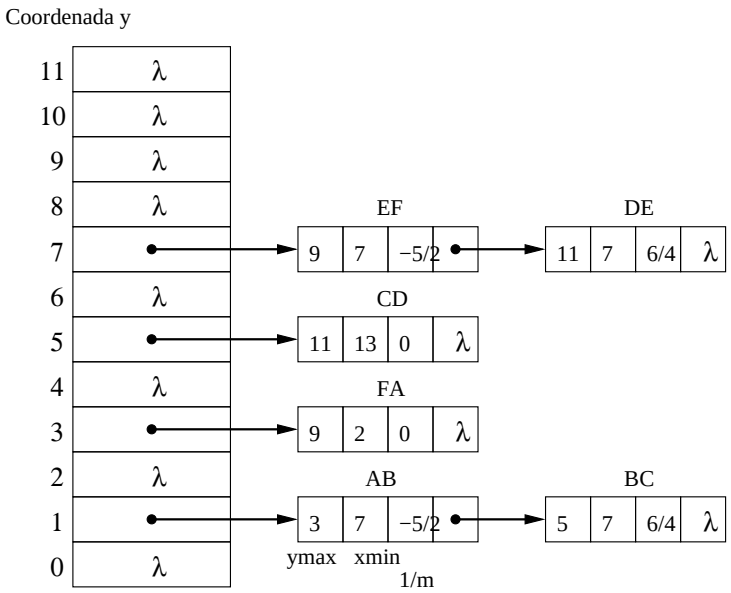

Figura 4: Tabla de aristas ordenada por cubetas para el polígono de la figura 1

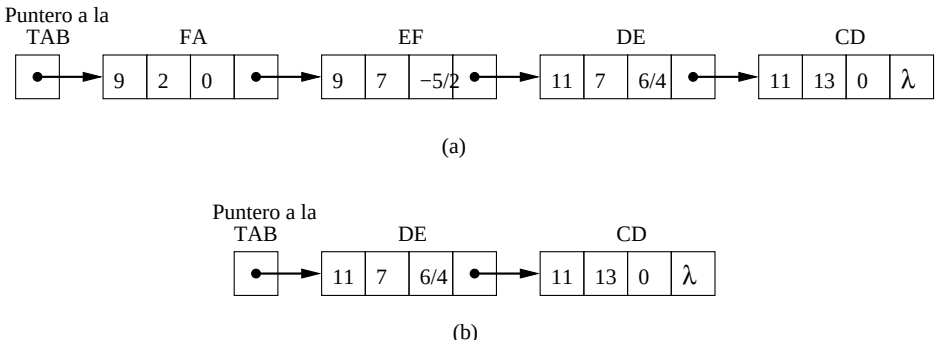

Figura 5: Tabla activa de aristas para el polígono de la figura 1. (a) Linea de escaneo 9. (b) Para la línea de escaneo 10. (Note que la coordenada x de DE en (b) ha sido redondeada hacia arriba para la arista izquierdo).