Graficación

Dr. Luis Gerardo de la Fraga

E-mail: fraga@cs.cinvestav.mx
Departamento de Computación
Cinvestav

octubre, 2011

Contenido. Parte 1

Motivación

Es muy importarte conocer cómo se dibujan formas geométricas en el plano, tanto para diseñar nuevas aplicaciones como para escoger cuál es la aplicación/tecnología más eficiente para el dibujado de escenas bidimensionales.

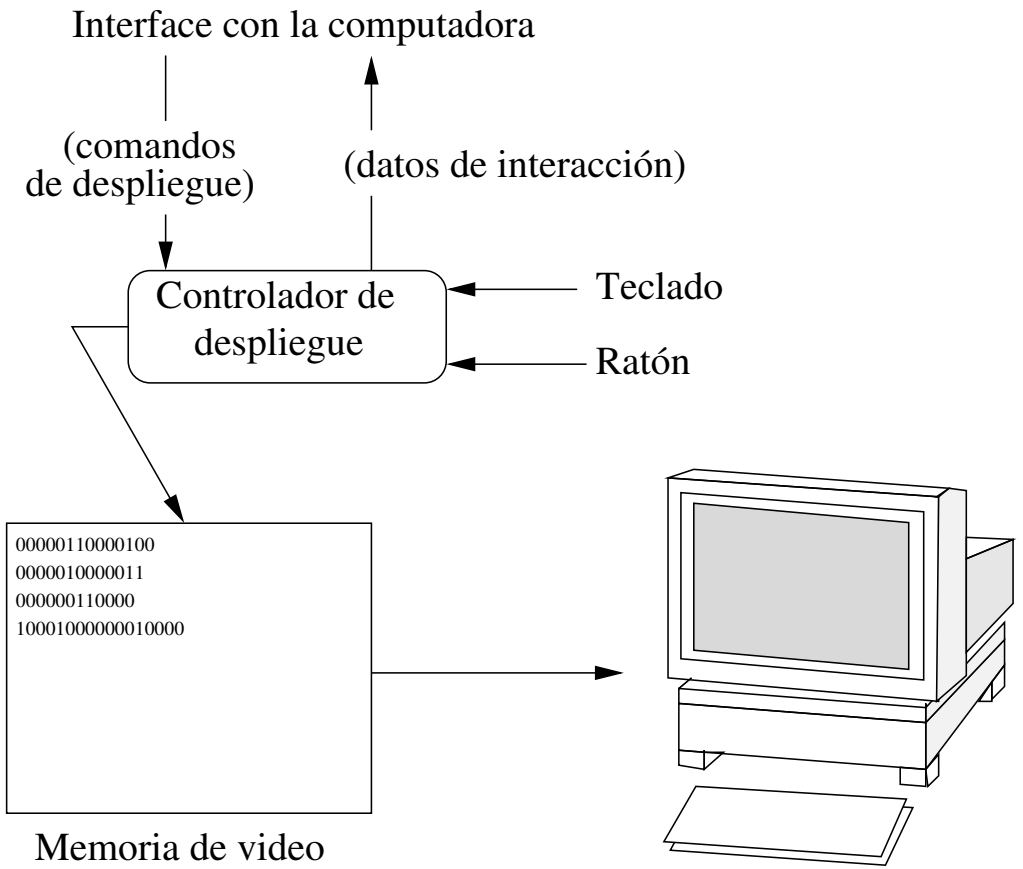
Introducción (1/2)

Áreas relacionadas

1. **Graficación**: Trata la síntesis de escenas con objetos reales o imaginarios a partir de sus modelos computacionales.
2. **Procesamiento de Imagen**: Trata el análisis de escenas a partir de sus fotografías.
   - 2.1 **Realzado de imagen**
   - 2.2 **Segmentación de formas**
   - 2.3 **Reconocimiento de patrones**
3. **Visión por computadora**: Reconstrucción de un modelo 3D de una escena a partir de varias imágenes 2D.

Foley, van Dam, Feiner, Hughes, Computer Graphics: principles and practice, 2000, Addison Wesley.

Arquitectura de trabajo



Alternativas actuales para visualización

- Usar las primitivas de dibujo que ofrecen paquetes (Qt).
  Ventaja: fácil de usar.
  Desventaja: no adecuado para gráficos complejos. Podría usar todo el CPU.
- Usar las primitivas en el procesador gráfico (OpenGL)
  Ventaja: lo mejor para visualización iteractiva
  Desventaja: no ofrece los mejores efectos posibles en la visualización
- Precalcular todas las vistas a visualizar
  Ventaja: lo mejor si se visualizará una misma encena desde varios ángulos. Se puede usar trazo de rayo.
  Desventaja: Configurar la base de datos de imágenes

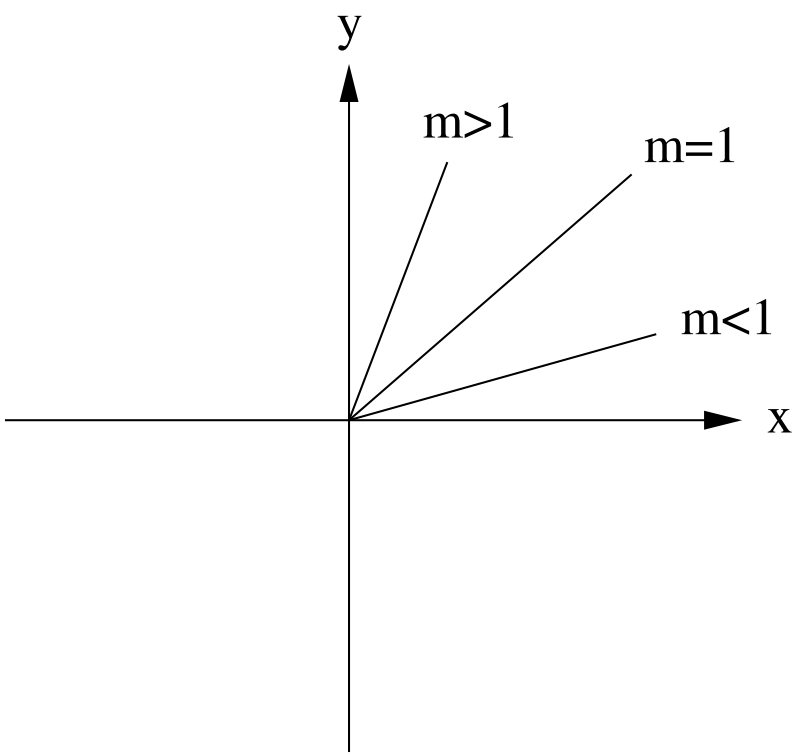
Dibujado en 2D

Vamos a conocer qué tan dificil es realizar dibujos en 2D conociendo como se realizan las siguientes primitivas:

1. Trazo de una línea
2. Trazo de líneas
3. Trazo de círculos
4. Rellenado de polígonos
5. Primitivas gruesas

Algoritmo 1 para trazar una línea (1/2)

- Para el algoritmo básico que vamos a desarrollar se considerará que se trazará una línea en el primer octante, donde su pendiente es menor a 1.
- Las otras líneas pueden trazarse usando simetrías sobre el algoritmo básico.
- No se puede usar como un algoritmo básico trazar una línea con pendiente mayor a 1, debido a que quedan huecos ($m = dy/dx$).



Algoritmo 1 para trazar una línea (2/2)

Se trazará una línea del punto (x_1, y_1) al punto (x_2, y_2)

```
double m, y;
m = (y2 − y1)/(x2 − x1);
for x = x1 : x2 do
    y = (x − x1)m + y1;
    drawPixel( x, Round(y) );
end for
```

Esto representa 4 operaciones en punto flotante, para calcular cada posición de cada píxel de la línea.

Algoritmo 2

¿Se puede hacer un mejor algoritmo?
Si, usando un algoritmo incremental. Un punto se calcula como:

$$y_i = x_i m + B$$

El siguiente punto será:

$$y_{i+1} = x_{i+1} m + B,$$
$$y_{i+1} = (x_i + \Delta x) m + B,$$

pero como $\Delta x = 1$, tenemos:

$$y_{i+1} = (x_i + 1) m + B,$$
$$y_{i+1} = x_i m + B + m,$$
$$y_{i+1} = y_i + m.$$

Algoritmo 2

Se trazará una línea del punto (x_1, y_1) al punto (x_2, y_2)

```
double m, y;
m = (double)(y2 − y1)/(double)(x2 − x1);
y = y1;
for x = x1 : x2 do
    drawPixel( x, Round(y) );
    y+ = m;
end for
```

Este algoritmo requiere dos operaciones en punto flotante para calcular cada posición de un pixel. Dado que una variable real tiene precisión limitada, sumar un valor inexacto de m introducirá un error acumulado para trazar lineas muy largas.

Ecuación implícita de la línea

$$y = \frac{dy}{dx}x + B$$

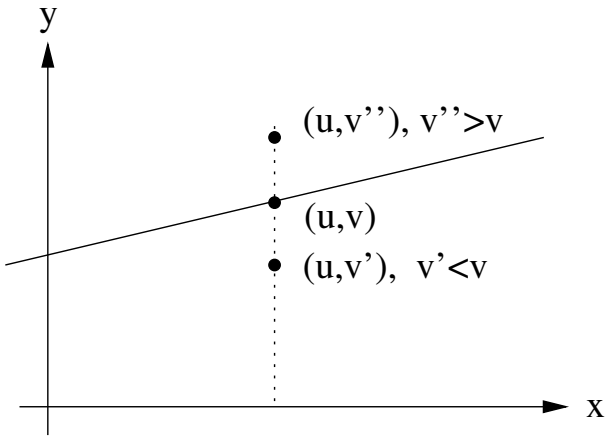
La ecuación anterior se puede cambiar a una forma implícita de la forma:

$$dx \cdot y = dy \cdot x + dx \cdot B,$$

$$f(x, y) = dy \cdot x - dx \cdot y + dx \cdot B = 0.$$

Quedando de la forma

$$f(x, y) = ax + by + c = 0.$$

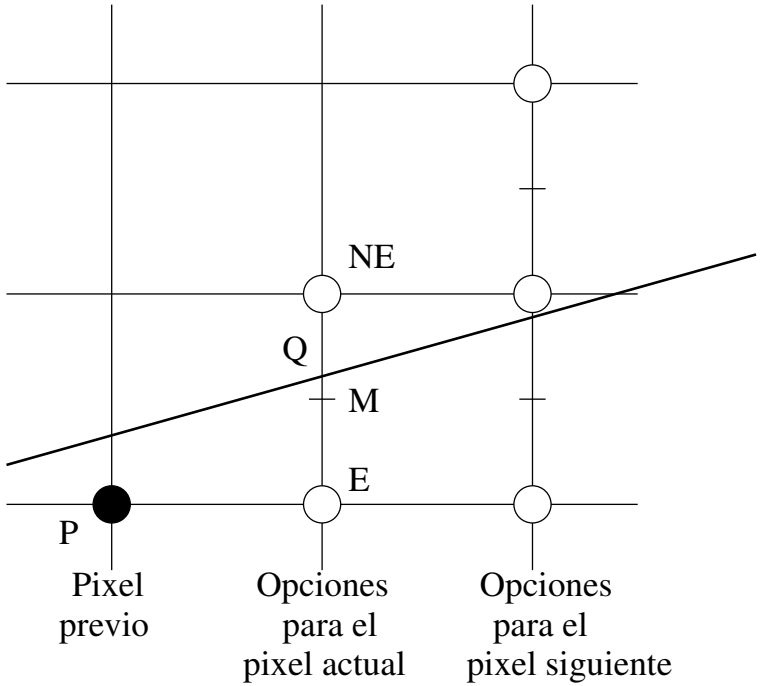


- En el punto (u, v), $f(u, v) = 0$
- En el punto (u, v'), $f(u, v') > 0$
- En el punto (u, v''), $f(u, v'') < 0$

Algoritmo del punto medio (1/2)

Consideremos que se ha seleccionado el pixel $P = (x_p, y_p)$. El pixel siguiente que se escogerá depende de la evaluación de la función f en el siguiente punto medio $M = (x_p + 1, y_p + \frac{1}{2})$.

- Si $f(M) > 0$, se escogerá el pixel NE.
- Si $f(M) < 0$, se escogerá el pixel E.



Algoritmo del punto medio (2/2)

Se hará un algoritmo incremental sobre la variable de decisión $d = f(M)$.
Si ha escogido el punto E, M se incrementa un paso en la dirección x. Entonces,

$$d_{\text{act}} = f\left(x_p + 2, y_p + \frac{1}{2}\right),$$
$$= a(x_p + 2) + b\left(y_p + \frac{1}{2}\right) + c,$$

pero

$$d_{\text{ant}} = a(x_p+1)+b\left(y_p + \frac{1}{2}\right)+c.$$

Y calculando $d_{\text{act}} - d_{\text{ant}}$ para obtener la diferecia incremental, se obtiene $\Delta_E = a = dy$.

Si se escoge el pixel NE, el siguiente punto a evaluar es

$$d_{\text{act}} = f\left(x_p + 2, y_p + \frac{3}{2}\right),$$
$$= a(x_p + 2) + b\left(y_p + \frac{3}{2}\right) + c.$$

Substrayendo d_{ant} de d_{act} obtenemos

$$d_{\text{act}} - d_{\text{ant}} = \Delta_{NE} = a+b = dy-dx;$$

Punto de arranque

El algoritmo se inicia en el punto $P = (x_0, y_0)$, el primer punto medio es $(x_0 + 1, y_0 + \frac{1}{2})$, la variable de decisión tendrá el valor inicial:

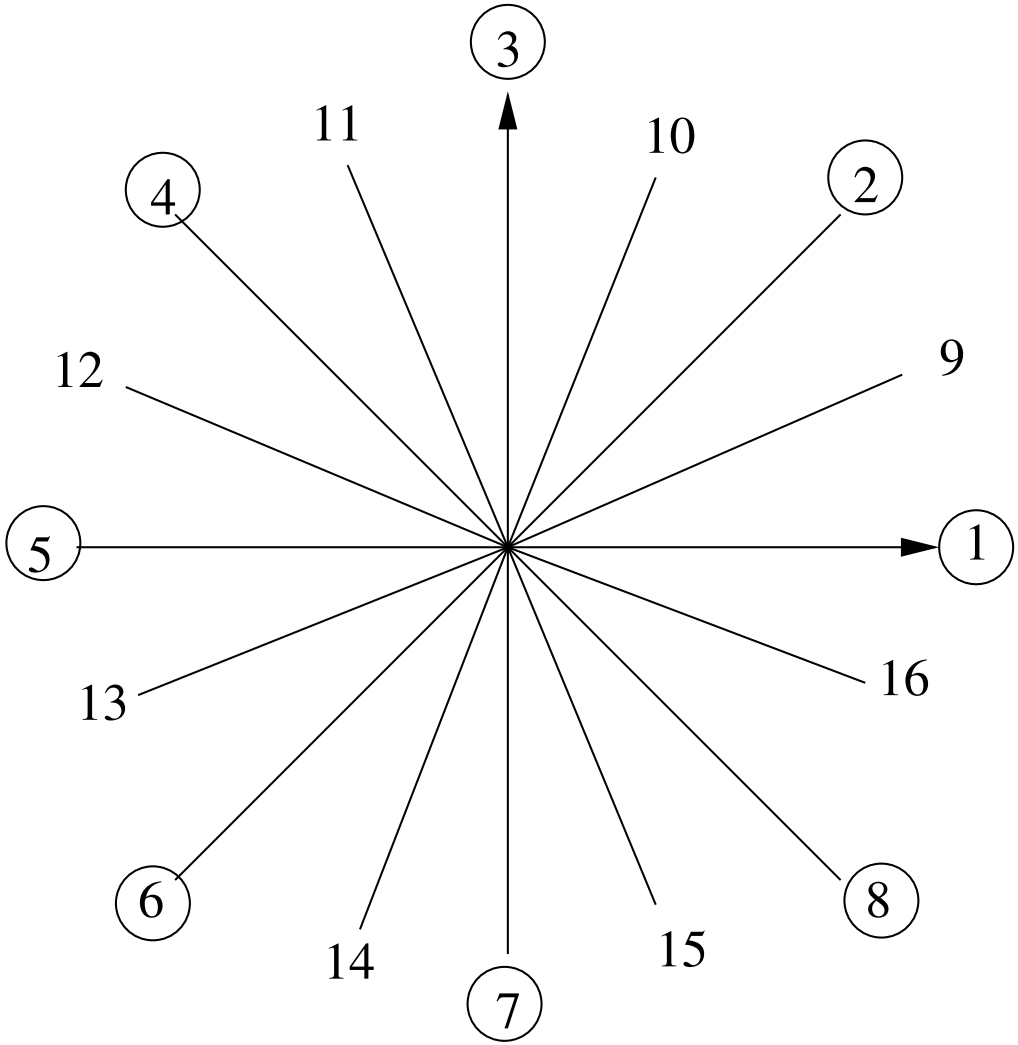
$$\begin{aligned} f\left(x_0 + 1, y_0 + \frac{1}{2}\right) &= a(x_0 + 1) + b\left(y_0 + \frac{1}{2}\right) + c \\ &= ax_0 + by_0 + c + a + b/2 \\ &= f(x_0, y_0) + a + b/2. \end{aligned}$$

Pero $f(x_0, y_0) = 0$, por lo tanto $d_{\text{inicial}} = dy - dx/2$

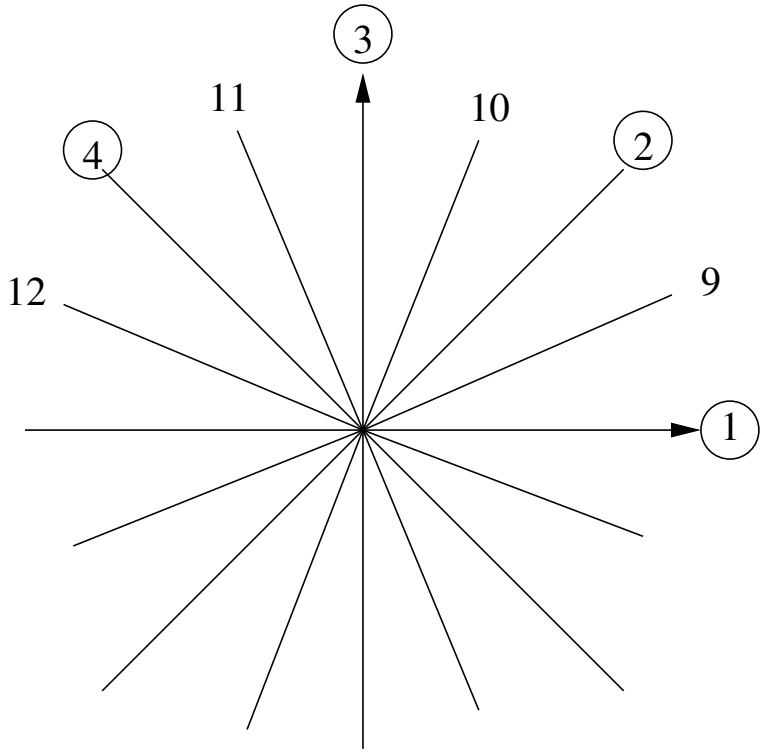
Función para trazar una línea

```
lineaPuntoMedio(int x1, int y1, int x2, int y2, int valor )
int dx = x2-x1;
int dy = y2-y1;
int d = 2*dy - dx;
int incrE = 2*dy;
int incrNE = 2*(dy-dx);
int x=x1, y=y1;
WritePixel( x, y, valor);
while ( x < x2 ) do
   if  d <=0  then
      d + = incrE;
   else
      d + = incrNE;
      y++;
   end if
   x++;
   WritePixel( x, y, valor);
end while
```

Algoritmo general para el trazo de líneas (1/2)



Algoritmo general para el trazo de líneas (2/2)

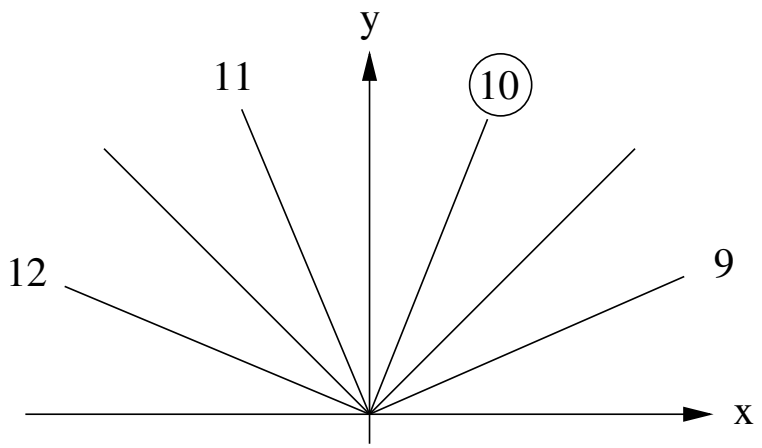


Se pueden reducir el número de casos a la mitad usando simetrías. Esto funcionará si no se utiliza el algoritmo para animación.

¿Cómo detectar cada caso?

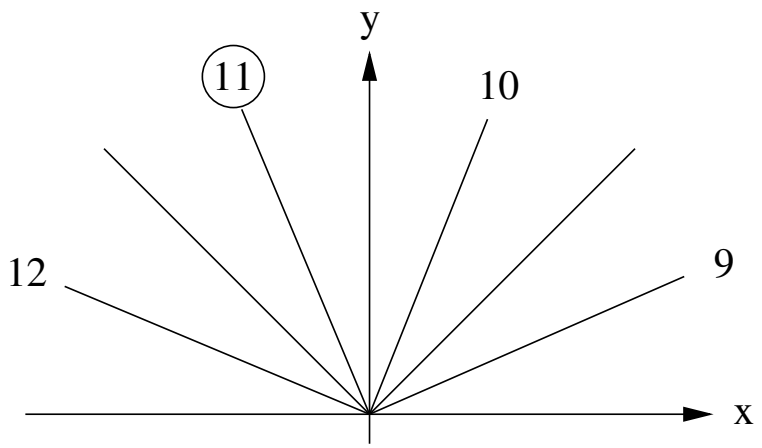
Caso	Condición
0	$dx = 0,\ dy = 0$
1	$dy = 0,\ dx > 0$
2	$dx = dy,\ dx > 0$
3	$dx = 0,\ dy > 0$
4	$dy = \lvert dx \rvert,\ dx < 0$
9	$dx > 0,\ dy > 0,\ dy < dx$
10	$dx > 0,\ dy > 0,\ dx < dy$
11	$dx < 0,\ dy > 0,\ \lvert dx \rvert < dy$
12	$dx < 0,\ dy > 0,\ dy < \lvert dx \rvert$

Caso 10




```
lineaPuntoMedio(int x1, int y1, int
x2, int y2, int valor )
int dx = y2-y1;
int dy = x2-x1;
int d = 2*dy - dx;
int incrE = 2*dy;
int incrNE = 2*(dy-dx);
int x=x1, y=y1;
WritePixel( x, y, valor);
while ( y < y2 ) do
   if  d <=0  then
      d + = incrE;
   else
      d + = incrNE;
      x++;
   end if
   y++;
   WritePixel( x, y, valor);
end while
```

Caso 11




```
lineaPuntoMedio(int x1, int y1, int
x2, int y2, int valor )
int dx = y2-y1;
int dy = abs(x2-x1);
int d = 2*dy - dx;
int incrE = 2*dy;
int incrNE = 2*(dy-dx);
int x=x1, y=y1;
WritePixel( x, y, valor);
while ( y < y2 ) do
   if  d <=0  then
      d + = incrE;
   else
      d + = incrNE;
      x−−;
   end if
   y++;
   WritePixel( x, y, valor);
end while
```

```
Require: (x1, y1), (x2, y2),
  valor pixel
Ensure: la línea
  if y1 > y2 then
    swap( (x1, y1), (x2, y2) );
  end if
  dx = x2 − x1;
  dy = y2 − y1;
  if dy == 0 then
    if dx == 0 then
      return {Caso 0}
    else if x1 > x2 then
      Caso 5!
    else
      Caso 1
    end if
  end if
  if dy == 0 then
    Caso 3
  end if
  if abs(dx) == dy then
    if dx > 0 then
      Caso 2
    else
      Caso 4
    end if
  end if
  if abs(dy) < dx then
    if dx > 0 then
      Caso 9
    else
      Caso 12
    end if
  else
    if dx > 0 then
      Caso 10
    else
      Caso 11
    end if
  end if
```

Trazo de círculos

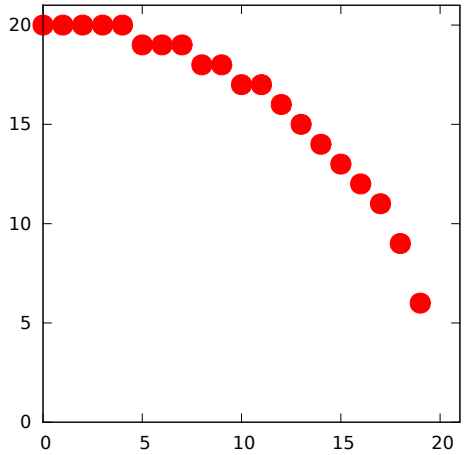
Trazo de círculos

Vamos a realizar una primitiva para trazar círculos.

No podemos trazar un círculo usando la ecuación

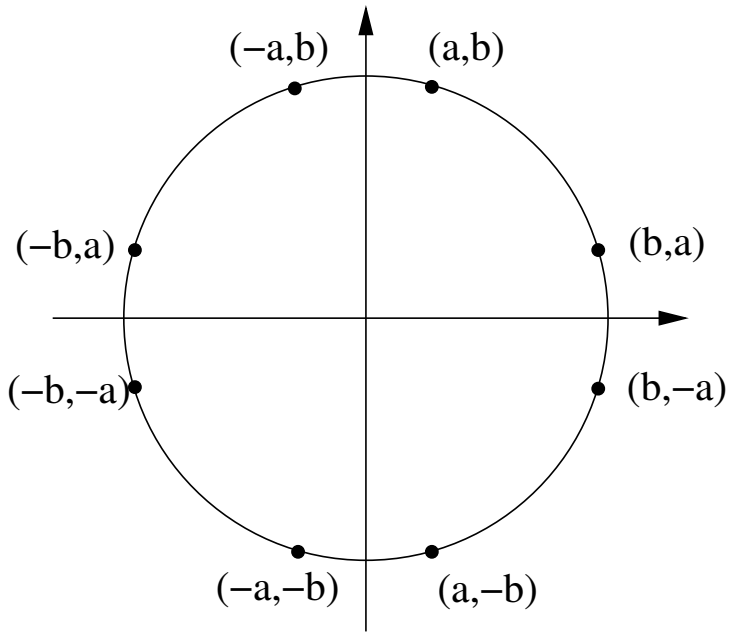
$$y = \pm\sqrt{r^2 - x^2},$$

debido a que quedan huecos donde la pendiente es mayor a 1.



Cada punto en el segundo octante puede replicarse por la simetría del círculo

```
PuntosCirculo( int c, int y, int valor)
do
    WritePixel(y, x, valor);
    WritePixel(x, y, valor);
    WritePixel(-x, y, valor);
    WritePixel(-y, x, valor);
    WritePixel(-y, -x, valor);
    WritePixel(-x, -y, valor);
    WritePixel( x, -y, valor);
    WritePixel( y, -x, valor);
```

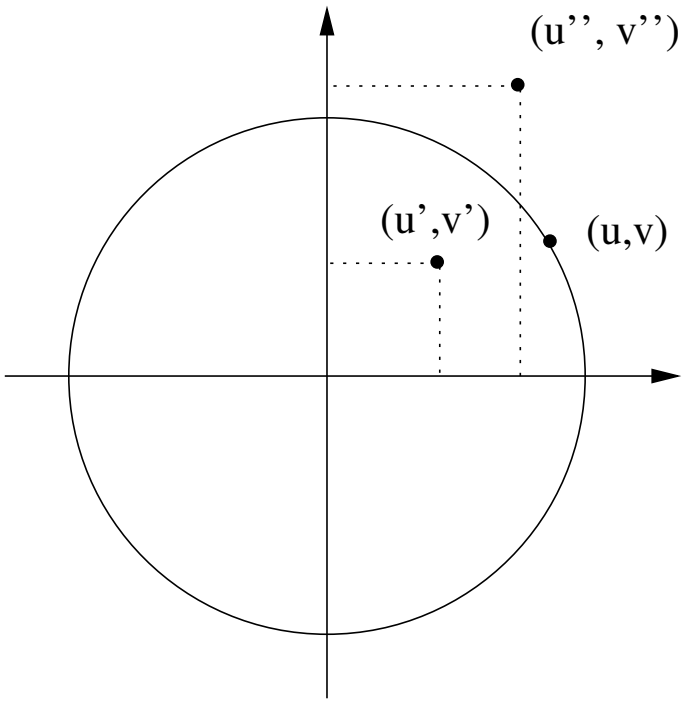


Ecuación implícita del círculo

$$x^2 + y^2 = r^2,$$

Arreglando la ec. anterior de la forma

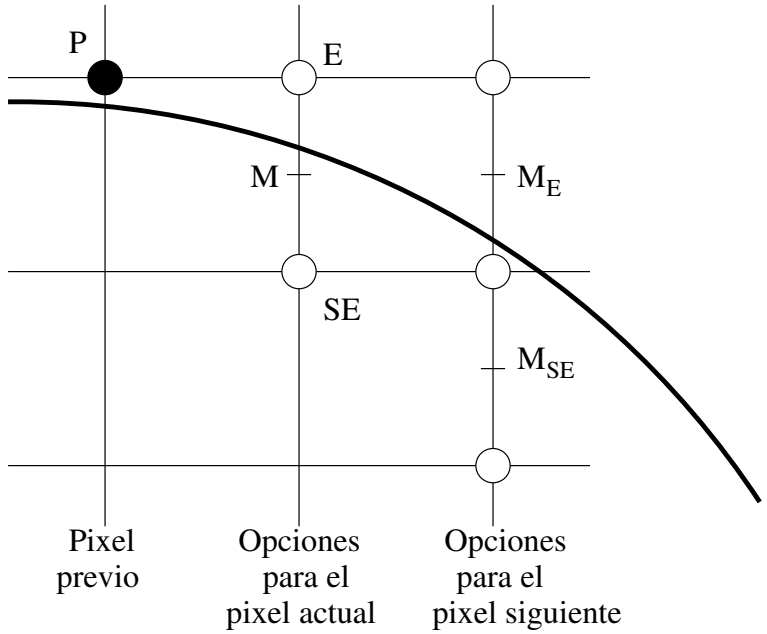
$$f(x, y) = x^2 + y^2 - r^2 = 0$$



- En el punto (u, v), $f(u, v) = 0$
- En el punto (u', v'), $f(u', v') < 0$
- En el punto (u'', v''), $f(u'', v'') > 0$

Si se ha seleccionado el píxel $P = (x_p, y_p)$, el siguiente píxel que se selecionará dependerá de la evaluación $f(M)$

- Si $f(M) < 0$, se escogerá el píxel E.
- Si $f(M) > 0$, se escogerá el píxel SE.



Algoritmo del punto medio (1/3)

El algoritmo incremental se base en la variable de decisión $d = f(M)$, que resulta en:

$$d_{\text{ant}} = f(M) = f\left(x_p + 1, y_p - \frac{1}{2}\right)$$

$$= (x_p + 1)^2 + \left(y_p - \frac{1}{2}\right)^2 - r^2.$$

Si $d_{\text{ant}} < 0$, se escoge E, y el siguiente punto medio es M_E, entonces

$$d_{\text{act}} = f(M_E) = f\left(x_p + 2, y_p - \frac{1}{2}\right)$$

$$= (x_p + 2)^2 + \left(y_p - \frac{1}{2}\right)^2 - r^2,$$

y $d_{\text{act}} - d_{\text{ant}} = \Delta_E = 2x_p + 3$.

Si $d_{\text{ant}} > 0$, se escoge SE, y el siguiente punto medio es M_{SE}, por lo que

$$d_{\text{act}} = f(M_{SE}) = f\left(x_p + 2, y_p - \frac{3}{2}\right) =$$

$$= (x_p + 2)^2 + \left(y_p - \frac{3}{2}\right)^2 - r^2,$$

y $d_{\text{act}} - d_{\text{ant}} = \Delta_{SE} = 2x_p - 2y_p + 5$.

Algoritmo del punto medio (2/3)

Para iniciar el algoritmo, el primer punto es $(0, r)$, y el primer punto medio está en $(1, r - \frac{1}{2})$, por lo que el valor inicial de la variable de decisión será:

$$
\begin{aligned}
d &= f\left(1, r - \frac{1}{2}\right) = \\
&= 1 + \left(r - \frac{1}{2}\right)^2 - r^2 \\
&= 1 + \left(r^2 - r + \frac{1}{4}\right) - r^2 \\
&= \frac{5}{4} - r
\end{aligned}
$$

Algoritmo del punto medio (3/3)

```
int x = 0;
int y = radio;
double d = 5.0/4.0 - radio;
PuntosCirculo( x, y, valor );
while  y > x  do
   if  d < 0  then
      d += 2.0 * x + 3.0;
   else
      d += 2.0 *(x-y) + 5.0;
      y --;
   end if
   x ++;
   PuntosCirculo( x, y, valor );
end while
```

Sin variables reales

Si hacemos el cambio de variable, $h = d - \frac{1}{4}$, entonces

$$d = h + \frac{1}{4}.$$

Y la variable de decisición inicial será

$$\begin{aligned} d &= \frac{5}{4} - r, \\ h + \frac{1}{4} &= \frac{5}{4} - r, \\ h &= \frac{5}{4} - \frac{1}{4} - r, \\ h &= 1 - r; \end{aligned}$$

Consideración

En el algoritmo anterior, d se inicializa como un número real, y todos los incrementos son en enteros, por lo que nunca la variable de decisición será igual a 0.

Pero ahora, con h, se inicializa a un número entero, con incrementos enteros, por lo que puede ser igual a cero. Si $h = 0$, entonces

$$d = \frac{1}{4},$$

$d > 0$ y debe entonces escogerse el píxel SE.

El algoritmo finalmente queda (debería ser *h*, pero lo dejamos como *d* el nombre de la variable de decisión):

```
int x = 0;
int y = radio;
int d = 1 - radio;
PuntosCirculo( x, y, valor );
while y > x do
  if d < 0 then
    d += 2 * x + 3;
  else { d == 0 se escoge SE. Ok! }
    d += 2 *(x-y) + 5;
    y --;
  end if
  x ++;
  PuntosCirculo( x, y, valor );
end while
```

Diferencias de segundo orden (1/3)

Se puede mejorar el rendimiento del algoritmo del punto medio del círculo usando la técnica incremental de forma más extensiva.
Si se escoge el píxel E en la interación actual, el punto de evaluación se mueve de (x_p, y_p) a (x_p+1, y_p). Como ya vimos, la diferencia de primer order es $\Delta_{E_{\text{ant}}}$ en $(x_p, y_p) = 2x_p+3$. Por lo tanto, $\Delta_{E_{\text{act}}}$ en $(x_p + 1, y_p)$ es

$$\begin{aligned}\Delta_{E_{\text{act}}} &= 2(x_p + 1) + 3 \\ &= 2x_p + 5\end{aligned}$$

y la diferencia de segundo orden es

$$\Delta_{E_{\text{act}}} - \Delta_{E_{\text{ant}}} = 2$$

De forma similar, $\Delta_{SE_{\text{ant}}}$ en (x_p, y_p) es $2x_p - 2y_p + 5$, por lo que $\Delta_{SE_{\text{act}}}$ en $(x_p + 1, y_p)$ es

$$\begin{aligned}&= 2(x_p + 1) - 2y_p + 5 = \\ &= 2x_p - 2y_p + 7,\end{aligned}$$

y la diferencia de segundo orden es

$$\Delta_{SE_{\text{act}}} - \Delta_{SE_{\text{ant}}} = 2.$$

Diferencias de segundo orden (2/3)

Si escogemos el punto SE en la iteración actual, el punto de evaluación se mueve de (x_p, y_p) a $(x_p + 1, y_p - 1)$, por lo que $\Delta_{E_{\text{act}}}$ en $(x_p + 1, y_p - 1)$ es

$$= 2(x_p + 1) + 3,$$

y la diferencia de segundo orden es

$$\Delta_{E_{\text{act}}} - \Delta_{E_{\text{ant}}} = 2.$$

También $\Delta_{SE_{\text{act}}}$ en $(x_p + 1, y_p - 1)$ es

$$= 2(x_p + 1) - 2(y_p - 1) + 5 =$$
$$= 2x_p - 2y_p + 9,$$

y la diferencia de segundo orden es

$$\Delta_{SE_{\text{act}}} - \Delta_{SE_{\text{ant}}} = 4.$$

Los valores iniciales se calculan en $(0, r)$

$$\Delta_{E_0} = 3,$$
$$\Delta_{SE_0} = -2r + 5.$$

El algoritmo para trazar el círculo, usando las diferencias de segundo orden, queda finalmente como

```
int x = 0;
int y = radio;
int d = 1 - radio;
int deltaE = 3;
int deltaSE = -2*radio + 5;
PuntosCirculo( x, y, valor );
while  y > x  do
   if  d < 0  then
      d + = deltaE;
      deltaE + = 2;
      deltaSE + = 2;
   else { d == 0 se escoge SE. Ok! }
      d + = deltaSE;
      deltaE + = 2;
      deltaSE + = 4;
      y −−;
   end if
   x ++;
   PuntosCirculo( x, y, valor );
end while
```

Rellenado de rectángulos y polígonos

Para rellenar un rectángulo tenemos que realizar dos pasos:

1. Decidir qué pixeles se tienen que rellenar
2. Decidir el color del rellenado

Para un rectángulo

```
for y = y_mín : y_máx do
    for x = x_mín : x_máx do
        WritePixel(x, y, valor);
    end for
end for
```

El rectángulos se rellena por tramos.

Un posible problema: ¿ Qué hacer con las aristas compartidas por dos rectángulos?

Aristas compartidas

Una regla es considerar los pixeles en las aristas a la izquierda o abajo como parte de la figura, y 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.

El algoritmo general que se describirá para rellenar polígonos, 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.

Rellenando tramos

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 políigono.
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.

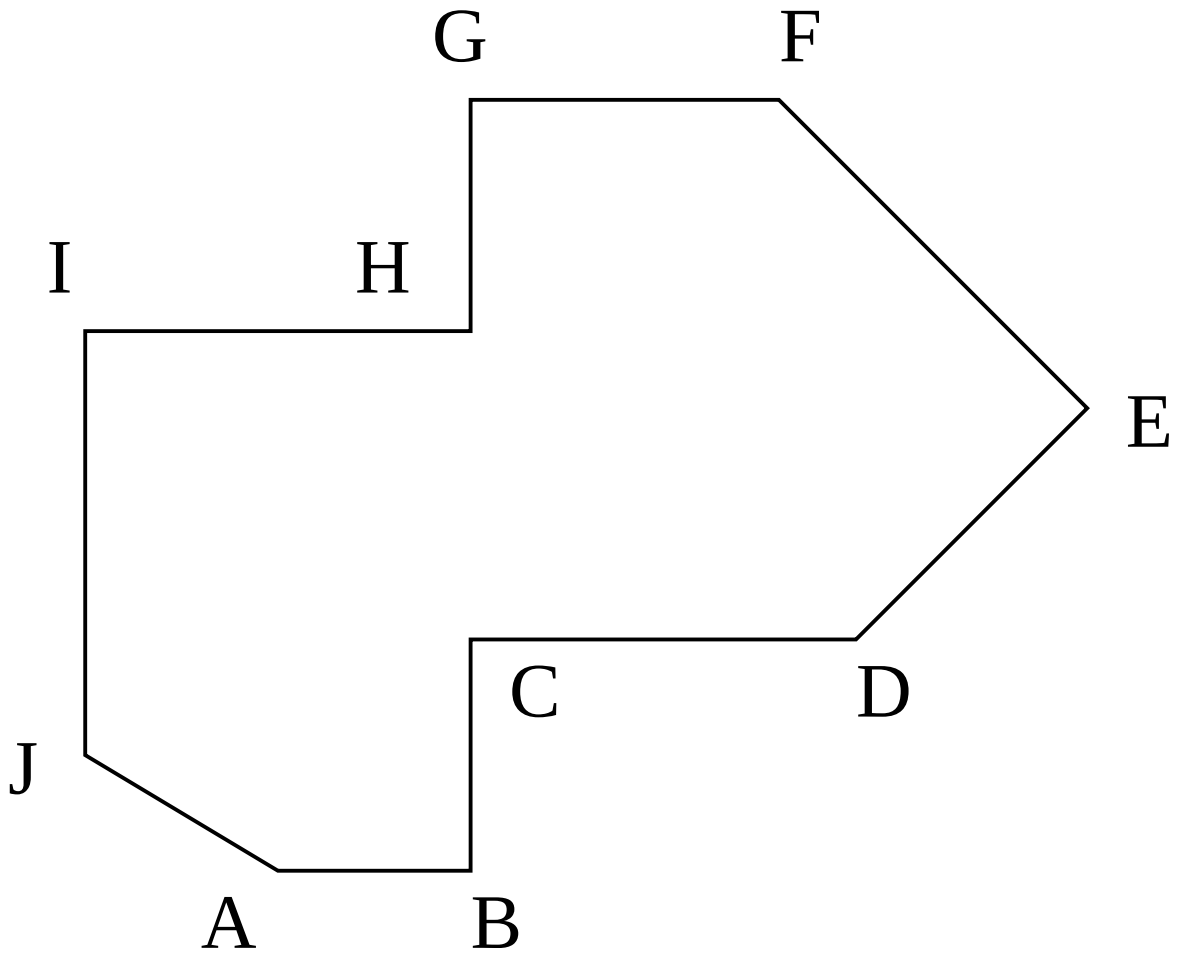
Paso 3

Este paso 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?

- Caso 3.1, se maneja redondeando de forma correcta.
- 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.
- Caso 3.3, se cuenta el $y_{\text{mín}}$ de una arista en el cálculo de paridad pero no se cuenta el vértice $y_{\text{máx}}$
- Caso 3.4, esto se resuelve automáticamente: una arista horizontal no tiene $y_{\text{mín}}$ ni $y_{\text{máx}}$.

Aristas horizontales



Algoritmo para una aristas izquierda

Require: *xmin*, *ymin*, *xmax*, *ymax*, *value*

$x \leftarrow xmin$

$numerator \leftarrow xmax - xmin$

$denominator \leftarrow ymax - ymin$

$increment \leftarrow denominator$

for $y \leftarrow ymin; y \leq ymax; y++$ **do**

WritePixel(x, y, *value*)

$increment += numerator$

if $increment > denominator$ **then**

{Sobreflujo, entonces se redondea el siguiente pixel}

$x++$

$increment -= denominator$

end if

end for

Ejemplo

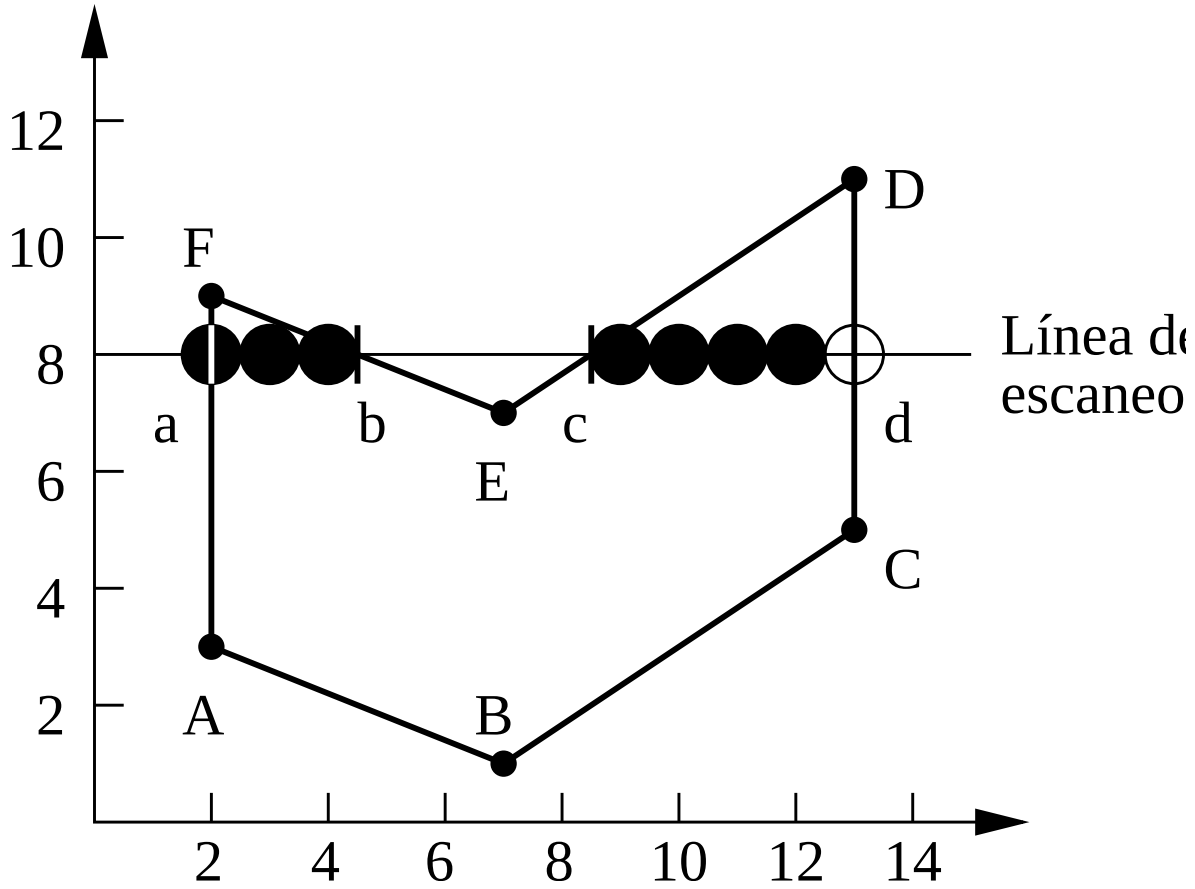


Tabla de aristas

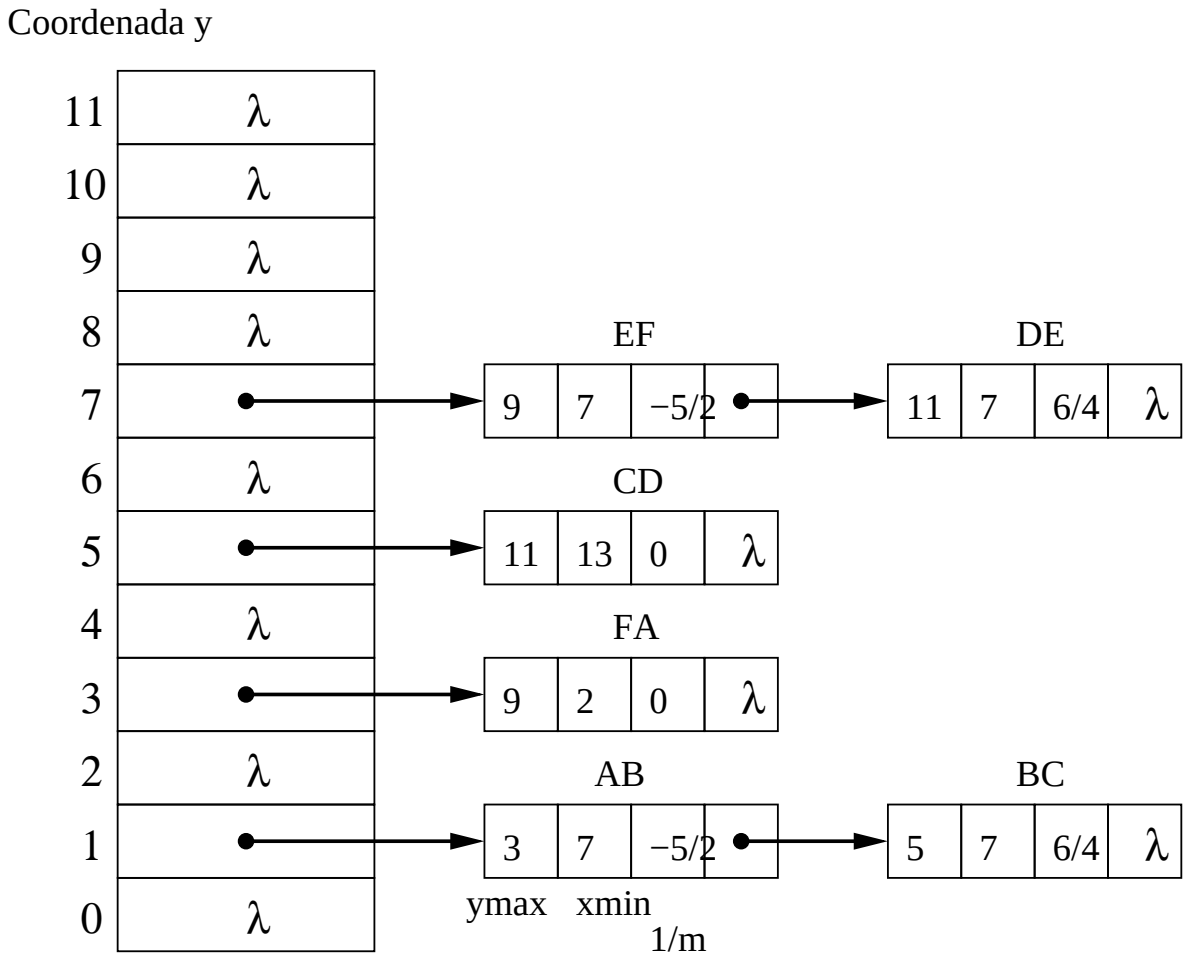


Tabla activa de aristas

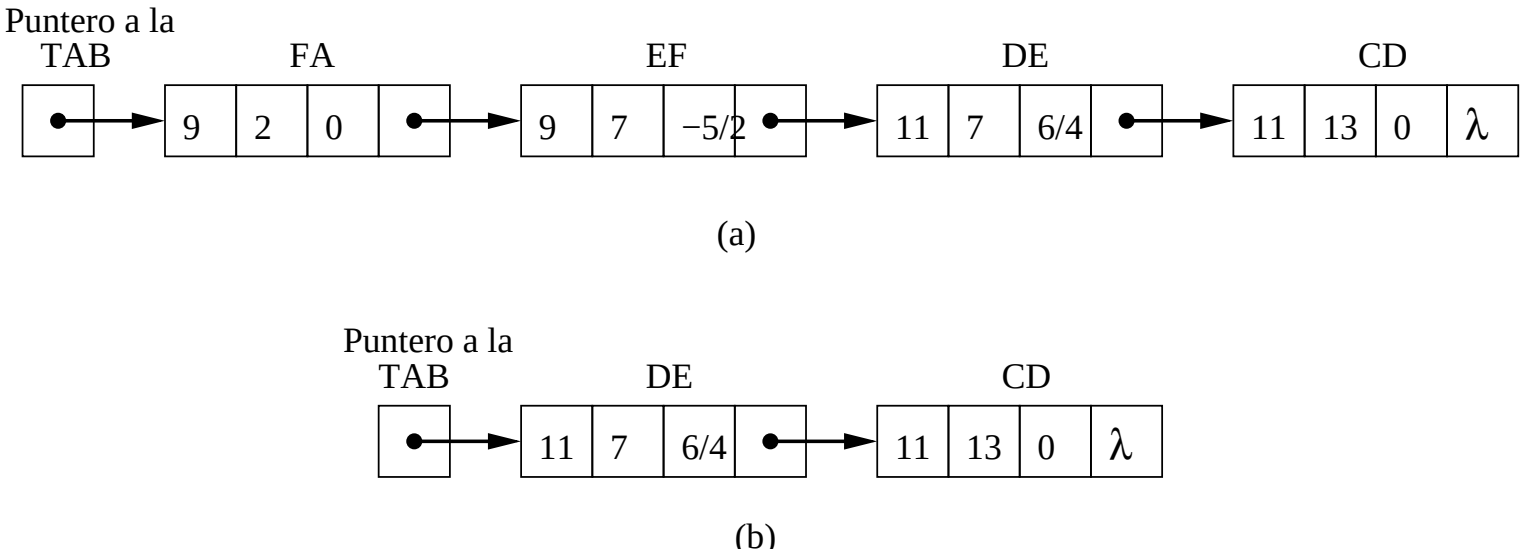


Tabla activa de aristas para el polígono de ejemplo. (a) Linea de escaneo 9. (b) Para la línea de escaneo 10.

Algoritmo

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

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.
2. Inicializar la TAA a vacío
3. Repetir hasta que la TAA y la TA estén vacíos
   - 3.1 Mover de la TA la cubeta y a la TAA aquellas aristas cuya $y_{\text{mín}} = y$ (aristas de entrada).
   - 3.2 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)
   - 3.3 Rellenar los pixeles deseados sobre la línea de escaneo y usando pares de coordenadas x de la TAA.
   - 3.4 Incrementa y en 1 (a la coordenada de la siguiente línea de escaneo.
   - 3.5 Para cada arista no vertical que quede en la TAA, poner al día x para la nueva y.

Primitivas gruesas

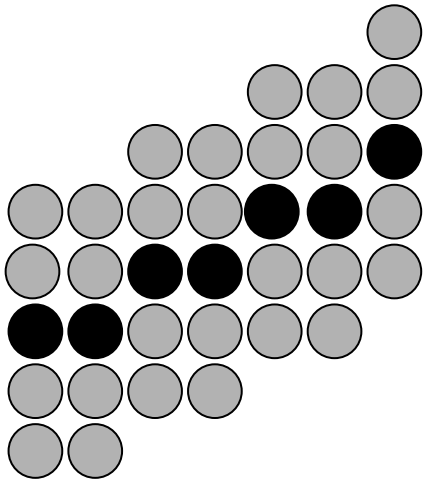
Primitivas gruesas

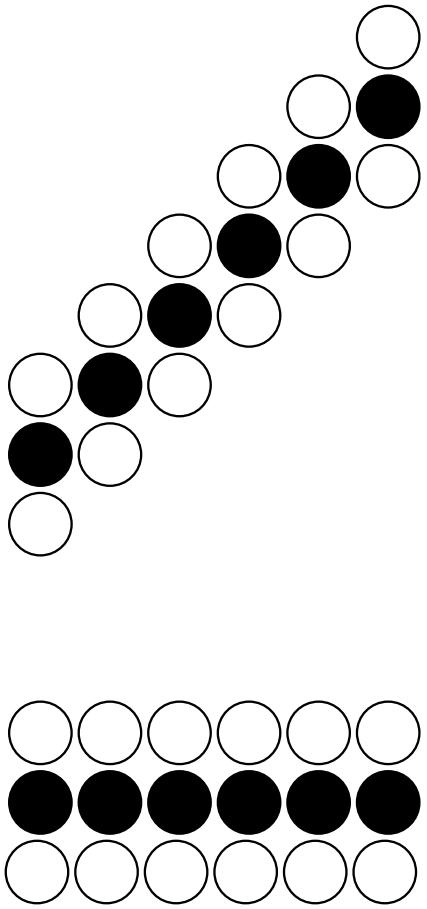
1. Replicación de pixeles (para lineas)
2. Pluma móvil (para lineas muy gruesas)
3. Rellenar entre dos primitivas, una externa y una interna (lo más general)

Replicación de pixeles

Para el algoritmo de trazo de líneas, se puede agregar un tercer ciclo, más interno, para duplicar pixeles en columnas, para lineas con pendientes menor a 1, y duplicar en los renglones para las otras líneas.

- Las líneas siempre terminan vertical u horizontalmente.
- Se producen huecos en los vértices donde se encuentran varias líneas.
- Las líneas parecen de distinto grueso según su pendiente.

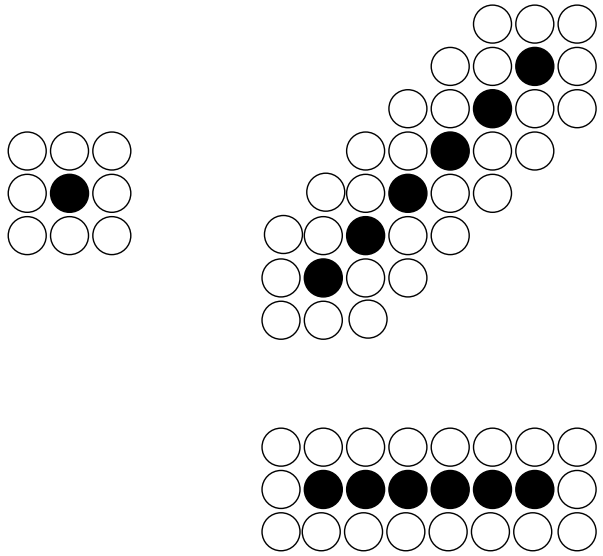




Es una aproximación eficiente pero muy cruda y sirve mejor para trazar líneas que no son muy gruesas.

Pluma móvil

Se puede usar una pluma rectángular, cuyo centro se mueve sobre los pixeles de la primitiva trazada. Esto funciona relativamente bien para trazar líneas gruesas.
El resultado es similar a la replicación de pixels pero es más grueso en los vertices terminales.



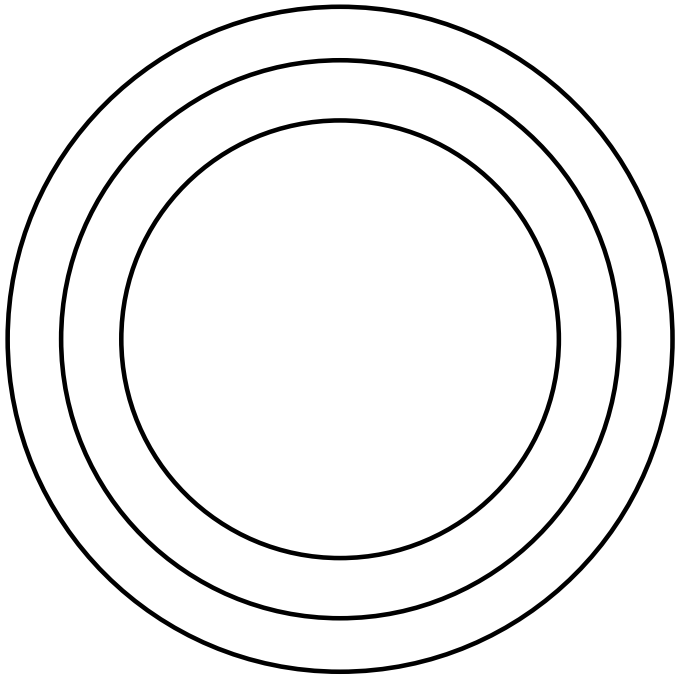
El resultado es mejor si se usa una pluma circular, para evitar que el grueso dependa de la orientación de la línea.
Una solución eficiente es calcular los tramos antes de rellenarlos.

Rellenar entre dos primitivas

Este tercer método para trazar primitivas gruesas consiste en construir dos primitivas, a una distancia $d/2$ a cada lado de la primitiva ideal.

- Asi se pueden manejar gruesos de tamaño par.
- Deben de calcularse los tramos.

Para el caso del círculo, dibujar sucesivos círculos concéntricos no sirve debido a que quedan huecos en el relleno.



Recortado

El recortado es una operación básica en los paquetes de dibujo. Es útil cuando se manejan posiciones enteras y es esencial cuando se usan posiciones en punto flotante.

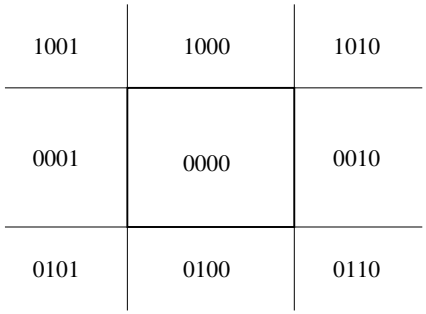
Vamo a ver come recortar líneas contra un solo rectángulo.
El recortado se hace para líneas y polígonos. Para otras primiticas es mejor hacerlo cuando se están dibujando los tramos.

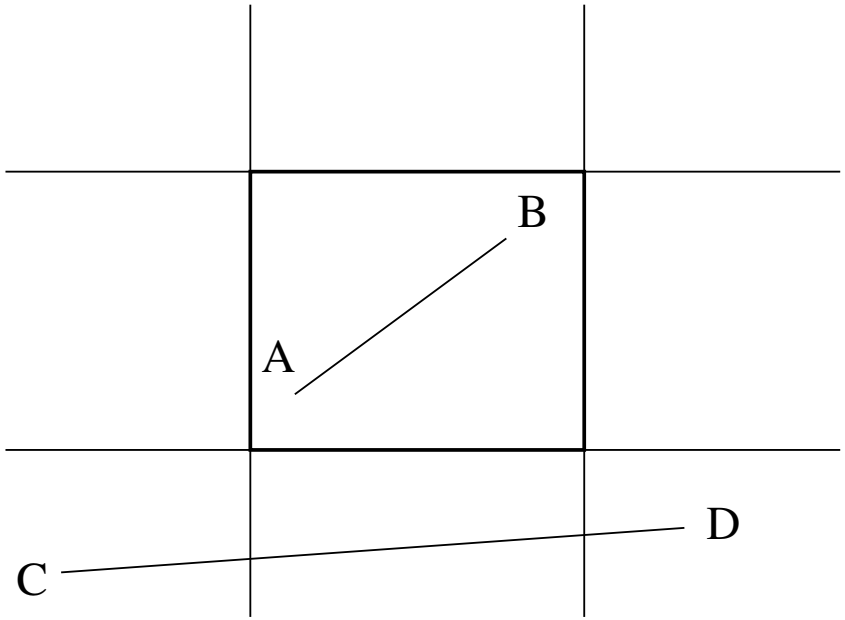
Algoritmo Cohen–Sutherland

- Este es un algoritmo eficiente para recortar
- Se basa en dividir el plano en zonas
- Una característica muy importante es que puede extenderse fácilmente a más dimensiones

1001	1000	1010
0001	0000	0010
0101	0100	0110

Primer bit	$y > y_{\text{máx}}$
Segundo bit	$y < y_{\text{mín}}$
Tercer bit	$x > x_{\text{máx}}$
Cuarto bit	$x > x_{\text{mín}}$



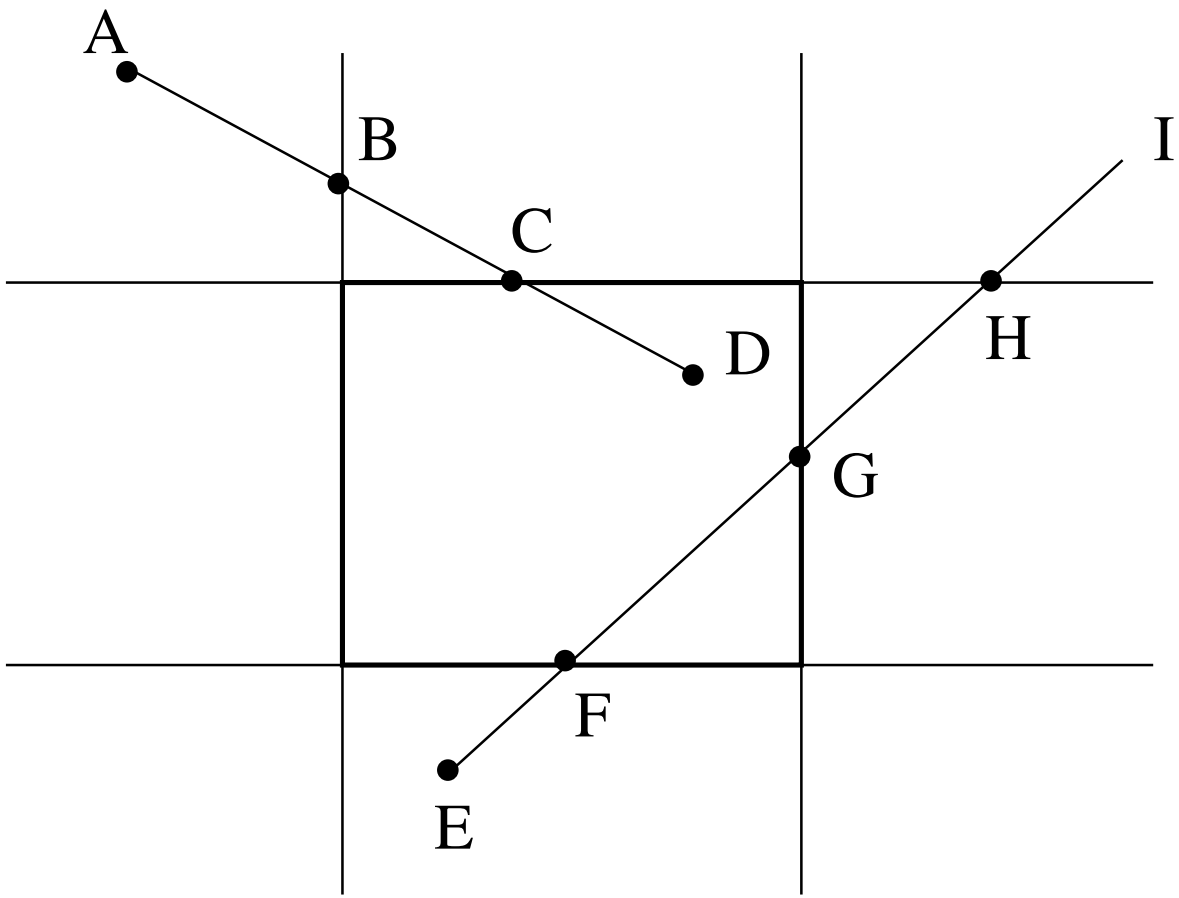


$$\begin{array}{rl} \text{codigo}(A) & = 0000 \\ \text{codigo}(B) & = 0000 \\ \hline \text{OR} & = 0000 \end{array}$$

y la linea se acepta trivialmente.

$$\begin{array}{rl} \text{codigo}(C) & = 0101 \\ \text{codigo}(D) & = 0110 \\ \hline \text{AND} & = 0100 \end{array}$$

y la linea se rechaza trivialmente.


A
B
C
D
E
F
G
H
I

```
typedef unsigned int outcode;
enum { TOP=0x1, BOTTOM=0x2, RIGHT=0x4, LEFT=0x8 };

void CohenSutherlandLineClipAndDraw(
    double x0, double y0, double x1, double y1, double xmin, double xmax,
    double ymin, duble ymax, int value)
/** Cohen-Sutherland clipping algorithm for line P0=(x0, y0) to
        P1 = (x1, y1) and clip rectangle with diagonal from
        (xmin, ymin) to (xmax, ymax) **/
{
    /** Outcode for P0, P1, and whatever point lies outside the clip
        rectabgle **/
    outcode outcode0, outcode1, outcodeOut;
    boolean accept=FALSE, done=FALSE;

    outcode0 = CompOutCode( x0, y0, xmin, ymin, xmax, ymax );
    outcode1 = CompOutCode( x1, y1, xmin, ymin, xmax, ymax );
    do {
        if( !(outcode0|outcode1) ) {   /* Trivial accept and exit */
            accept=TRUE; done=TRUE;
        }
        else if( outcode0&outcode1) /* Logical AND is true, so trivial */
            done = TRUE;            /* reject and exit                 */
```

```
else {
    /* Failed both tests, so calculate the line segment to clip
        from an outside point to an intersection with clip edge */
    double x, y;
    /* At least one endpoint is outside the clip rectangle; pick it */
    outcodeOut = outcode0 ? outcode0 : outcode1;

    /* Now find the intersecion point */

    if ( outcodeOut & TOP ) {
        x = x0 + (x1-x0)*(ymax - y0)/(y1 - y0);
        y = ymax;
    }
    else if (outcodeOut & BOTTOM) {
        x = x0 + (x1-x0)*(ymin - y0)/(y1 - y0);
        y = ymin;
    }
    else if (outcodeOut & RIGHT) {
        y = y0 + (y1-y0)*(xmax - x0)/(x1 - x0);
        x = xmax;
    }
    else {
        y = y0 + (y1-y0)*(xmin - x0)/(x1 - x0);
        x = xmin;
    }
    if ( outcodeOut == outcode0 ) {
        x0 = x;
        y0 = y;
        outcode0 = CompOutCode( x0, y0, xmin,xmax,ymin,ymax);
    }
    else {
        x1 = x;
        y1 = y;
        outcode1 = CompOutCode( x1, y1, xmin,xmax,ymin,ymax);
    }
}
```

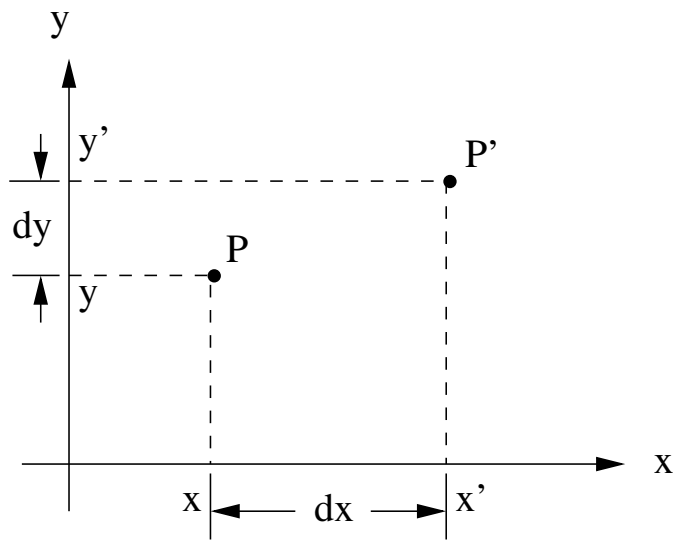
```
    } while( done==FALSE );

    if ( accept ) {
        /** We draw the line **/
        /** Function version for double coordinates **/
        MidPointLineReal( x0, y0, x1, y1, value );
    }
}


outcode CompOutCode (
    double x, double y,
    double xmin, double xmax,
    double ymin, double ymax )
{
    outcode code=0;
    if ( y > ymax )
        code |= TOP;
    else if( y < ymin )
        code |= BOTTOM;
    if ( x > xmax )
        code |= RIGHT;
    else if ( x < xmin )
        code |= LEFT;

    return code;
}
```

Transformaciones geométricas

Transformaciones geométricas

1. Translación
2. Rotación
3. Escalamiento
4. Sesgado (shear)

Translación



De la figura:

$$x' = x + dx,$$
$$y' = y + dy.$$

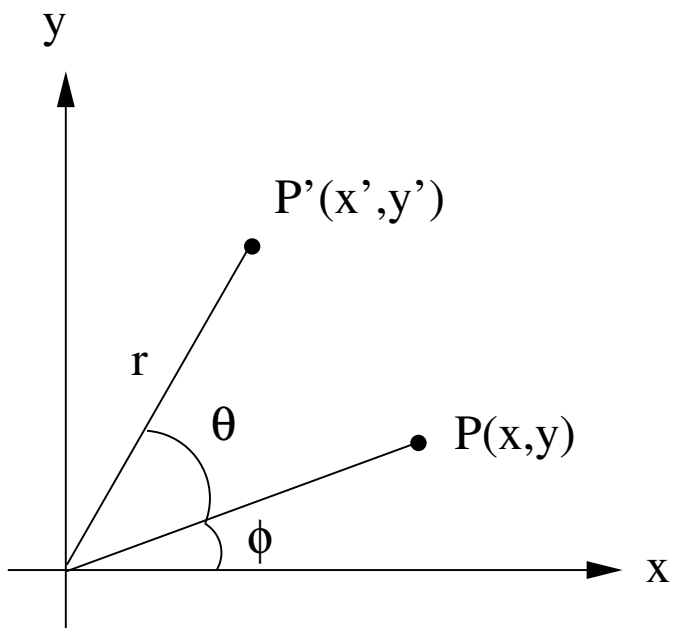
En forma matricial:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix},$$

o

$$P' = P + T$$

Rotación



De la figura:

$$x = r\cos\phi,$$
$$y = r\,\text{sen}\,\phi,$$

y

$$x' = r\cos(\theta+\phi),$$
$$y' = r\,\text{sen}(\theta+\phi),$$

Usando las fórmulas

$$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \text{sen}\,\alpha\,\text{sen}\,\beta,$$
$$\text{sen}(\alpha+\beta) = \cos\alpha\,\text{sen}\,\beta + \text{sen}\,\alpha\cos\beta,$$

Llegamos a

$$x' = x\cos\theta - y\,\text{sen}\,\theta$$
$$y' = x\,\text{sen}\,\theta + y\cos\theta$$

$$x' = x\cos\theta - y\,\text{sen}\,\theta$$
$$y' = x\,\text{sen}\,\theta + y\cos\theta,$$

en forma matricial:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta \\ \text{sen}\,\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

o

$$P' = R(\theta)P$$

Escalamiento

$$x' = e_x x,$$
$$y' = e_y y,$$

o en forma matricial

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} e_x & 0 \\ 0 & e_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

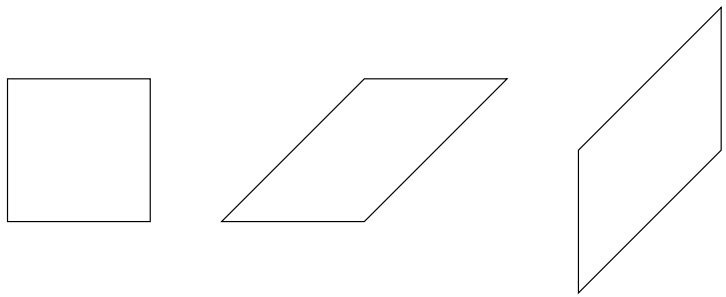
o

$$P' = E(e_x, e_y)P$$

$$x' = x + ay$$
$$y' = y$$

O en forma matricial

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



$$x' = x$$
$$y' = bx + y$$

O en forma matricial

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Coordenadas homogéneas (1/2)

La representación matricial para la rotación, translación y escalamiento son

$$P' = R(\theta)P$$
$$P' = T(dx, dy) + P$$
$$P' = E(e_x, e_y)P$$

Desafortudamente, la translación es tratada de forma diferente, como una adición.
Para poner todas las transformaciones en un mismo marco de referencia se usan las coordenadas homogéneas.

Coordenadas homogéneas (2/2)

En coordenadas homogéneas se adiciona una tercera coordenada a cada punto, (x, y) se representa por el vector

$$\begin{bmatrix} x \\ y \\ W \end{bmatrix}$$

Dos coordenadas homogéneas representan al mismo punto si y solo si una es un múltiplo de la otra.

- El punto $[0, 0, 0]^T$ no está permitido.
- Si la tercera coordenada es distinta de cero, se homogeiniza dividiéndolo entre W: $[x/W, y/W, 1]^T$
- Los puntos con $W = 0$ se llaman *puntos al infinito*

Transformaciones con coordenadas homogéneas

Translación:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotación:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Escalamiento:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} e_x & 0 & 0 \\ 0 & e_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translación:

$$T^{-1}(dx, dy) = \begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$$

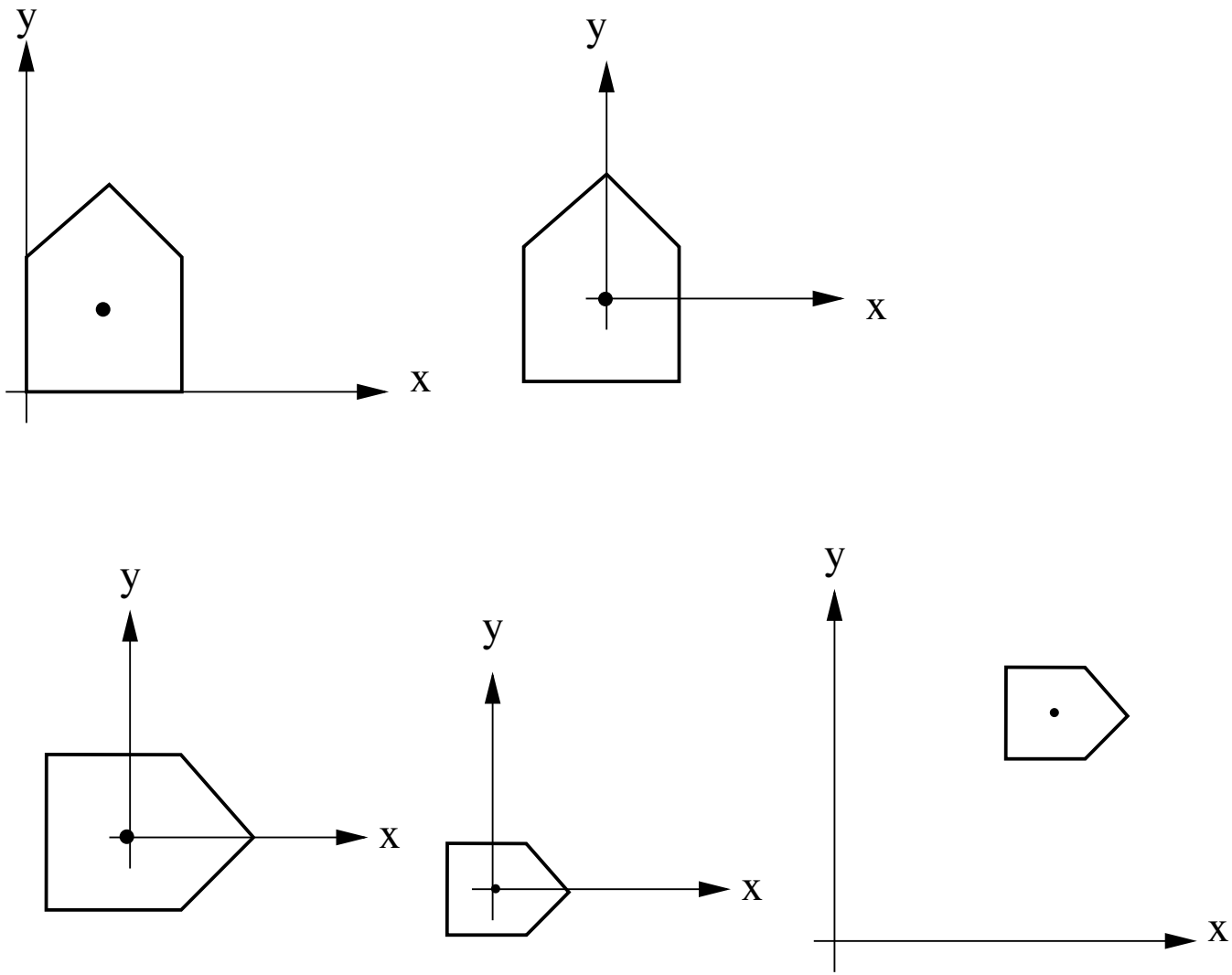
Escalamiento:

$$E^{-1}(e_x, e_y) = \begin{bmatrix} \frac{1}{e_x} & 0 & 0 \\ 0 & \frac{1}{e_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

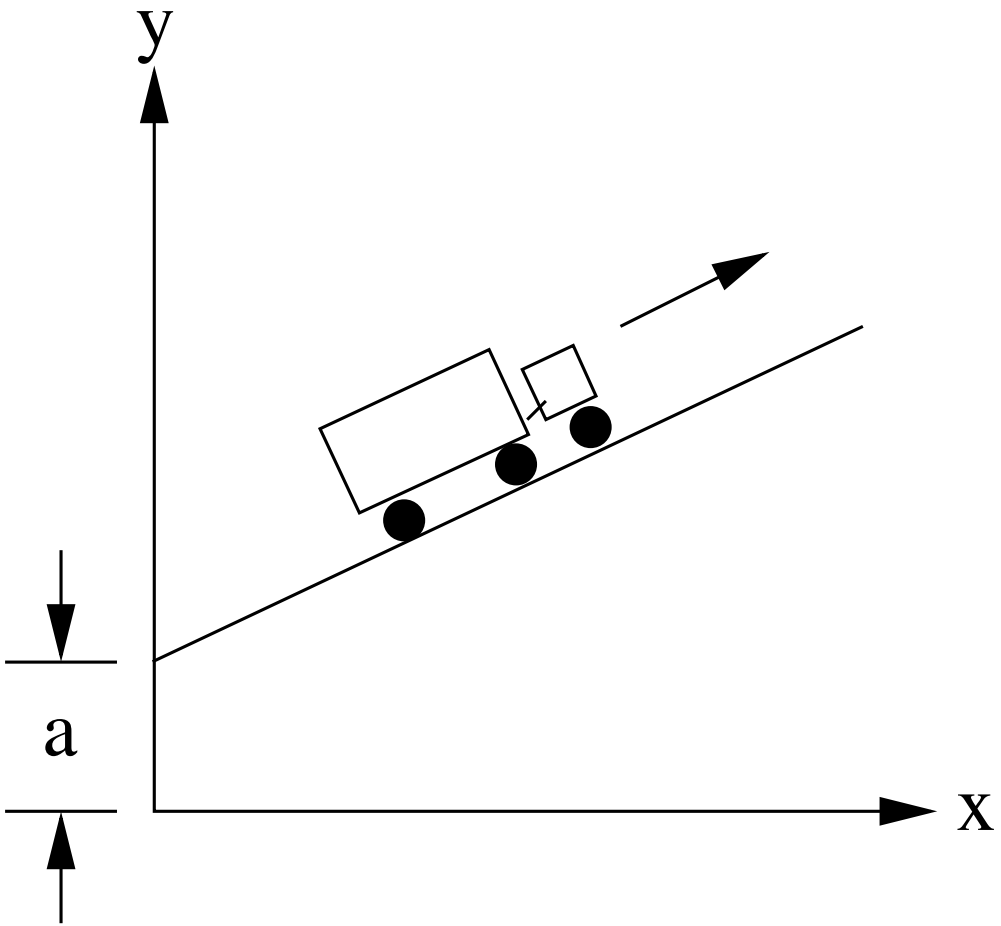
Rotación:

$$R^{-1}(\theta) = \begin{bmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

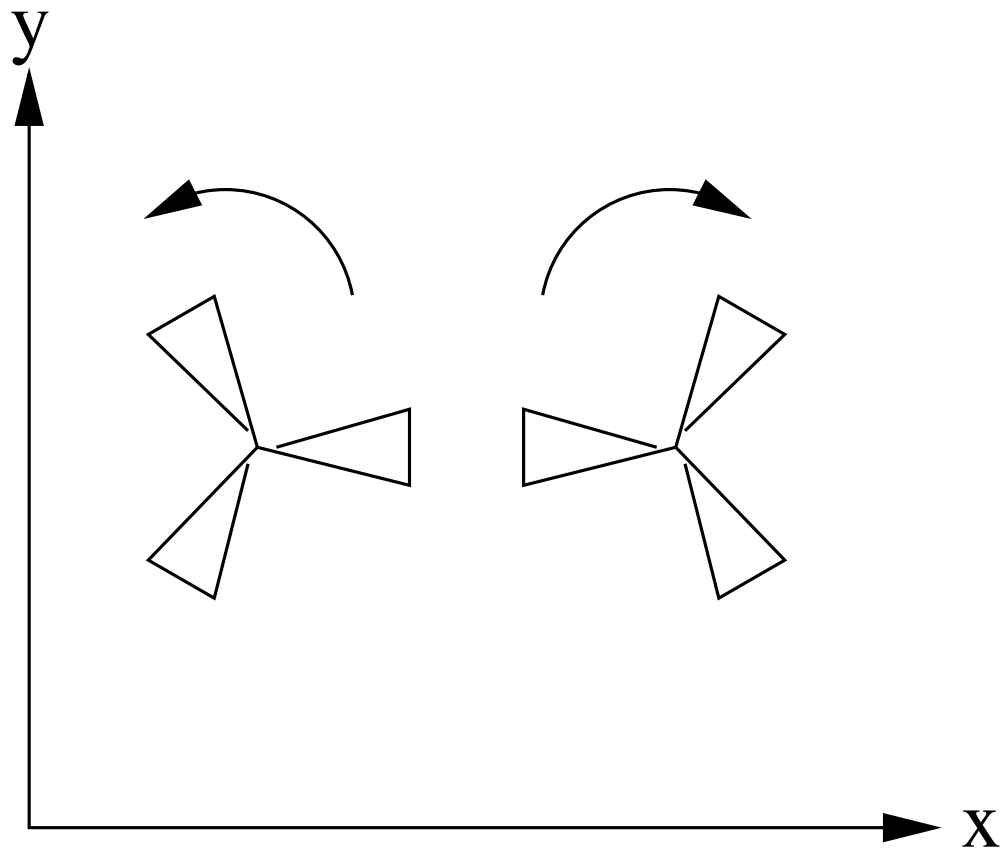
Composición de transformaciones



Ejemplo 2

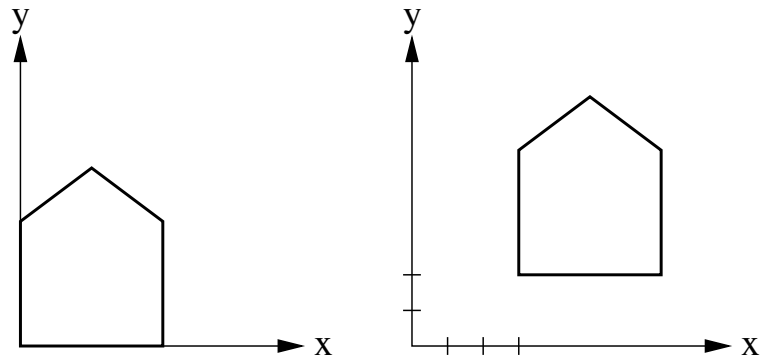


Ejemplo 3



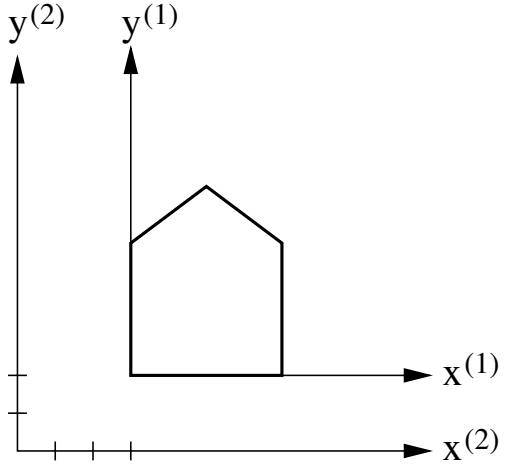
Las transformaciones como un cambio de coordenadas

En vez de mover el objeto, podemos mover los ejes de coordenadas.
Se efectuan los mismos cambios, solo es otra forma de visualizar las transformaciones.
Para transladar un cuerpo



$$P' = T(3,2)P$$

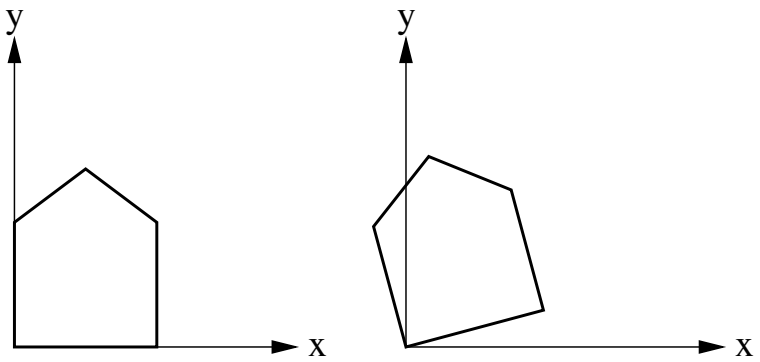
Cambiando los ejes:



$$P^{(2)} = M_{2\leftarrow 1}P^{(1)}$$

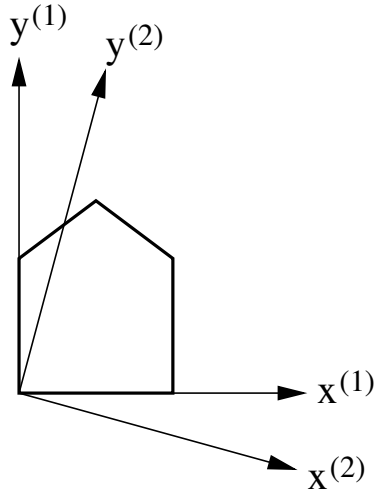
$$P^{(2)} = T(3,2)P^{(1)}$$

Rotación del objeto



$$P' = R(\theta)P$$

Rotación de los ejes



$$P^{(2)} = M_{2\leftarrow 1}P^{(1)}$$

$$P^{(2)} = R(\theta)P^{(1)}$$

Transformaciones de cónicas

De la ecuación del círculo

$$x^2 + y^2 = r^2,$$

puede expresarse también en forma matricial como

$$[x, y, 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -r^2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0,$$

$$\mathbf{x}^{\mathsf{T}} Q \mathbf{x} = 0.$$

Transformándolo

Se aplica una transformación cualquiera M al círculo cómo:

$$Q' = M^{-\mathsf{T}} Q M^{-1}.$$

Ec. de la elipse

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

en forma matricial:

$$[x, y, 1] \begin{bmatrix} \frac{1}{a^2} & 0 & 0 \\ 0 & \frac{1}{b^2} & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0.$$

Ecuación general de una cónica

La ecuación general de una cónica es:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0,$$

que representa

- Una elipse si $b^2 - 4ac < 0$, y un círculo si $a = c$ y $b = 0$,
- una parábola si $b^2 - 4ac = 0$,
- una hipérbola si $b^2 - 4ac > 0$,

En forma matricial:

$$[x, y, 1] \begin{bmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

OpenGL (Open Graphics Library) es una biblioteca estandar (API) para cualquier lenguaje de programación y cualquier sistema operativo para producir gráficos en 2D o 3D.
La biblioteca consiste de cerca de 250 funciones que pueden usarse para dibujar escenas complejas a partir de primitivas simples.

Fue desarollado por Silicon Graphics Inc. in 1992 y se usa en CAD, realidad virtual, visualización científica, visualización de información, simuladores de vuelo y video juegos. Actualmente OpenGL es administrado por el consorcio sin fines de lucro Khronos Group.

Programa "Hola mundo" en OpenGL (1/3)

Un programa en OpenGL debe tener al menos tres funciones

1. Una para inicializar la máquina de estados de OpenGL
2. Una que maneje el cambio de tamaño de la ventana
3. Una que muestre el dibujo

Programa "Hola mundo" en OpenGL (2/3)

```
program_entrypoint
{
    // Determine which depth or pixel format should be used.
    // Create a window with the desired format.
    // Create a rendering context and make it current with the window.
    // Set up initial OpenGL state.
    // Set up callback routines for window resize and window refresh.
}

handle_resize
{
    glViewport(...);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    // Set projection transform with glOrtho, glFrustum, gluOrtho2D, gluPerspec
}
```

Programa "Hola mundo" en OpenGL (3/3)

```
handle_refresh
{
    glClear(...);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // Set view transform with gluLookAt or equivalent

    // For each object (i) in the scene that needs to be rendered:
        // Push relevant stacks, e.g., glPushMatrix, glPushAttrib.
        // Set OpenGL state specific to object (i).
        // Set model transform for object (i) using glTranslatef,
        // glScalef, glRotatef, and/or equivalent.
        // Issue rendering commands for object (i).
        // Pop relevant stacks, (e.g., glPopMatrix, glPopAttrib.)
    // End for loop.

    // Swap buffers.
}
```

Tipos en OpenGL

Letra	Tipo en GL
b	byte
s	short
i	int
f	float
d	double
ub	ubyte
us	ushort
ui	uint

Formato de una función:

rtype Namef{ $\in$ 1234}{$\in$ b s i f d ub us ui}{$\in$ v}(argumentos);

Para dibujar

Las primitivas de dibujado van entre pares Begin() End()

void Begin(enum modo);
void End(void);

Modos:

- POINTS
- LINE_STRIP
- LINE_LOOP
- LINES
- POLYGON
- TRIANGLE_STRIP
- TRIANGLE_FAN
- TRIANGLES
- QUAD_STRIP
- QUADS

Dentro de pares Begin() End() puede haber

- Colores
- Normales
- Vértices

Especificación de vértices

void Vertex{234}{sifd}(T coords);
void Vertex{234}{sifd}{v}(T coords);

Proyecciones

Proyecciones

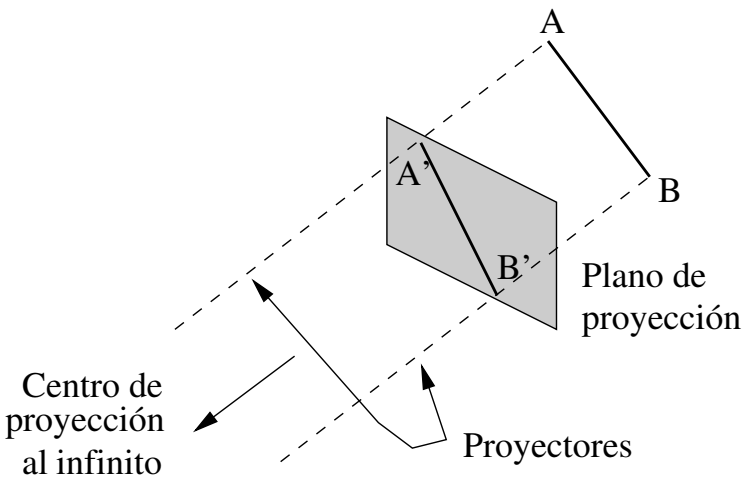
Las proyecciones transforman objetos 3D en proyecciones 2D planas.

En general, las proyecciones transforman puntos en un sistema de coordenadas de dimensión n, en puntos en un sistema de coordenadas de dimensión menor que n.

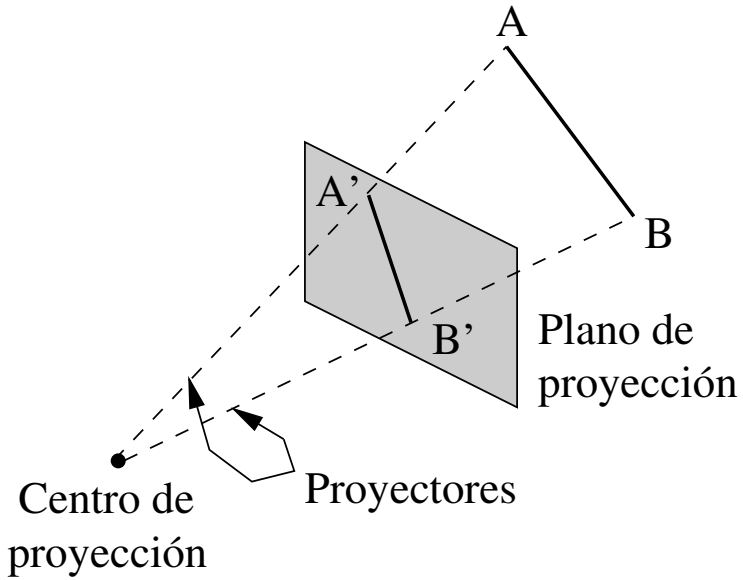
En graficación se estudia cómo proyectar objetos n–dimensionales hacia 2D, para visualizarlos.

Las proyecciones de un objeto 3D están formadas por *proyectores* que emanan de un *centro de proyección*, pasando por cada punto del objeto e intersectando un *plano de proyección*, que es donde se visualiza la proyección.

Las proyecciones que vamos a ver ahora son *proyecciones geométricas planas*, debido a que la proyección se realiza sobre un plano, en vez de alguna superficie curva, y usa proyectores rectos, en vez de curvados.



Proyección paralela



Proyección en perspectiva

La deformación visual que introduce la perspectiva se le llama *escorzado* (en inglés, *foreshortening*).

El tamaño de la proyección en pespectiva de un objeto varía inversamente con la distancia del objeto al centro de proyección.

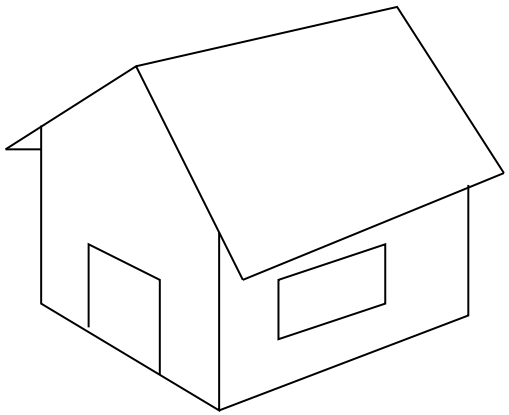
Subclases de proyecciones geométricas planas

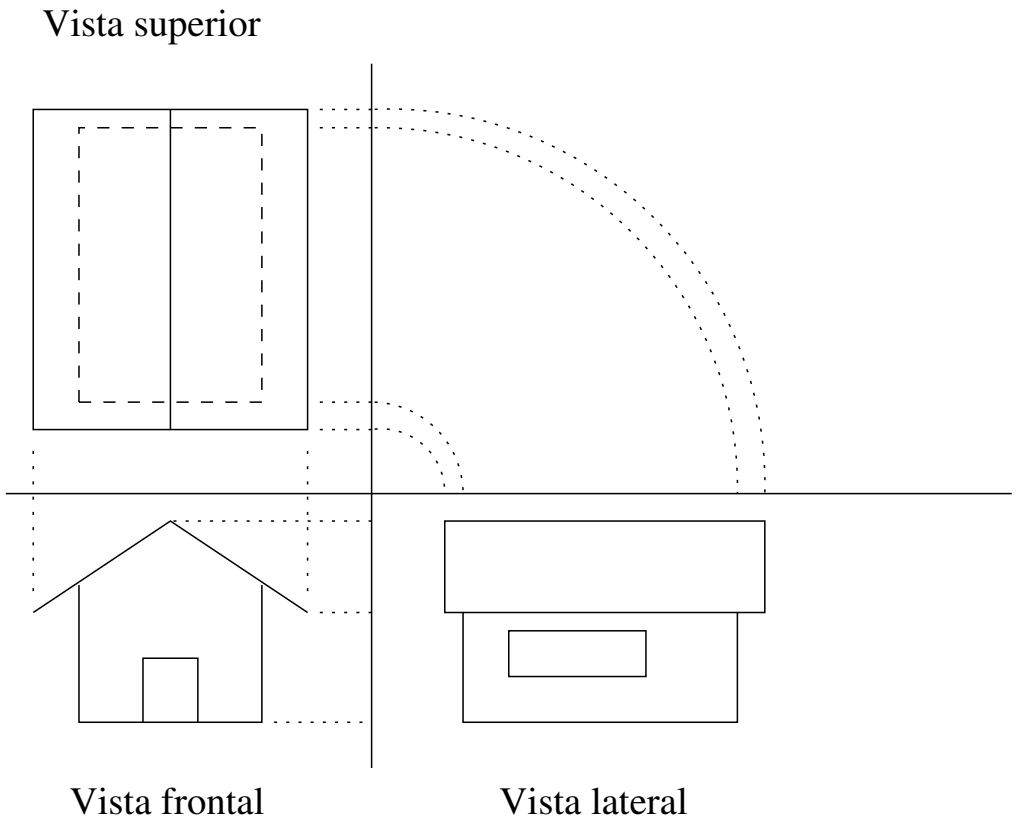
Paralelas

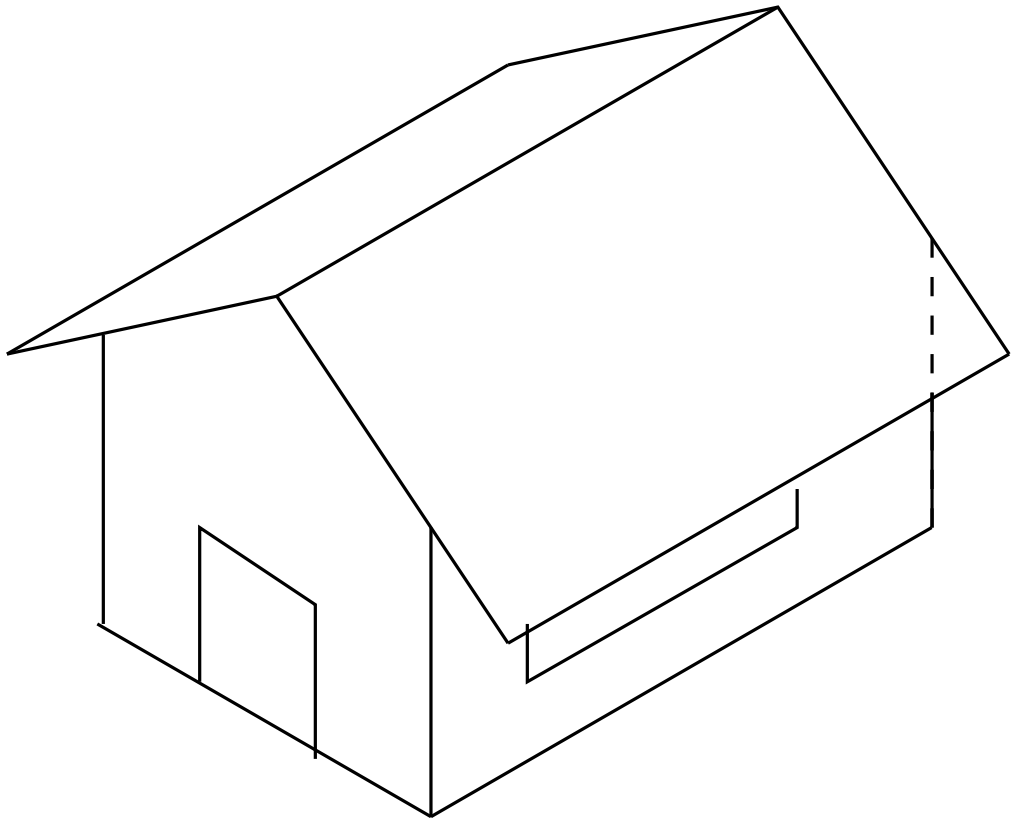
- Ortográficas
  - Vista superior
  - Vista frontal
  - Vista lateral
  - Axonométrica
    - Isométrica
    - Otra
- Oblicuas
  - Cabinet
  - Cavalier
  - Otra

Perspectiva

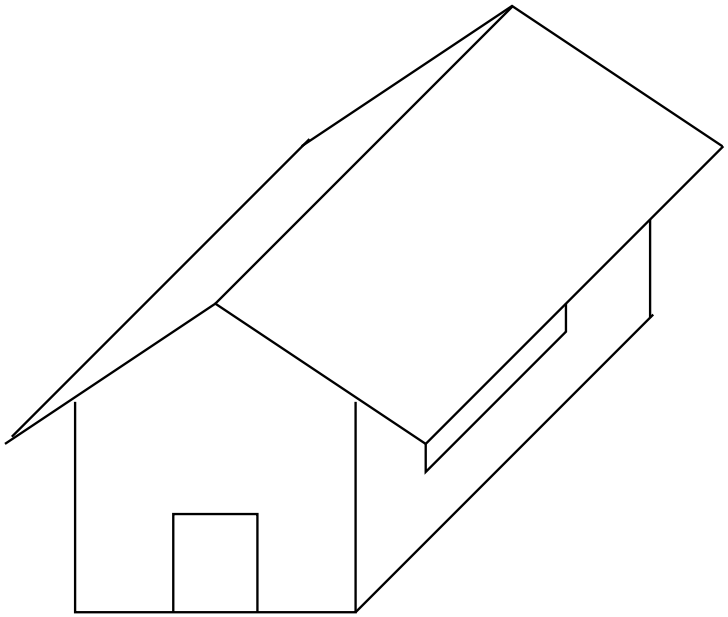
- Un punto
- Dos puntos
- Tres puntos



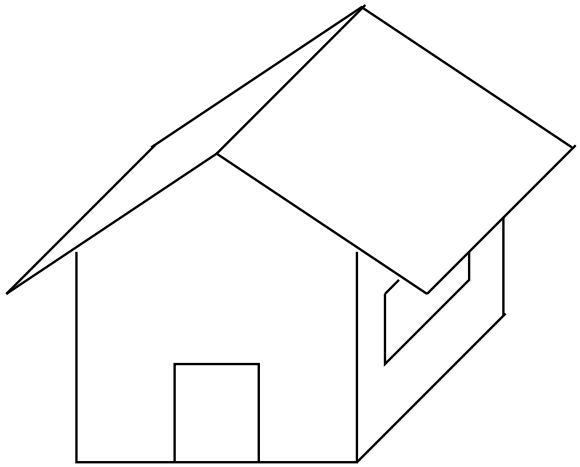

Vista superior
Vista frontal
Vista lateral



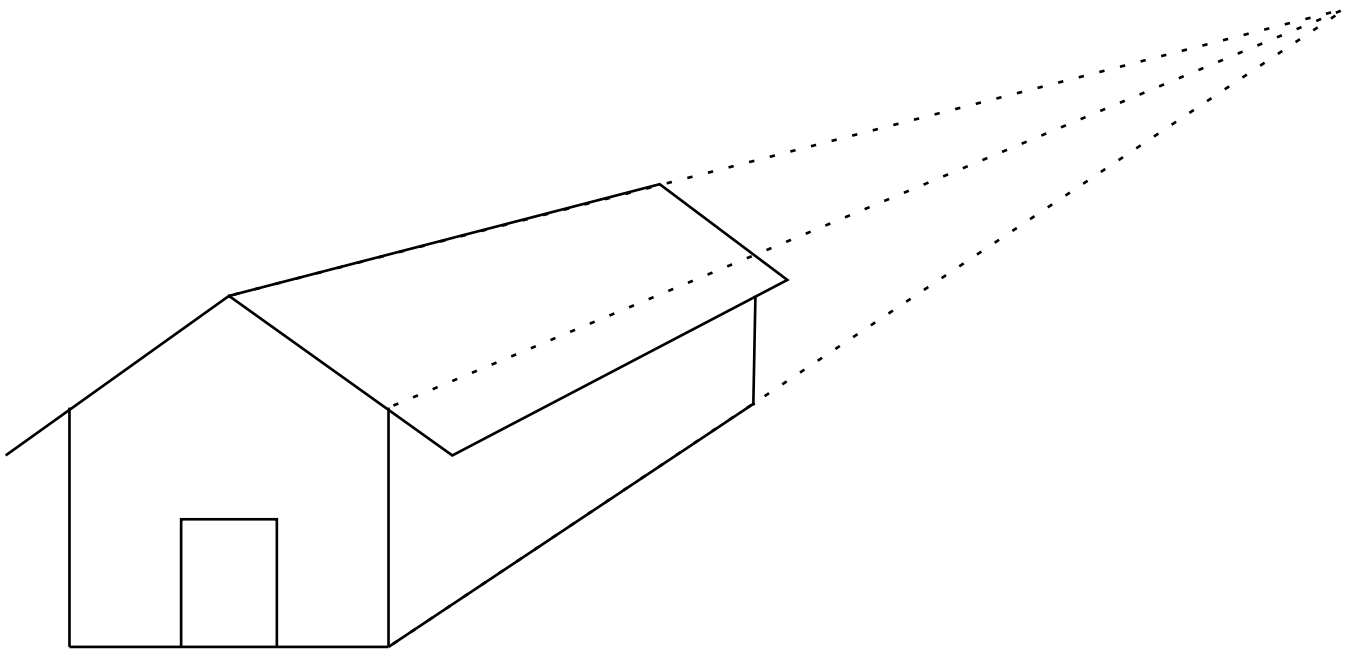
Isométrica



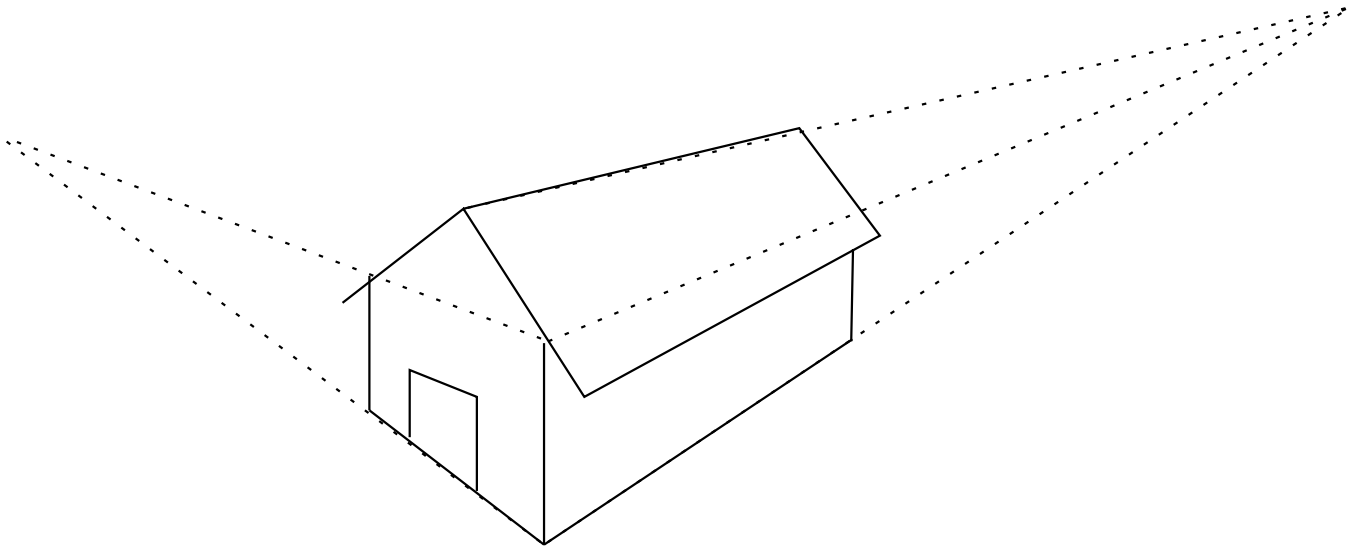
Cavalier



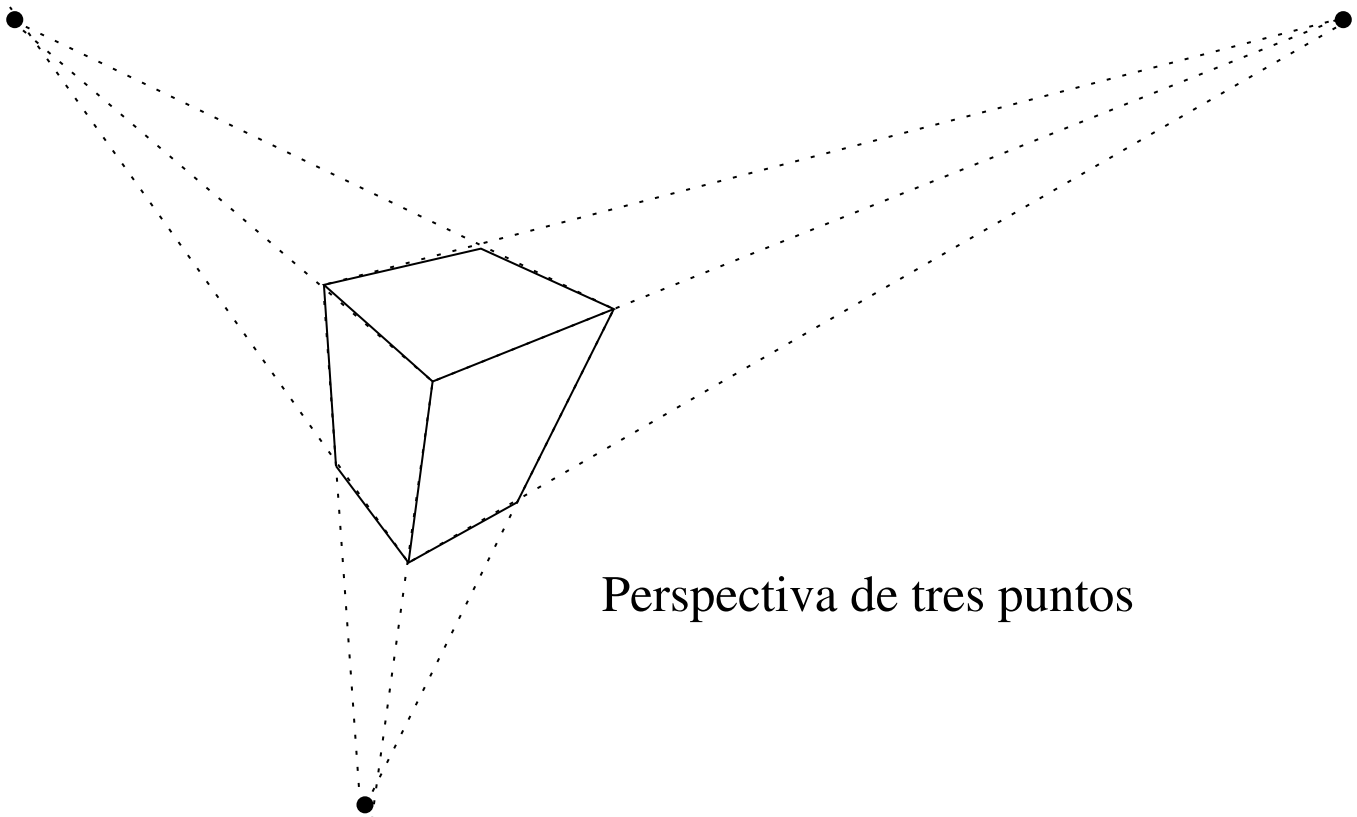
Cabinet



Perspectiva de un punto



Perspectiva de dos puntos



Perspectiva de tres puntos

Para proyectar ortográficamente sobre un plano de proyección en $z = 0$ es directo. La dirección de proyección es la misma que la normal al plano de proyección (esto es, el eje z).

Un punto P se proyecta como

$$\begin{aligned} x' &= x, \\ y' &= y, \\ z' &= 0. \end{aligned}$$

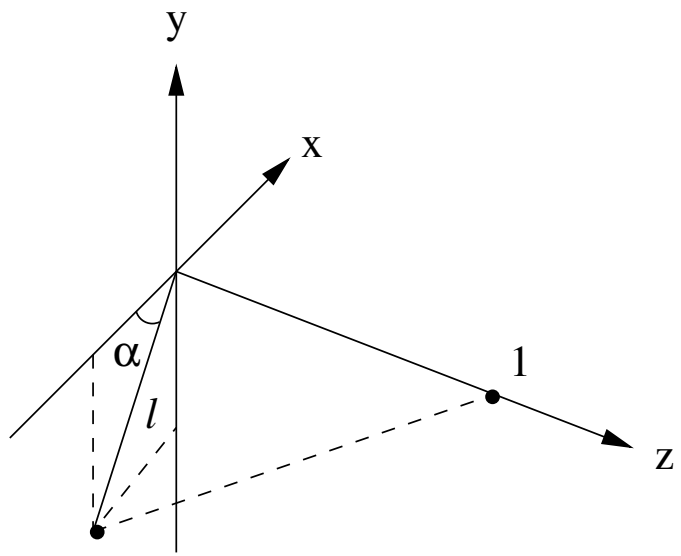
O en forma matricial

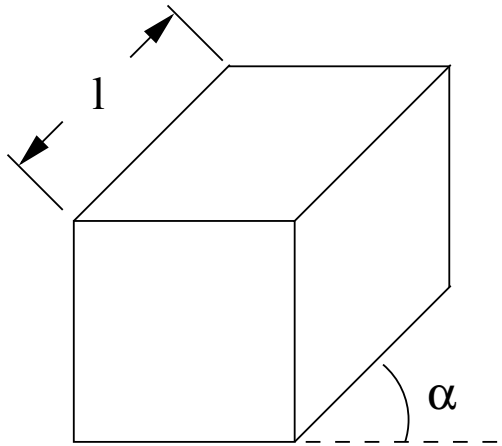
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

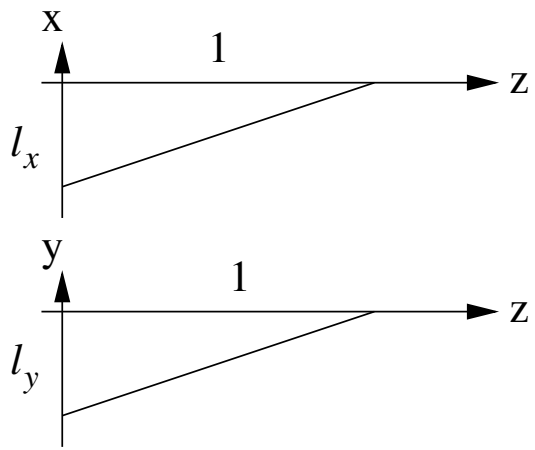
o

$$\mathbf{P}' = M_{\text{orto}}\mathbf{P}$$

Proyecciones oblicuas







El punto $[0, 0, 1]^{\mathsf{T}}$ se proyecta sobre el plano de proyección (que es el plano xy) en el punto

$$l_x = -l\cos\alpha,$$
$$l_y = -l\,\text{sen}\,\alpha.$$

Como es una proyección paralela, todos los proyectores tendrán la misma pendiente. De la ecuación de la línea

$$x = m_x z + B_x, \text{ y } y = m_y z + B_y,$$

Los puntos de intersección con los ejes son precisamente $x' = B_x$ y $y' = B_y$, por lo tanto

$$x' = x - zl\cos\alpha,$$
$$y' = y - zl\,\text{sen}\,\alpha,$$
$$z' = 0.$$

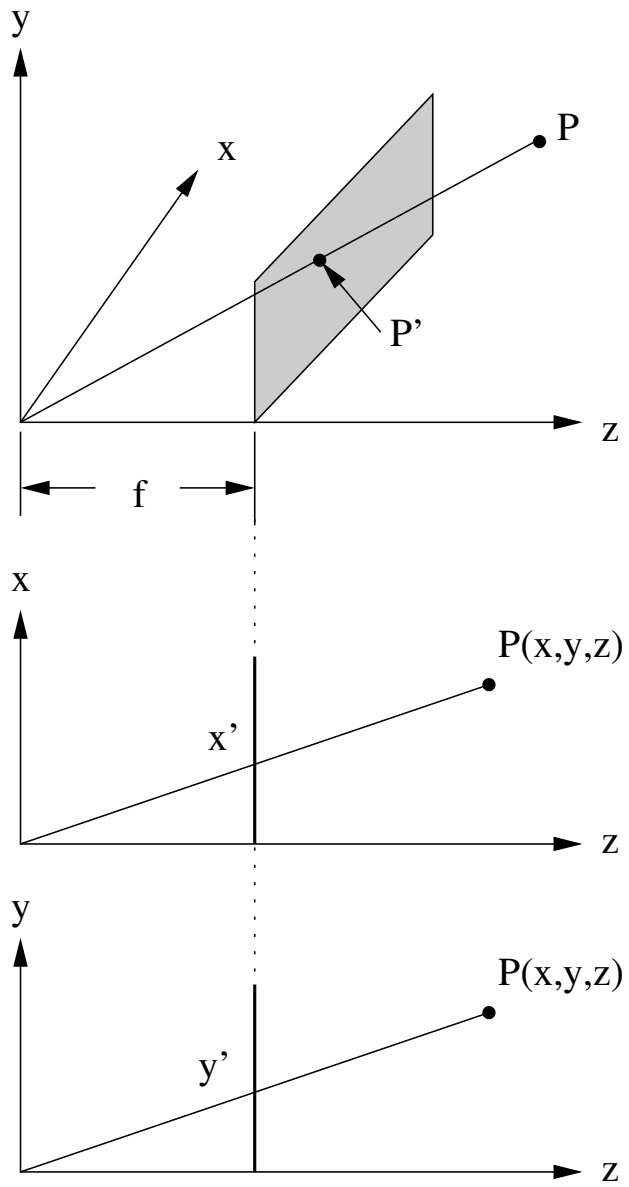
o en forma matricial

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -l\cos\alpha & 0 \\ 0 & 1 & -l\,\text{sen}\,\alpha & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

o

$$\mathbf{P}' = M_{\text{ob}}\mathbf{P}$$

Proyección en perspectiva



De la figura, por triángulos semejantes, tenemos

$$\frac{x'}{f} = \frac{x}{z}, \text{ y}$$
$$\frac{y'}{f} = \frac{y}{z},$$

por lo que

$$x' = \frac{xf}{z}$$
$$y' = \frac{yf}{z}$$

En forma matricial

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

o

$$\mathbf{P}' = M_{\text{pers}}\mathbf{P}$$