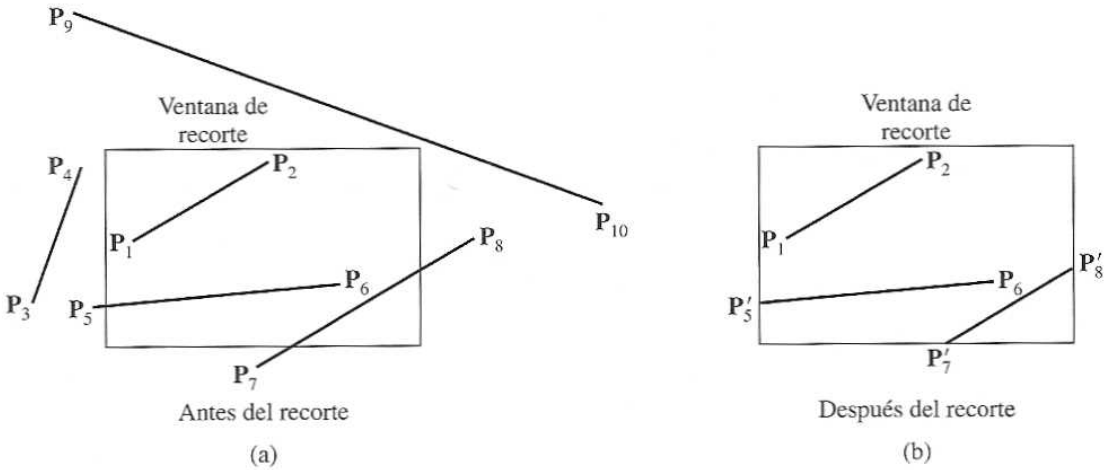

FIGURA 6.11. Recorte de segmentos de línea recta utilizando una ventana de recorte rectangular estándar.

tro u. A modo de ejemplo, el límite izquierda de la ventana está en la posición xw_{min}, por lo que sustituimos x por este valor y resolvemos para u, y calculamos el valor de intersección y correspondiente. Si este valor de u se encuentra fuera del rango que varía desde 0 a 1, el segmento de línea no intersecta con dicha arista de la ventana. Pero si el valor de u se encuentra dentro del rango que varía entre 0 y 1, parte de la línea se encuentra dentro de dicho borde. Podemos a continuación procesar esta porción interior del segmento de línea con respecto a los demás límites de recorte hasta que hayamos recortado la línea entera o encontremos una parte que esté dentro de la ventana.

El procesamiento de segmentos de línea de una escena utilizando la sencilla técnica descrita en el párrafo anterior es directo, pero no muy eficiente. Es posible reformular la prueba inicial y los cálculos de intersección para reducir el tiempo de procesamiento de un conjunto de segmentos de línea. Se han desarrollado algoritmos de recorte de líneas más rápidos. Algunos de estos algoritmos se han diseñado explícitamente para imágenes bidimensionales y algunos se adaptan fácilmente a conjuntos de segmentos de línea tridimensionales.

Recorte de líneas de Cohen-Sutherland

Éste es uno de algoritmos más antiguos que se ha desarrollado para el recorte de líneas rápido, y variaciones de este método se utilizan ampliamente. El tiempo de procesamiento se reduce en el método de Cohen-Sutherland realizando más pruebas antes de proceder con los cálculos de las intersecciones. Inicialmente, se asigna a cada punto extremo de las líneas de una imagen un valor binario de cuatro dígitos llamado **código de región**. Cada bit se utiliza para indicar si está dentro o fuera de uno de los límites de la ventana de recorte. Podemos hacer referencia a las aristas de la ventana en cualquier orden. La Figura 6.12 muestra una posible ordenación en la que los bits están numerados de 1 a 4 de derecha a izquierda. Por tanto, para esta ordenación, el bit situado más a la derecha (bit 1) hace referencia al borde izquierdo de la ventana de recorte, y el situado más a la izquierda (bit 4) hace referencia al borde superior de la ventana. Un valor de 1 (o *verdadero*) en cualquier bit indica que el punto extremo está fuera de la ventana. De forma similar, un valor de 0 (o *falso*) en cualquier bit indica que el punto extremo no está fuera (está dentro o sobre) del límite correspondiente de la ventana. A veces, un código de región se denomina **código de «fuera»** porque un valor de 1en cualquier bit indica que el punto del espacio está fuera del correspondiente borde de recorte.

Cada arista de la ventana de recorte divide el espacio bidimensional en un semiespacio interior y un semiespacio exterior. En total, los cuatro límites de la ventana crean nueve regiones. La Figura 6.13 enumera el valor del código binario en cada una de estas regiones. Por tanto, a un punto extremo que esté situado debajo y a la izquierda de la ventana de recorte se le asigna un código de región 0101, y el valor del código de región de cualquier punto interior a la ventana de recorte es 0000.

Los valores de los bits de un código de región se determinan comparando los valores de las coordenadas (x, y) de un punto extremo con los límites de recorte. El bit 1 se pone a 1 si $x < xw_{\text{min}}$. Los valores de los otros

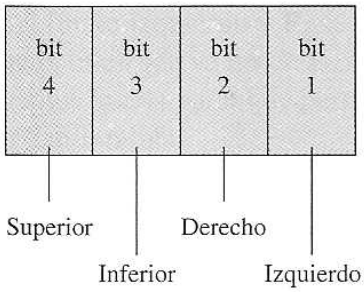

FIGURA 6.12. Una posible ordenación de los límites de la ventana de recorte correspondiente a las posiciones de los bits en el código de región de los puntos extremos del método Cohen-Sutherland.

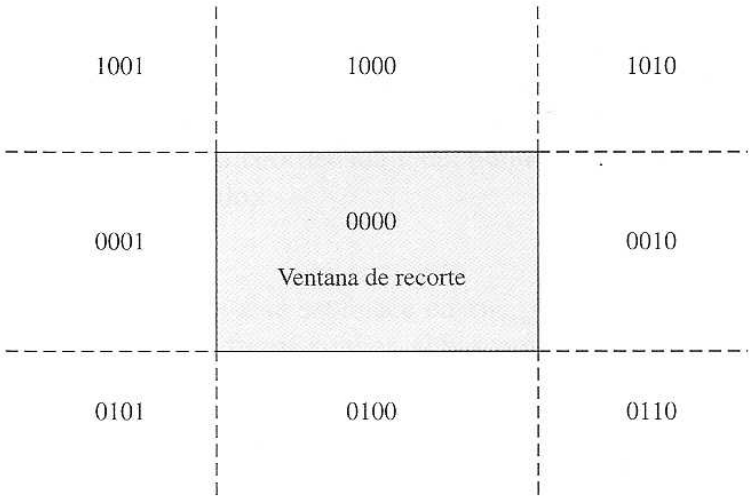

FIGURA 6.13. Los nueve códigos de región para la identificación de la posición de un punto extremo de una línea, relativos a los límites de la ventana de recorte.

tres bits se determinan de forma semejante. En lugar de utilizar pruebas de desigualdad, podemos determinar más eficientemente los valores de un código de región utilizando las operaciones de procesamiento de bits y siguiendo dos pasos: (1) Calcular las diferencias entre las coordenadas de los puntos extremos y los límites de recorte. (2) Utilizar el bit de signo resultante de cada cálculo de diferencia para cambiar el valor correspondiente de cada código de región. En el caso del esquema de ordenación mostrado en la Figura 6.12, el bit 1 es el bit de signo de $x - xw_{\min}$; el bit 2 es el bit de signo de $xw_{\max} - x$; el bit 3 es el bit de signo de $y - yw_{\min}$; y el bit 4 es el bit de signo de $yw_{\max} - y$.

Una vez que hemos establecido los códigos de región de todos los puntos extremos de todas las líneas, podemos determinar rápidamente qué líneas se encuentran completamente dentro de la ventana de recorte y cuáles se encuentran claramente fuera. Cualesquiera líneas que se encuentran completamente contenidas dentro de las aristas de la ventana tienen un código de región 0000 en ambos puntos extremos y estos segmentos de línea los guardamos. Cualquier línea que tenga un código de región de valor 1 en el mismo bit en cada punto extremo está completamente fuera del rectángulo de recorte, por lo que eliminamos dicho segmento de línea. A modo de ejemplo, una línea que tenga un código de región 1001 en un punto extremo y un código 0101 en el otro punto extremo está completamente a la izquierda de la ventana de recorte, como lo indica el valor 1 en el primer bit de cada código de región.

Podemos realizar las pruebas de dentro-fuera para los segmentos de línea utilizando operadores lógicos. Cuando la operación *or* entre los dos códigos de región de los puntos extremos de un segmento de línea es *falsa* (0000), la línea se encuentra dentro de la ventana de recorte. Por tanto, guardamos la línea y procedemos a comprobar la línea siguiente de la descripción de la escena. Cuando la operación *and* entre los dos códigos de región de los puntos extremos de una línea es *verdadera* (no 0000), la línea está completamente fuera de la ventana de recorte, y podemos eliminarla de la descripción de la escena.

En el caso de las líneas que no se pueden identificar como que están completamente dentro o completamente fuera de una ventana de recorte mediante las pruebas del código de región, se comprueba a continuación si intersectan con los límites de la ventana. Como se muestra en la Figura 6.14, los segmentos de línea pueden intersectar con los límites de recorte sin entrar dentro del interior de la ventana. Por tanto, para recortar un segmento de línea podrían ser necesarios varios cálculos de intersecciones, dependiendo del orden en

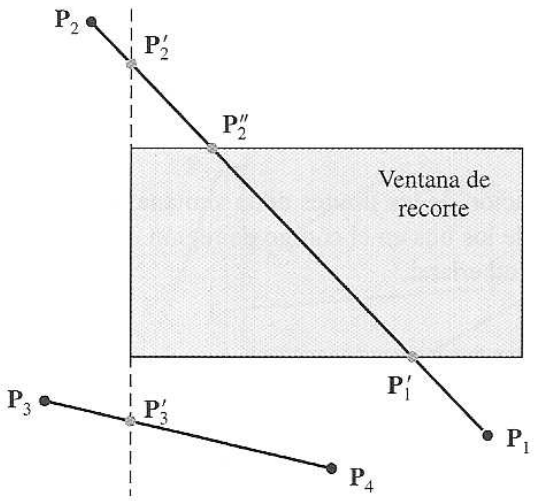

FIGURA 6.14. Las líneas que se extienden desde una región de la ventana de recorte a otra pueden atravesar la ventana de recorte o pueden intersectar con uno o más límites de recorte sin entrar en el interior de la ventana.

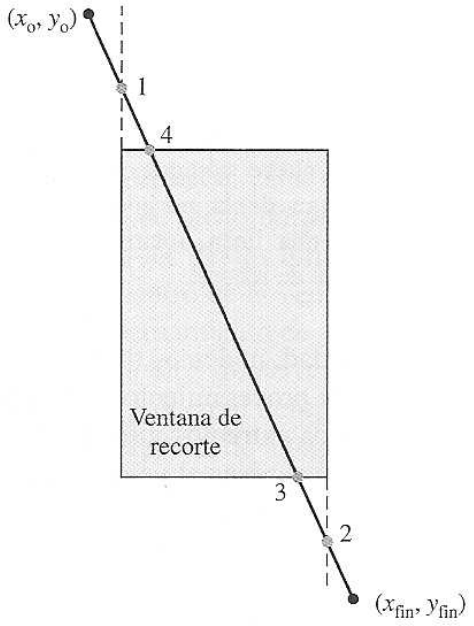

FIGURA 6.15. Los cuatro puntos de intersección (etiquetados de 1 a 4) de un segmento de línea que se recorta de acuerdo con los límites de la ventana en el orden izquierda, derecha, inferior, superior.

que procesemos los límites de recorte. Cuando procesamos cada arista de la ventana de recorte, se recorta una parte de la línea, y la parte que permanece de la línea se comprueba frente a los restantes límites de la ventana. Continuaremos eliminando partes hasta que la línea esté totalmente recortada o la parte que permanece de la línea se encuentre dentro de la ventana de recorte. En el siguiente estudio, asumimos que las aristas de la ventana se procesan en el orden: izquierda, derecha, inferior, superior. Para determinar si una línea cruza un límite de recorte seleccionado, podemos comprobar los valores correspondientes de los bits de los códigos de región de los dos puntos extremos. Si uno de estos bits es 1 y el otro es 0, el segmento de línea cruza dicho límite.

La Figura 6.14 muestra dos segmentos de línea que se pueden identificar inmediatamente como completamente dentro o completamente fuera de la ventana de recorte. Los códigos de región de la línea desde $\mathbf{P}_1$ a $\mathbf{P}_2$ son 0100 y 1001. Por tanto, $\mathbf{P}_1$ está dentro del límite izquierdo de recorte y $\mathbf{P}_2$ está fuera de dicho límite. A continuación, calculamos la intersección $\mathbf{P'}_2$, y recortamos la parte de la línea desde $\mathbf{P}_2$ a $\mathbf{P'}_2$. La parte que permanece de la línea se encuentra dentro de la línea límite derecha, y por ello a continuación comprobamos el límite inferior. El punto extremo $\mathbf{P}_1$ se encuentra por debajo de la arista inferior de recorte y $\mathbf{P'}_2$ se encuentra por encima de ésta, por lo que determinamos la intersección con esta arista ($\mathbf{P'}_1$). Eliminamos la parte de la línea desde $\mathbf{P}_1$ a $\mathbf{P'}_1$ y procedemos con la arista superior de la ventana. Allí determinamos la intersección $\mathbf{P''}_2$. El último paso consiste en recortar la parte situada por encima del límite superior y guardar el segmento interior desde $\mathbf{P'}_1$ hasta $\mathbf{P''}_2$. En el caso de la segunda línea, obtenemos que el punto $\mathbf{P}_3$ se encuentra fuera

del límite izquierdo y $\mathbf{P}_4$ se encuentra dentro. Por tanto, calculamos la intersección $\mathbf{P'}_3$ y eliminamos la parte de la línea que va desde $\mathbf{P}_3$ a $\mathbf{P'}_3$. Comprobando los códigos de región de los puntos extremos $\mathbf{P'}_3$ y $\mathbf{P}_4$, observamos que la parte que permanece de la línea se encuentra por debajo de la ventana de recorte y se puede eliminar también.

Cuando se recorta un segmento de línea utilizando esta técnica se puede calcular una intersección con los cuatro límites de recorte, dependiendo de cómo se procesen los puntos extremos de la línea y qué ordenación utilicemos en los límites. La Figura 6.15 muestra las cuatro intersecciones que se podrían calcular para un segmento de línea que se procesa frente a las aristas de la ventana de recorte en el orden izquierda, derecha, inferior, superior. Por tanto, se han desarrollado variaciones de esta técnica básica en un esfuerzo por reducir los cálculos de intersecciones.

Para determinar una intersección de un segmento de línea con la frontera, podemos utilizar la forma pendiente-punto de corte de la ecuación de la línea. Para una línea con coordenadas en sus puntos extremos (x_0, y_0) y (x_{fin}, y_{fin}), la coordenada y del punto de intersección con un borde de recorte vertical se puede obtener mediante el cálculo,

$$y = y_0 + m(x - x_0) \tag{6.14}$$

donde el valor de x se establece en xw_{min} o xw_{max}, y la pendiente de esta línea se calcula como $m = (y_{fin} - y_0)/(x_{fin} - x_0)$. De forma similar, si buscamos la intersección con el borde horizontal, la coordenada x se puede calcular como:

$$x = x_0 + \frac{y - y_0}{m} \tag{6.15}$$

(estableciendo y en yw_{min} o en yw_{max}.

En los siguientes procedimientos se proporciona una implementación del algoritmo de recorte de líneas de Cohen-Sutherland bidimensional. La ampliación de este algoritmo a tres dimensiones es directa. Estudiaremos los métodos de visualización tridimensional en el capítulo siguiente.

```
class wcPt2D {
   public:
      GLfloat x, y;
};

inline GLint round (const GLfloat a)  { return GLint (a + 0.5); }

/*  Define un código de cuatro bits para cada una de las regiones
 *  exteriores de una ventana rectangular de recorte.
 */

const GLint winLeftBitCode = 0x1;
const GLint winRightBitCode = 0x2;
const GLint winBottomBitCode = 0x4;
const GLint winTopBitCode = 0x8;

/*  Un código de región de máscara de bit se asigna también a cada extremo
 *  de un segmento de línea de entrada, de acuerdo con su posición respecto de
 *  los cuatro bordes de una ventana de recorte rectangular de entrada.
 *
 *  Un extremo con un valor de código de región de 0000 se encuentra dentro
 *  de la ventana de recorte, en otro caso, está fuera al menos respecto
```

```
 *  de uno de  los límites de recorte. Si la operación 'or'entre los dos
 *  códigos de los puntos extremos da como resultado un valor falso, la línea
 *  completa definida por dichos dos puntos se guarda (se acepta).
 *  Si la operación 'and' entre los códigos de los puntos extremos da como
 *  resultado verdadero, quiere decir que la línea está completamente fuera de
 *  la ventana de recorte y se elimina (se rechaza) para posteriores
 *  procesamientos.
 */

inline GLint inside (GLint code) { return GLint (!code); }
inline GLint reject (GLint code1, GLint code2)
                          { return GLint (code1 & code2); }
inline GLint accept (GLint code1, GLint code2)
                          { return GLint (!(code1 | code2)); }

GLubyte encode (wcPt2D pt, wcPt2D winMin, wcPt2D winMax)

{
  GLubyte code = 0x00;

  if (pt.x < winMin.x)
    code = code | winLeftBitCode;
  if (pt.x > winMax.x)
    code = code | winRightBitCode;
  if (pt.y < winMin.y)
    code = code | winBottomBitCode;
  if (pt.y > winMax.y)
    code = code | winTopBitCode;
  return (code);
}

void swapPts (wcPt2D * p1, wcPt2D * p2)
{
  wcPt2D tmp;
  tmp = *p1; *p1 = *p2; *p2 = tmp;
}

void swapCodes (GLubyte * c1, GLubyte * c2)
{
   GLubyte tmp;
   tmp = *c1; *c1 = *c2; *c2 = tmp;
}

void lineClipCohSuth (wcPt2D winMin, wcPt2D winMax, wcPt2D p1, wcPt2D p2)
{
   GLubyte code1, code2;
   GLint done = false, plotLine = false;
   GLfloat m;
```

```
    while (!done) {
       code1 = encode (p1, winMin, winMax);
       code2 = encode (p2, winMin, winMax);
       if (accept (code1, code2)) {
          done = true;
       plotLine = true;
     }
     else
       if (reject (code1, code2))
         done = true;
       else {
         /*  Etiqueta el punto extremo que está fuera de la ventana
          *  de visualización como p1. */
         if (inside (code1)) {
           swapPts (&p1, &p2);
           swapCodes (&code1, &code2);
         }
         /*  Usa la pendiente m para hallar la intersección línea-límite
          *  de recorte.*/
         if (p2.x != p1.x)
           m = (p2.y - p1.y) / (p2.x - p1.x);
         if (code1 & winLeftBitCode) {
           p1.y += (winMin.x - p1.x) * m;
           p1.x = winMin.x;
         }
         else
           if (code1 & winRightBitCode) {
             p1.y += (winMax.x - p1.x) * m;
             p1.x = winMax.x;
           }
           else
             if (code1 & winBottomBitCode) {
               /*  Es necesario actualizar p1.x sólo para líneas no verticales.  */
               if (p2.x != p1.x)
                 p1.x += (winMin.y - p1.y) / m;
               p1.y = winMin.y;
             }
             else
               if (code1 & winTopBitCode) {
                 if (p2.x != p1.x)
                   p1.x += (winMax.y - p1.y) / m;
                 p1.y = winMax.y;
               }
       }
   }
   if (plotLine)
     lineBres (round (p1.x), round (p1.y), round (p2.x), round (p2.y));
}
```

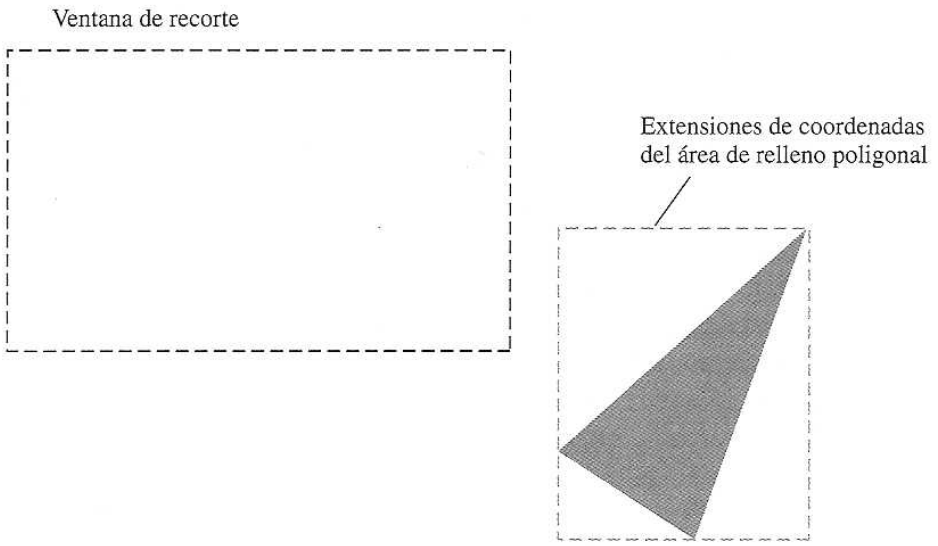

FIGURA 6.24. Un área de relleno poligonal con las extensiones de coordenadas fuera del límite derecho de recorte.

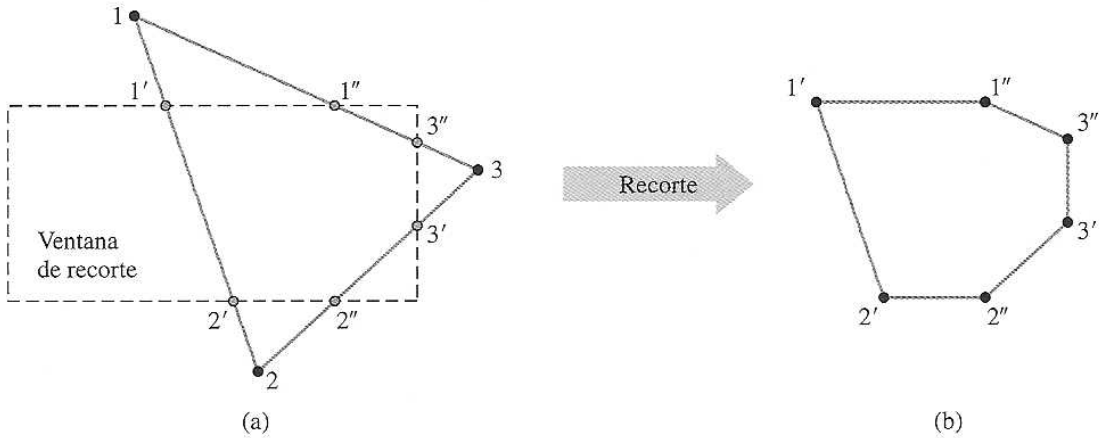

FIGURA 6.25. Un área de relleno poligonal convexa (a), definida por la lista de vértices {1, 2, 3}, se recorta para producir la forma del área de relleno mostrada en (b), que se define por la lista de vértices de salida {1′, 2′, 2″, 3′, 3″, 1″}.

Cuando no podemos identificar si un área de relleno está totalmente dentro o totalmente fuera de la ventana de recorte, necesitamos entonces localizar las intersecciones del polígono con los límites de recorte. Una manera de implementar el recorte de polígonos convexos consiste en crear una nueva lista de vértices en cada límite de recorte, y entonces pasar esta nueva lista de vértices al siguiente recortador de límites. La salida de la etapa final de recorte es la lista de vértices del polígono recortado (Figura 6.25). Para el recorte de polígonos cóncavos, es necesario modificar esta técnica básica para que se puedan generar múltiples listas de vértices.

Recorte de polígonos de Sutherland-Hodgman

Un método eficiente de recorte de áreas de relleno poligonales convexas, desarrollado por Sutherland y Hodgman, consiste en enviar los vértices del polígono a través de cada etapa de recorte para que un único vértice recortado se pueda pasar inmediatamente a la etapa siguiente. Esto elimina la necesidad de producir como salida un conjunto de vértices en cada etapa de recorte, lo que permite que las subrutinas de recorte de bordes se implementen en paralelo. La salida final es una lista de vértices que describe las aristas del área de relleno poligonal recortada.

Ya que el algoritmo de Sutherland-Hodgman produce sólo una lista de vértices de salida, no puede generar correctamente los dos polígonos de salida de la Figura 6.22(b), que son el resultado del recorte del polí-

gono cóncavo mostrado en el apartado (a) de la figura. Sin embargo, se puede añadir un procesamiento adicional al algoritmo de Sutherland-Hodgman para obtener múltiples listas de vértices como salida, con el fin de poder realizar el recorte de polígonos cóncavos generales. El algoritmo básico de Sutherland-Hodgman es capaz de procesar polígonos cóncavos cuando el área de relleno recortada se puede describir con una única lista de vértices.

La estrategia general de este algoritmo es enviar el par de puntos extremos de cada sucesivo segmento de línea del polígono a través de la serie de recortadores (izquierdo, derecho, inferior y superior). Tan pronto como un recortador completa el procesamiento de un par de vértices, los valores de las coordenadas recortadas, si existen, para dicha arista se envían al siguiente recortador. Después, el primer recortador procesa el siguiente par de puntos extremos. De este modo, los recortadores individuales de bordes pueden funcionar en paralelo.

Existen cuatro posibles casos que hay que considerar cuando se procesa una arista de un polígono de acuerdo con uno de los límites de recorte. Una posibilidad es que el primer punto extremo de la arista esté fuera del límite de recorte y el segundo extremo esté dentro. O, ambos extremos podrían estar dentro del límite de recorte. Otra posibilidad es que el primer extremo esté dentro del límite de recorte y el segundo fuera. Y, finalmente, ambos puntos extremos podrían estar fuera del límite de recorte.

Para facilitar el paso de los vértices de una etapa de recorte a la siguiente, la salida de cada recortador se puede formular como se muestra en la Figura 6.26. A medida que cada sucesivo par de puntos extremos se pasa a uno de los cuatro recortadores, se genera una salida para el siguiente recortador de acuerdo con los resultados de las siguientes pruebas.

(1) El primer vértice de entrada está fuera del borde de la ventana de recorte y el segundo está dentro, tanto el punto de intersección de la arista del polígono con el borde de la ventana como el segundo vértice se envían al siguiente recortador.

(2) Si ambos vértices de entrada se encuentran dentro de este borde de la ventana de recorte, sólo el segundo vértice se envía al siguiente recortador.

(3) Si el primer vértice está dentro de este borde de la ventana de recorte y el segundo vértice está fuera, sólo la intersección de la arista del polígono con el borde de la ventana de recorte se envía al siguiente recortador.

(4) Si ambos vértices de entrada se encuentran fuera de este borde de la ventana de recorte, no se envían vértices al siguiente recortador.

El último recortador de esta serie genera una lista de vértices que describe el área final de relleno recortada.

La Figura 6.27 proporciona un ejemplo del algoritmo de recorte de polígonos de Sutherland-Hodgman para un área de relleno definida por el conjunto de vértices {1, 2, 3}. Tan pronto como un recortador recibe un par de puntos extremos, determina la salida apropiada utilizando las pruebas mostradas en la Figura 6.26.

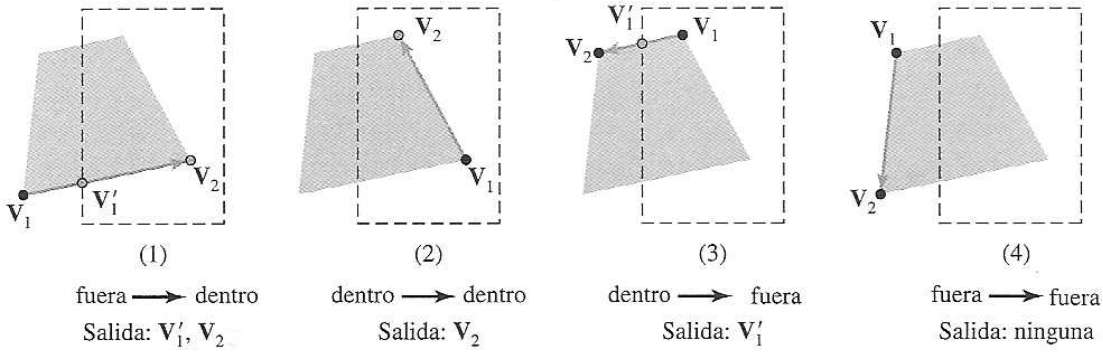

FIGURA 6.26. Las cuatro posibles salidas generadas por el recortador izquierdo, dependiendo de la posición de un par de puntos extremos respecto del borde izquierdo de la ventana de recorte.

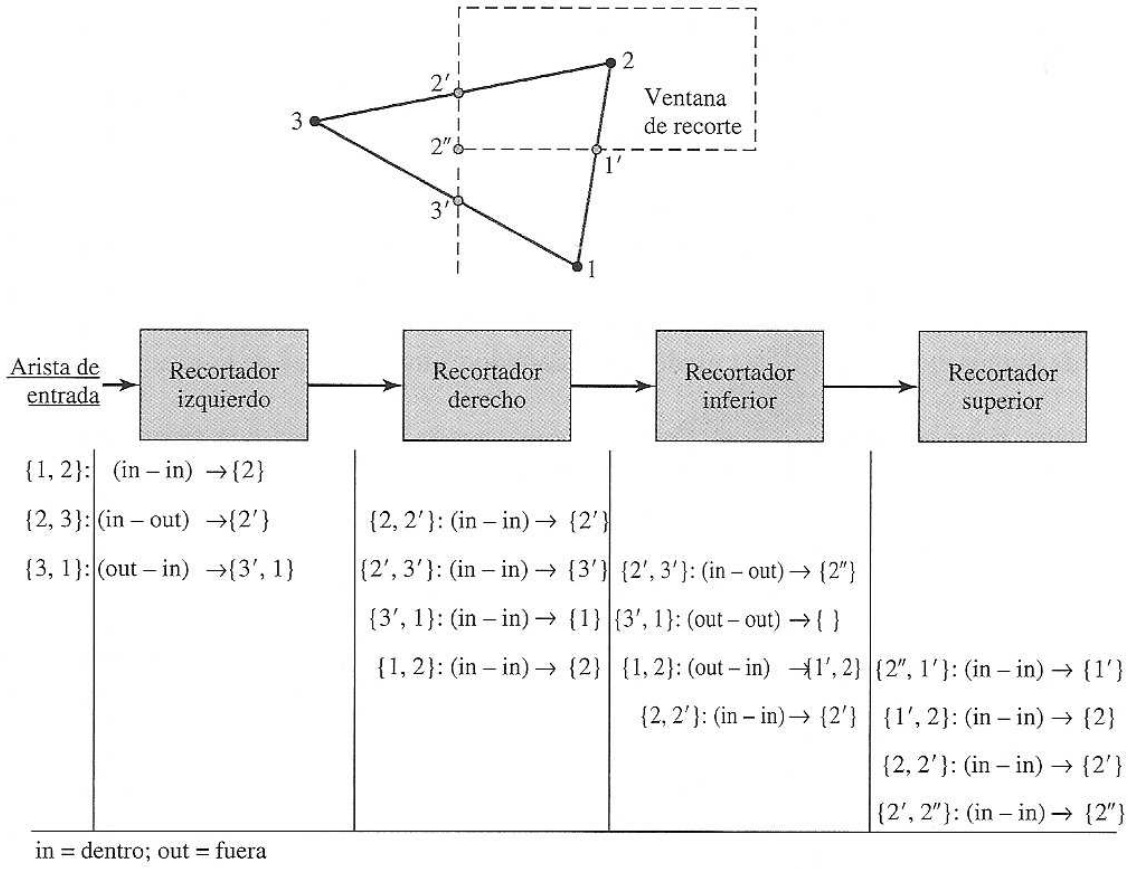

FIGURA 6.27. Procesamiento de un conjunto de vértices de un polígono, {1, 2, 3}, mediante los recortadores de bordes utilizando el algoritmo de Sutherland-Hodgman. El conjunto final de vértices recortados es {1′, 2, 2′, 2″}.

Estas salidas se pasan sucesivamente desde el recortador izquierdo al derecho, al inferior y al superior. La salida del recortador superior es el conjunto de vértices que define el área de relleno recortada. En este ejemplo, la lista de vértices de salida es {1′, 2, 2′, 2″}.

En el siguiente conjunto de procedimientos se muestra una implementación secuencial del algoritmo de recorte de polígonos de Sutherland-Hodgman. Un conjunto de vértices de entrada se convierte en una lista de vértices de salida mediante las subrutinas de recorte izquierda, derecha, inferior y superior.

```
typedef enum { Left, Right, Bottom, Top } Boundary;
const GLint nClip = 4;

GLint inside (wcPt2D p, Boundary b, wcPt2D wMin, wcPt2D wMax)
{
    switch (b) {
    case Left: if (p.x < wMin.x) return (false); break;
    case Right: if (p.x > wMax.x) return (false); break;
    case Bottom: if (p.y < wMin.y) return (false); break;
    case Top: if (p.y > wMax.y) return (false); break;
    }
    return (true);
}
GLint cross (wcPt2D p1, wcPt2D p2, Boundary winEdge, wcPt2D wMin, wcPt2D wMax)
{
    if (inside (p1, winEdge, wMin, wMax) == inside (p2, winEdge, wMin, wMax))
```

```
        return (false);
    else return (true);
}

wcPt2D intersect (wcPt2D p1, wcPt2D p2, Boundary winEdge, wcPt2D wMin, wcPt2D wMax)
{
    wcPt2D iPt;
    GLfloat m;

    if (p1.x != p2.x) m = (p1.y - p2.y) / (p1.x - p2.x);
    switch (winEdge) {
    case Left:
        iPt.x = wMin.x;
        iPt.y = p2.y + (wMin.x - p2.x) * m;
        break;
    case Right:
        iPt.x = wMax.x;
        iPt.y = p2.y + (wMax.x - p2.x) * m;
        break;
    case Bottom:
        iPt.y = wMin.y;
        if (p1.x != p2.x) iPt.x = p2.x + (wMin.y - p2.y) / m;
        else iPt.x = p2.x;
        break;
    case Top:
        iPt.y = wMax.y;
        if (p1.x != p2.x) iPt.x = p2.x + (wMax.y - p2.y) / m;
        else iPt.x = p2.x;
        break;
    }
    return (iPt);
}

void clipPoint (wcPt2D p, Boundary winEdge, wcPt2D wMin, wcPt2D wMax,
                wcPt2D * pOut, int * cnt, wcPt2D * first[], wcPt2D * s)
{
    wcPt2D iPt;

    /* Si no existe ningún punto anterior para este límite de recorte,
     * guardar este punto. */
    if (!first[winEdge])
        first[winEdge] = &p;
    else
        /* Existe un punto previo. Si p y un punto anetrior cruzan este
         * límite de recorte, hallar la interseccción. Recortar de acuerdo
         * con el siguiente límite, si lo hay. Si no hay más límites de recorte,
         * añadir la intersección a la lista de salida. */

        if (cross (p, s[winEdge], winEdge, wMin, wMax)) {
            iPt = intersect (p, s[winEdge], winEdge, wMin, wMax);
            if (winEdge < Top)
```

```
            clipPoint (iPt, b+1, wMin, wMax, pOut, cnt, first, s);
         else {
            pOut[*cnt] = iPt; (*cnt)++;
         }
      }

   /* Guardar p como el punto más reciente para este límite de recorte. */
   s[winEdge] = p;

   /* Para todos, si el punto está dentro, pasar al siguiente límite
    * de recorte, si lo hay. */
   if (inside (p, winEdge, wMin, wMax))
      if (winEdge < Top)
         clipPoint (p, winEdge + 1, wMin, wMax, pOut, cnt, first, s);
      else {
         pOut[*cnt] = p; (*cnt)++;
      }
   }

void closeClip (wcPt2D wMin, wcPt2D wMax, wcPt2D * pOut,
                GLint * cnt, wcPt2D * first [ ], wcPt2D * s)
{
   wcPt2D pt;
   Boundary winEdge;

   for (winEdge = Left; winEdge <= Top; winEdge++) {
      if (cross (s[winEdge], *first[winEdge], winEdge, wMin, wMax)) {
         pt = intersect (s[winEdge], *first[winEdge], winEdge, wMin, wMax);
         if (winEdge < Top)
            clipPoint (pt, winEdge + 1, wMin, wMax, pOut, cnt, first, s);
         else {
            pOut[*cnt] = pt; (*cnt)++;
         }
      }
   }
}

GLint polygonClipSuthHodg (wcPt2D wMin, wcPt2D wMax, GLint n, wcPt2D * pIn,
                           wcPt2D * pOut)
{
   /* El parámetro "first" guarda un puntero a primer punto porcesado por un
    * límite de recorte; "s" almacena el punto más recientemente procesado
    * por el límite de recorte.*/
   wcPt2D * first[nClip] = { 0, 0, 0, 0 }, s[nClip];
   GLint k, cnt = 0;

   for (k = 0; k < n; k++)
      clipPoint (pIn[k], Left, wMin, wMax, pOut, &cnt, first, s);
   closeClip (wMin, wMax, pOut, &cnt, first, s);
   return (cnt);
}
```