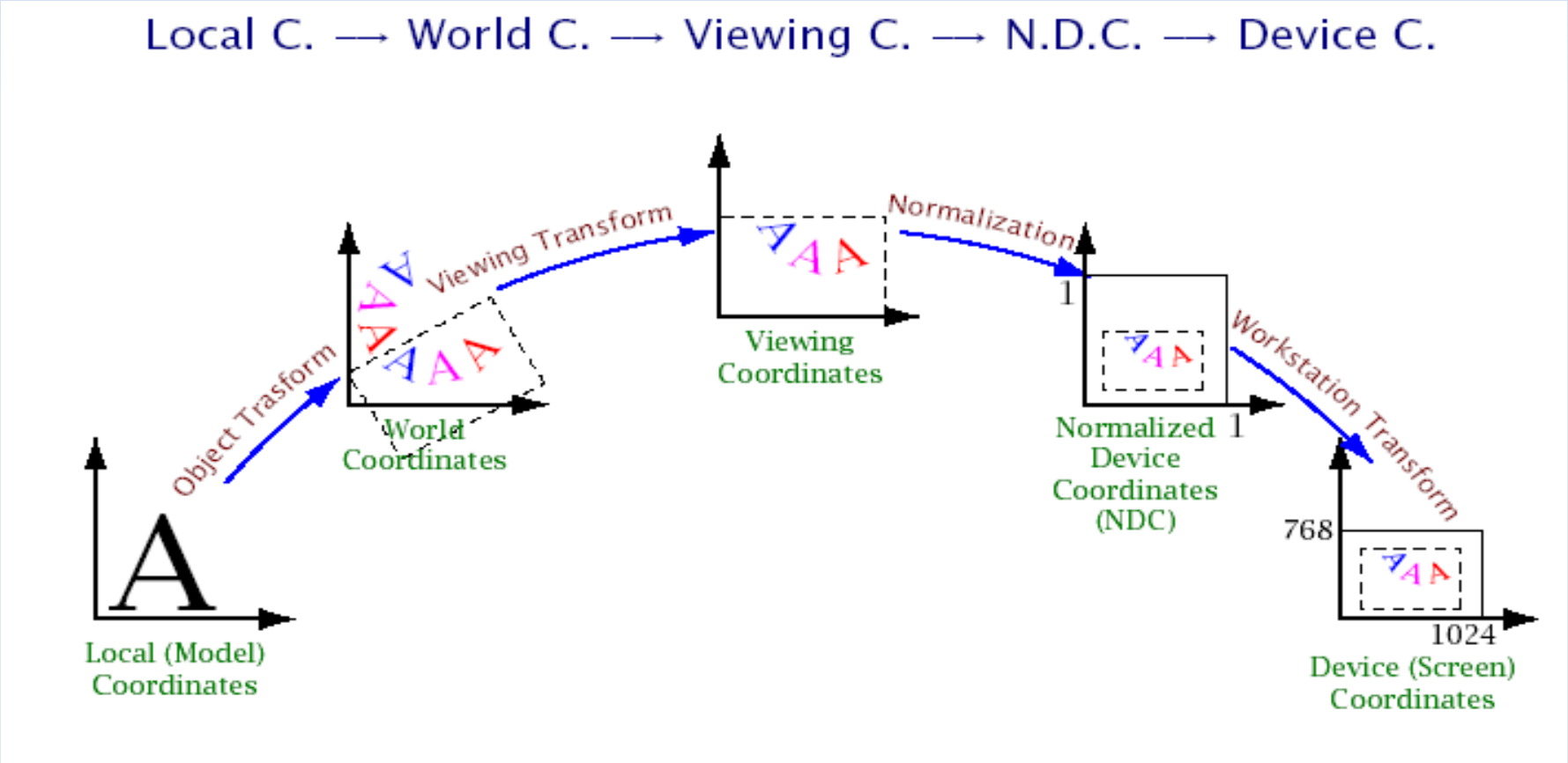
Transformaciones de Viewing 2D y 3D (Parte I)

Contenido

- 2D viewing pipeline
- Viewing en OpenGl
- Algoritmos de Clipping

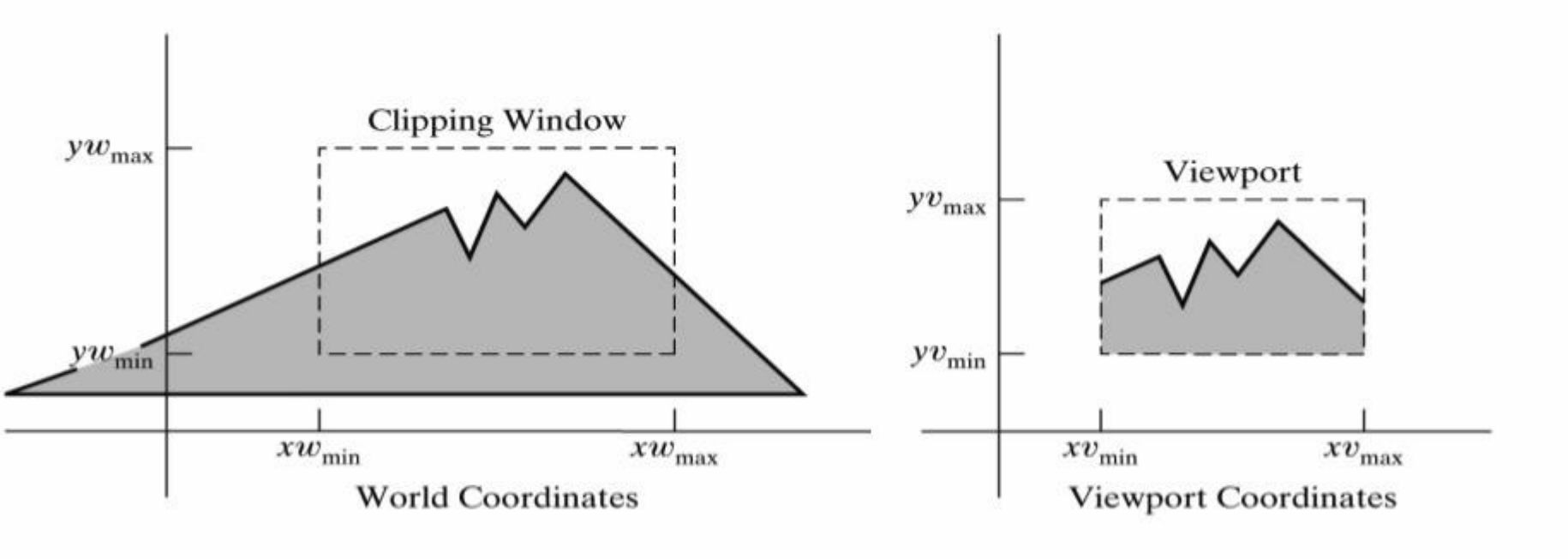
2D viewing pipeline

- Conceptos importantes:
  - Transformaciones de viewing 2D: mapeo from WC a DC



2D viewing pipeline

- Ventana de clipping y viewport



2D viewing pipeline

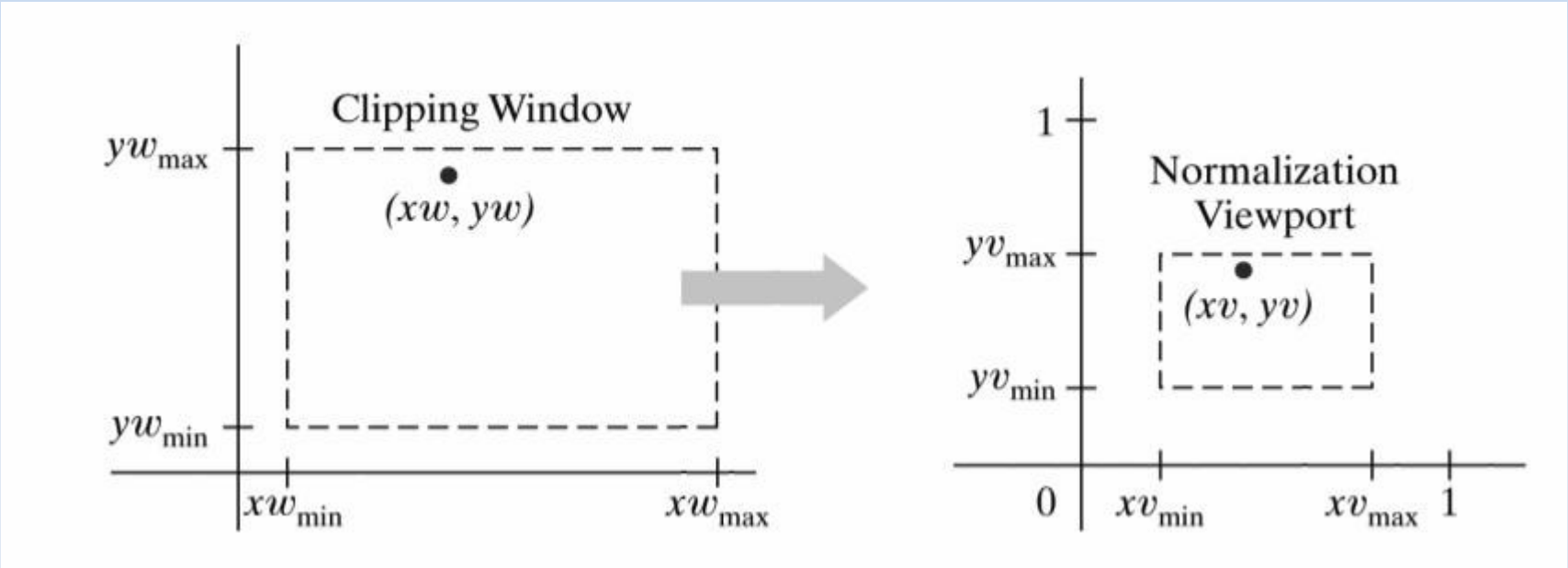
- Conceptos importantes:
  - Ventana de clipping (WC): selecciona las partes de la escena a mostrar en la pantalla (u otro dispositivo de salida)
  - Ventana Viewport (VC): especifica en qué parte de la ventana de display se mostrará la parte de la escena seleccionada en la ventana de clippling
    - Múltiples viewports pueden ser usados para mostrar mútiples vistas de la escena
    - Cambiando el tamaño de los viewports se puede mostrar los objetos de diferentes tamaños
  - Propuesto: Cómo lograr zoom in y zoom out?

2D viewing pipeline

- Ventana de Clipping:
  - Forma: polígono convexo, círculo, etc
    - Más lento de procesar
  - Forma rectangular: inclinada o no
- En general los paquetes gráficos solo se proveen ventanas de forma rectangular y con arcos paralelos a los eje x e y

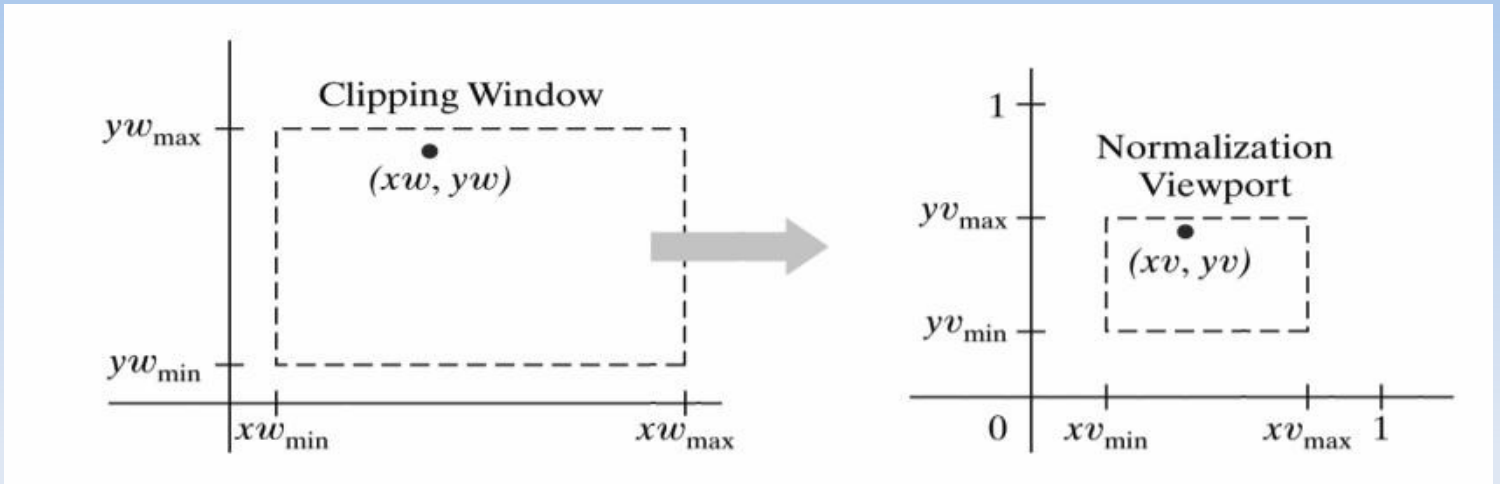
2D viewing pipeline

- Mapeo Ventana de clipping- ventana viewport normalizada



- Cómo calcular la transformación asociada?

2D viewing pipeline



$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Resolviendo la expresión anterior se obtiene:

$$xv = s_x\, xw + t_x \qquad yv = s_y\, yw + t_y$$

2D viewing pipeline

- Donde

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} \qquad s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

- y

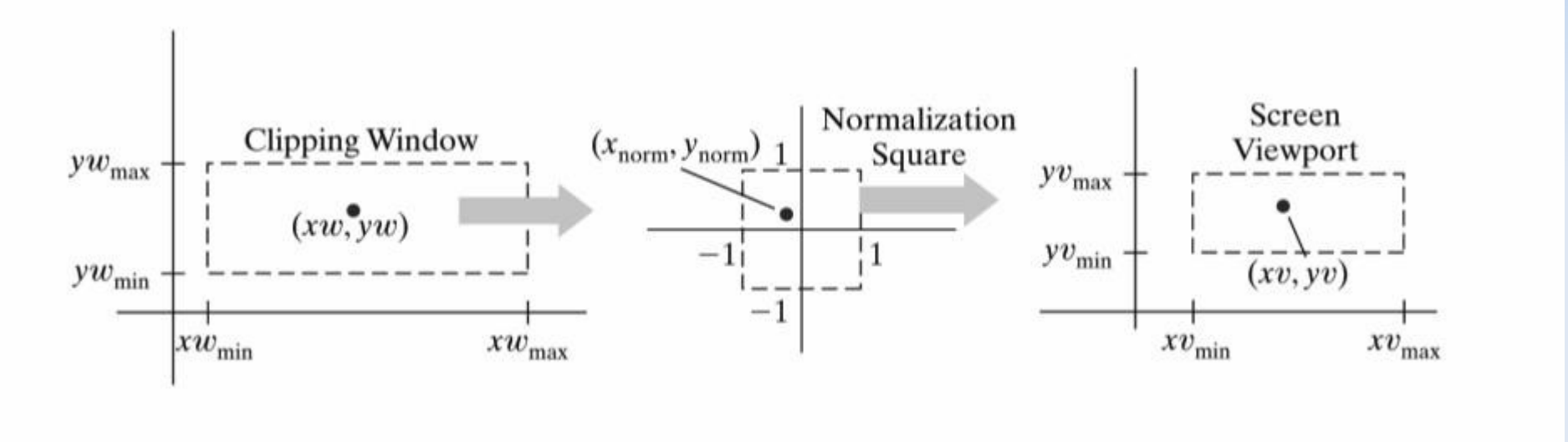
$$t_x = \frac{xw_{max}\, xv_{min} - xw_{min}\, xv_{max}}{xw_{max} - xw_{min}}$$

$$t_y = \frac{yw_{max}\, yv_{min} - yw_{min}\, yv_{max}}{yw_{max} - yw_{min}}$$

- Propuesto: obtener la transformación aplicando las transformaciones vistas las clases anteriores

2D viewing pipeline

- Otro forma de viewing pipeline es:



- Propuesto: Construir la matriz resultante.

OpenGL 2D viewing

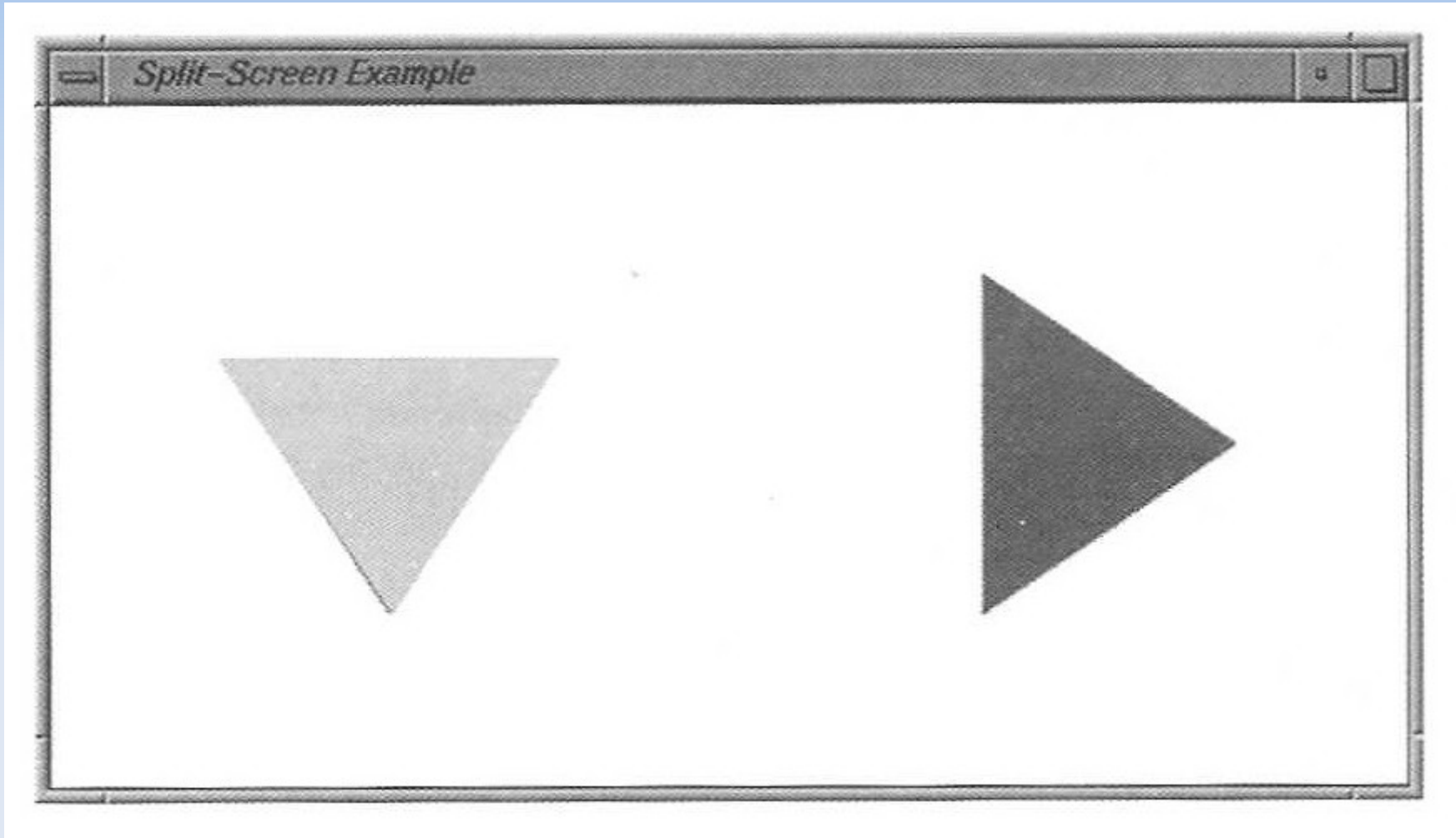
- En realidad, OpenGl no provee funciones especiales para 2D viewing pipeline
- Cómo se especifica la ventana de clipping?
  - glMatrixMode(GL_PROJECTION);
  - glLoadIdentity();
  - glOrtho2D(xwmin,xwmax, ywmin,ywmax);
- Si no se especifica una ventana de clipping, los valores usados son:

$$(xw_{min}, yw_{min}) = (-1.0, -1.0) \quad y \quad (xw_{max}, yw_{max}) = (1.0, 1.0)$$

OpenGL 2D viewing

- Cómo se especifica viewport?
  - glViewport(xvmin,xvmax,vpWidth,vpHeight);
  - Los parámetros son coordenadas enteras
  - (xvmin,yvmin) esquina inferior izquierda relativa a la esquina interior izquierda de la ventana de display
- Cómo crear múltiples vistas?
  - glGetIntegerv(GL_VIEWPORT, vparray);
  - Obtiene los parámetros del viewport actual

Ejemplo en OpenGl



Visualización de un triángulo en dos viewports

```
void main (int argc, char ** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (50, 50);
    glutInitWindowSize (600, 300);
    glutCreateWindow («Split-Screen Example»);

    init ( );
    glutDisplayFunc (displayFcn);

    glutMainLoop ( );
}
```

Inicialización

```
#include <GL/glut.h>

class wcPt2D {
   public:
      GLfloat x, y;
}

void init (void)
{
   /*  Establece el color de la ventana visualización en blanco.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);

   /*  Establece los parámetros de la ventana de recorte en
    *  coordenadas universales.  */
   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (-100.0, 100.0, -100.0, 100.0);

   /*  Establece el modo de construcción de la matriz de
    *  transformación geométrica.  */
   glMatrixMode (GL_MODELVIEW);
}
```

Especificación del triángulo

```
void triangle (wcPt2D *verts)
{
    GLint k;

    glBegin (GL_TRIANGLES);
        for (k = 0; k < 3; k++)
            glVertex2f (verts [k].x, verts [k].y);
    glEnd ( );
}
```

Función que dibuja

```
void displayFcn (void)
{
   /*  Define la posición inicial del triángulo.  */
   wcPt2D verts [3] = { {-50.0, -25.0}, {50.0, -25.0}, {0.0, 50.0} };

   glClear (GL_COLOR_BUFFER_BIT);    //  Borra la ventana de visualización.

   glColor3f (0.0, 0.0, 1.0);        //  Establece el color de relleno en azul.
   glViewport (0, 0, 300, 300);      //  Establece el visor izquierdo.
   triangle (verts);                 //  Muestra el triángulo.

   /*  Gira el triángulo y lo visualiza en la mitad derecha de la
    * ventana de visualización.  */

   glColor3f (1.0, 0.0, 0.0);        //  Establece el color de relleno en rojo.
   glViewport (300, 0, 300, 300);    //  Establece el visor derecho.
   glRotatef (90.0, 0.0, 0.0, 1.0);  //  Gira alrededor del eje z.
   triangle (verts);                 //  Muestra el triángulo rojo girado.

   glFlush ( );
}
```

Algoritmos de clipping

- Cómo eliminar las porciones de la escena que estan fuera?
  - Clipping de puntos
  - Clipping de líneas
  - Clipping de polígonos
  - Clipping de texto
  - Clipping de curvas

Clipping de un punto

- Ventana de clipping definida por:

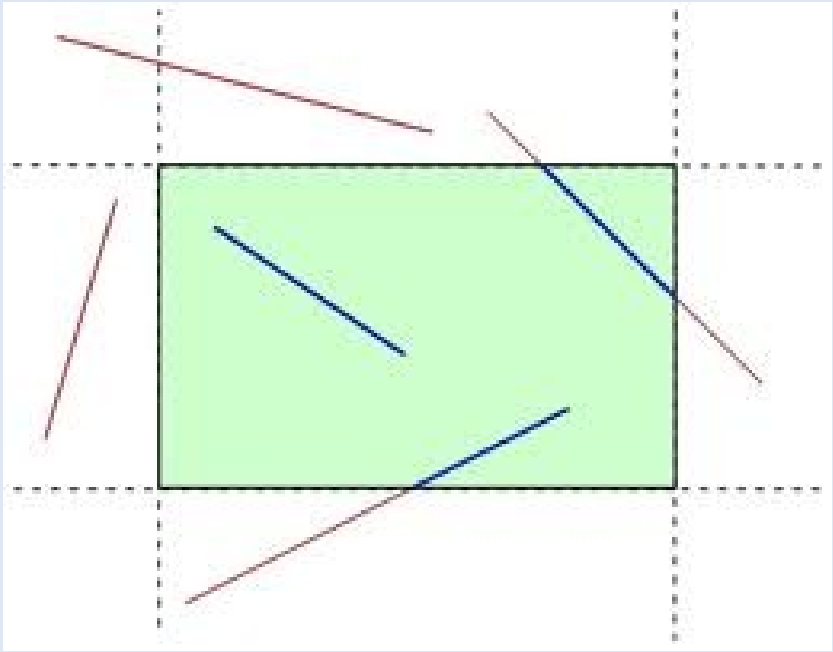
$$(xw_{min}, yw_{min}) \quad y \quad (xw_{max}, yw_{max})$$

- Clipping de un punto: $P(x, y)$

$$xw_{min} \leqslant x \leqslant xw_{max}$$
$$yw_{min} \leqslant y \leqslant yw_{max}$$

Clipping de una línea

- Cómo chequear si un segmento está dentro, fuera o intersecta el borde de la ventana?



Algoritmo Cohen-Sutherland

- Ideas:
  - Cada punto extremo de un segmento se clasifica con un número binario de 4 bits
    - Bit 1: Left (lado izquierdo de la ventana)
    - Bit 2: Right (lado derecho de la ventana)
    - Bit 3: Bottom (bajo la ventana)
    - Bit 4: Top (arriba de la ventana)
  - Si un punto extremo está fuera de la ventana se asigna al (o los) bit(s) correspondiente(s) un 1

Algoritmo Cohen-Sutherland

- Cómo clasificar P(x,y)?
- Bit 1: bit de signo de

$$x - xw_{min}$$

- Bit 2: bit de signo de

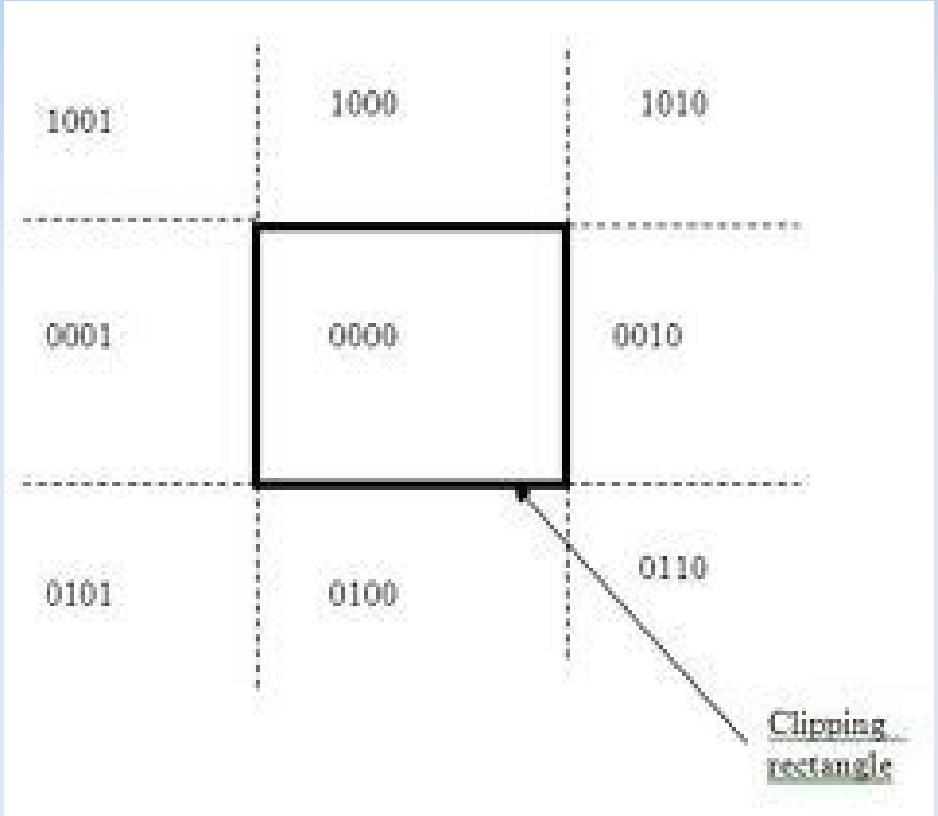
$$xw_{max} - x$$

- Bit 3: bit de signo de

$$y - yw_{min}$$

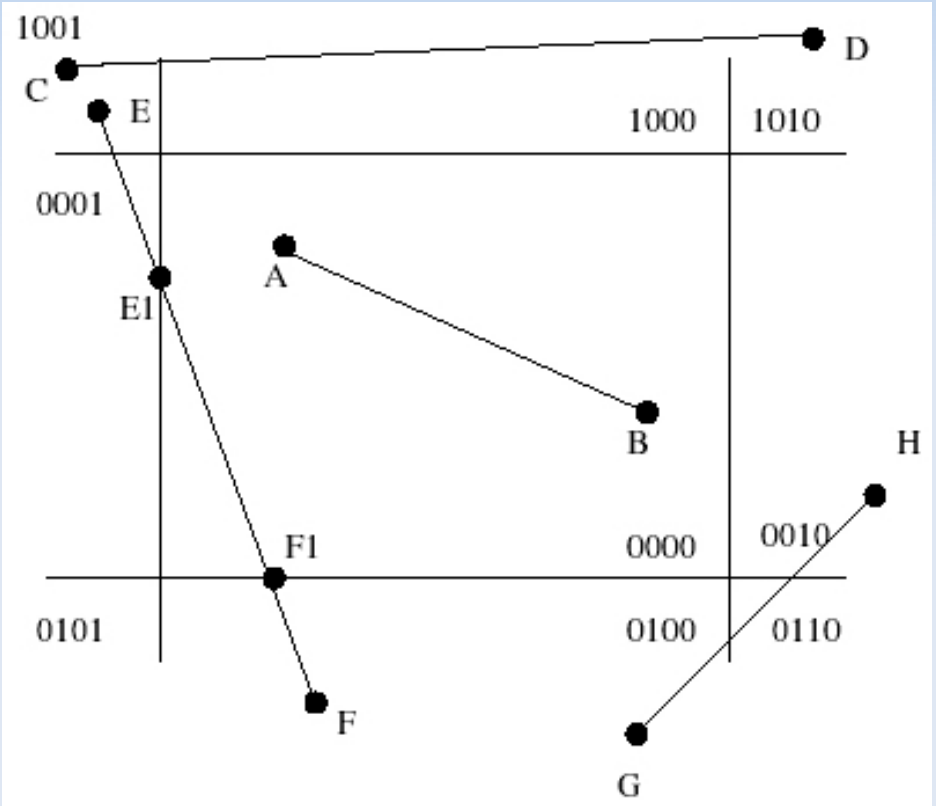
- Bit 4: bit de signo de

$$yw_{max} - y$$



Algoritmo Cohen-Sutherland

- Uso de operaciones lógicas la secuencia de bits de cada punto extremo para decidir:
  - If la operación OR resulta 0 => segmento “in”
  - elseif la operación AND resulta != 0 => segmento “out”
  - else calcular intersección



Algoritmo Cohen-Sutherland

- Variaciones para el cálculo de las intersecciones:

$$P_0(x_{0,}y_0), P_{end}(x_{end}, y_{end})$$

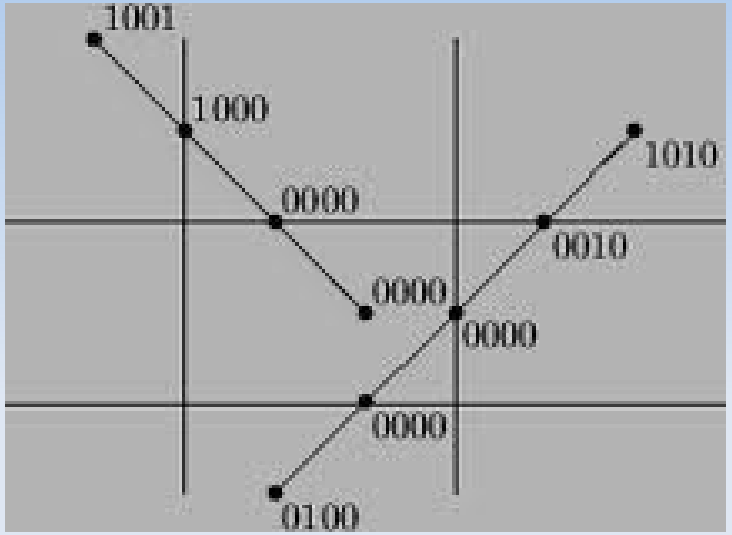
  - Usando representación con la pendiente del segmento

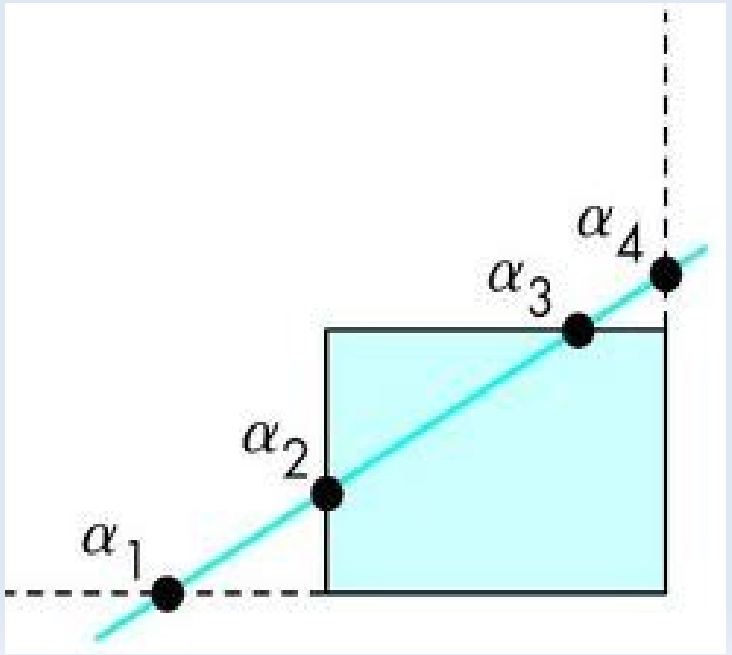
$$y = mx + b$$

  - Usando la representación paramétrica (Liang-Barsky)

$$x = x_0 + u(x_{end} - x_0) \qquad y = y_0 + u(y_{end} - y_0)$$

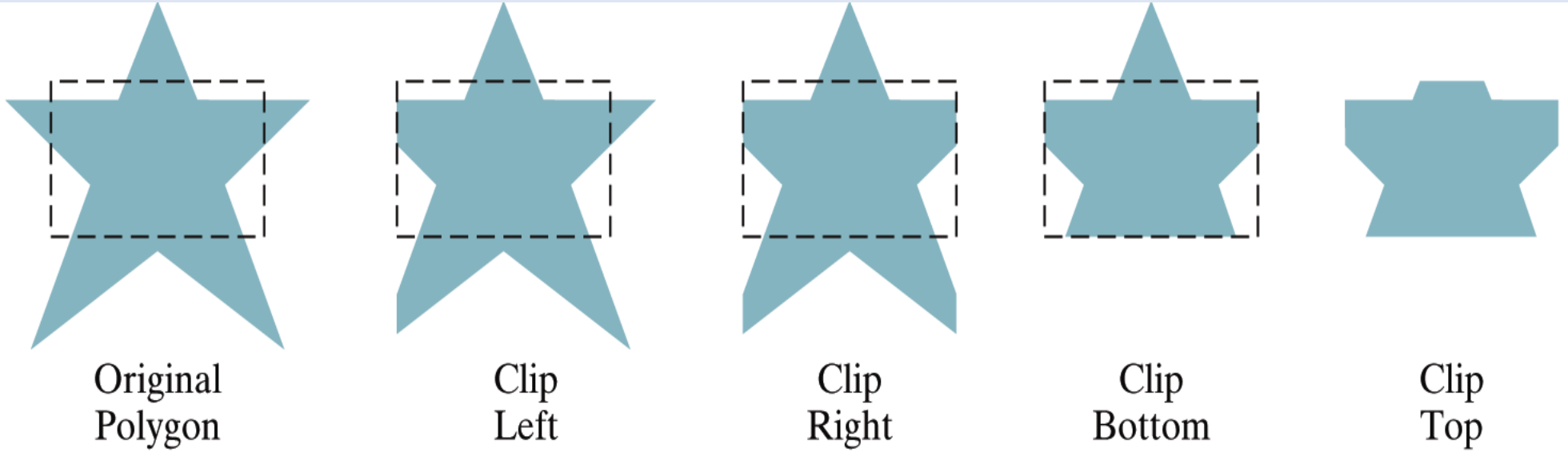
$$0 \leqslant u \leqslant 1$$





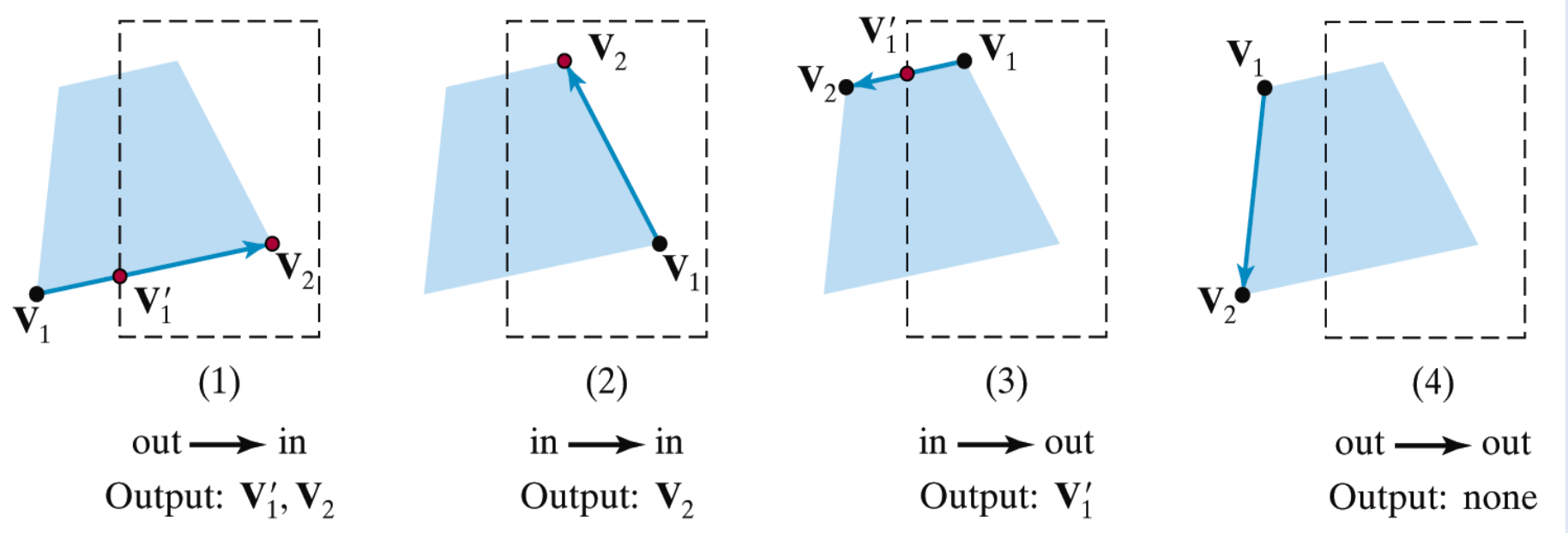
Clipping polígonos

- Sutherland-Hodgeman Algorithm
  - Recorta el polígono contra los cuatro arcos



Clipping polígonos

- Sutherland-Hodgeman Algorithm
  - Sobre polígonos convexos
  - Input: vértices del polígonos. Output: vértices del polígono recortado
  - Se analizan 4 casos



Recorte de texto

- Usar bounding-box que rodea el texto
- Usar bounding-box de cada caracter
- Recortar cada caracter si es necesario