Modelación y Computación Gráfica para Ingenieros*

Patricio Inostroza, Dr.

*Material creado originalmente por Dr. Ivan Sipiran

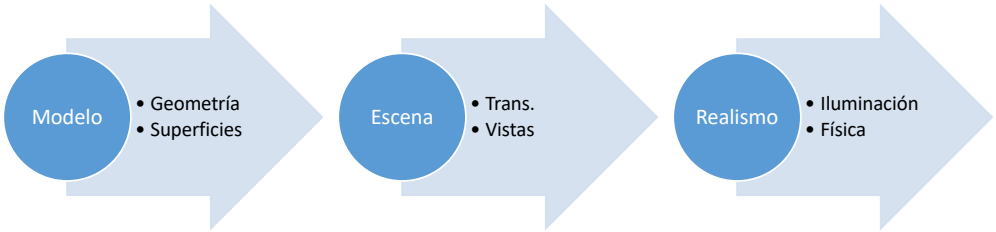
https://www.youtube.com/watch?v=g65Cp_pakM0

Computación Gráfica

- Simular el mundo real usando computadoras
- Especial énfasis en modelos físicos para añadir realismo
  - Geometría 3D
  - Iluminación
  - Cinemática
  - Dinámica de fluidos

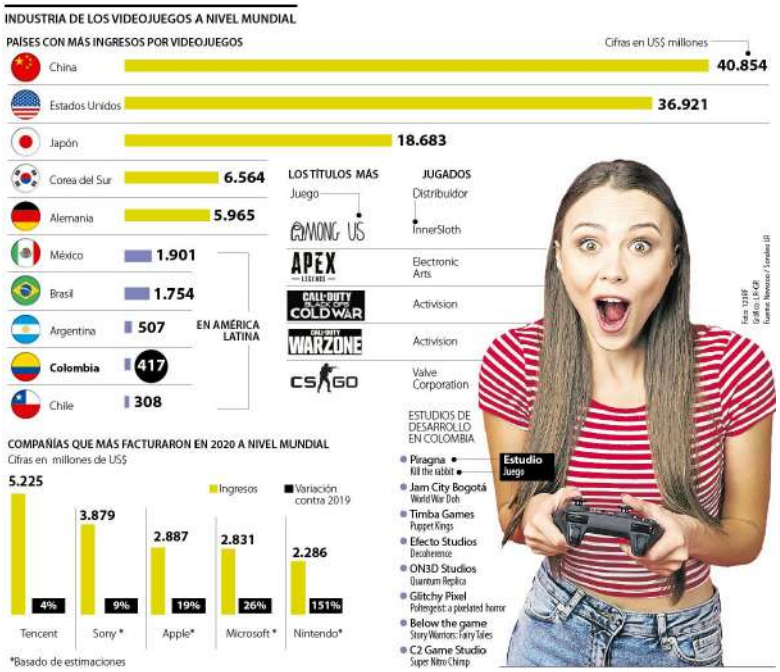
Computación Gráfica

- Generación de imágenes foto-realistas desde escenas computacionales
- Escena es una representación 2D o 3D (geometría)
- Proceso de convertir una escena en una imagen se llama Rendering



Aplicaciones

Videojuegos



Aplicaciones

- Videojuegos

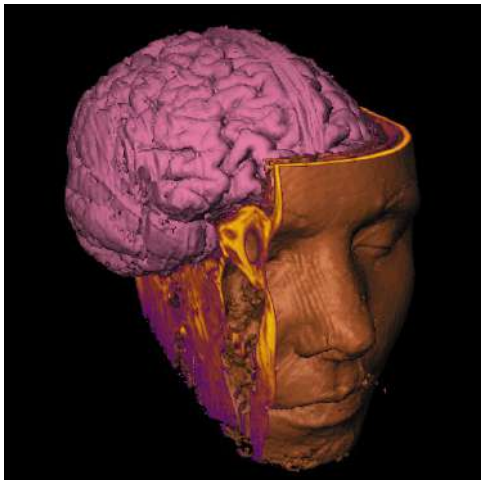
Aplicaciones

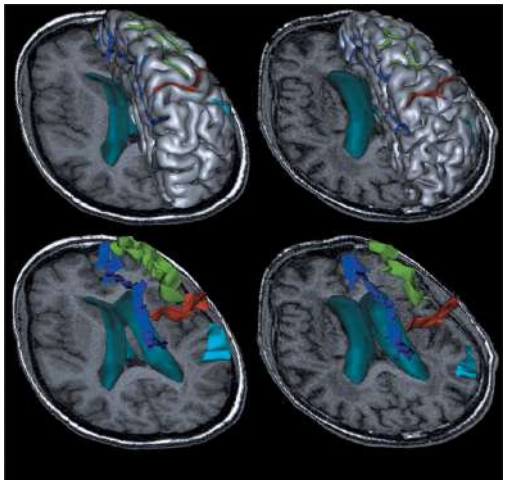
- Industria cinematográfica
  - Estudios Dreamworks, Pixar-Disney
  - 800 millones de dólares por película en promedio (taquilla)

https://en.wikipedia.org/wiki/List_of_highest-grossing_animated_films

Aplicaciones

- Medicina

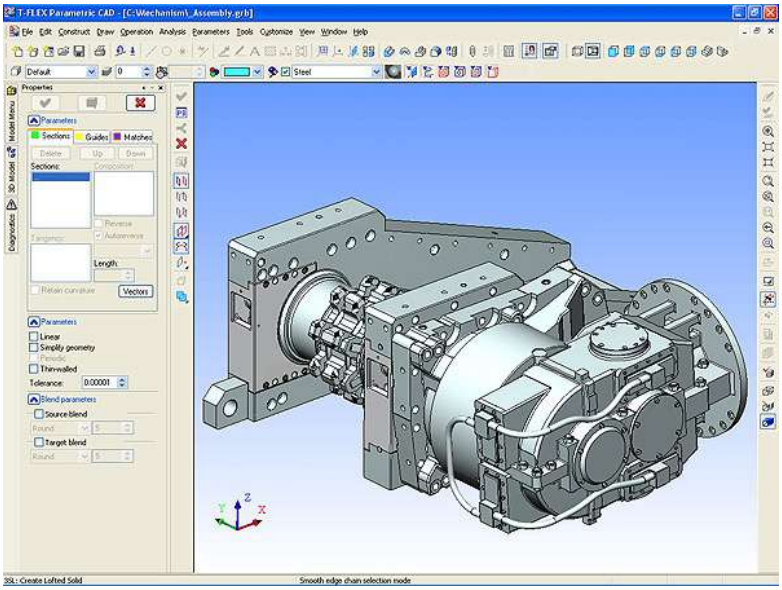




https://www.healthcareitnews.com/news/apac/nuhs-demonstrates-use-holographic-tech-brain-surgery

Aplicaciones

- Diseño e Ingeniería



Nuestro curso

- **Propósito**
  - Introducir a los estudiantes en la disciplina de computación gráfica.
  - Al término del curso, se espera que los estudiantes modelen, solucionen y visualicen computacionalmente problemas aplicados que involucren datos, geometrías y escenas complejas en 2D y 3D.
  - A modo de ejemplo se pueden desarrollar aplicaciones gráficas interactivas funcionales como videojuegos, simulaciones físicas, sistemas CAD, visualización arquitectónica, modelación de terrenos, entre otras aplicaciones posibles.

Nuestro curso

Metodología

Cátedras
- Visión amplia y fundamentos
- Estrategias para resolver problemas
- Análisis de código de ejemplo

Material de Apoyo
- Diapositivas
- Enlaces
- Bibliografía
- Código fuente de ejemplo
- Lecturas complementarias

Clases auxiliares
- Expositivas e interactivas resolviendo problemas tanto teóricos como prácticos
- Apoyo a tareas computacionales
- Participar activamente!
- No se puede explicar nuevamente toda la materia
- Se requiere conocimiento básico de contenidos vistos en cátedra

Carga Académica

- Evaluaciones
  - Cinco tareas computacionales
  - Cinco preguntas tipo control
  - Doce cuestionarios

- Nota final del curso
  - TA: promedio de tareas
  - PA: promedio de preguntas
  - CA: promedio de los mejores 10 cuestionarios

  - FINAL: 20% CA + 30% PA + 50% TA

  - Bono Tareas por asistencia a clases síncronas:
    - Asistencia 100% se premia con +0.3 a TA
    - Asistencia +70% se premia con +0.1 a TA

- Detalles sobre evaluaciones en documento “Reglas del Juego” en U-Cursos.

Bibliografía

- Códigos de ejemplo (fork del Prof. Daniel Calderón)

https://github.com/ivansipiran/grafica

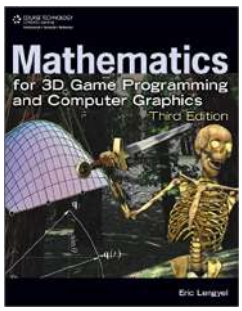
  - Instalar Git
    - Windows: https://git-scm.com/download/win
    - Ubuntu: sudo apt install git

  - Descargar el repositorio
    - Ejecutar el comando:
      >> git clone https://github.com/ivansipiran/grafica

    - Para actualizar el contenido de su carpeta, ejecutar:
      >> git pull

Bibliografía - Teoría

- Mathematics for 3D Game Programming and Computer Graphics
  - Third Edition
  - Eric Lengyel
  - 2012

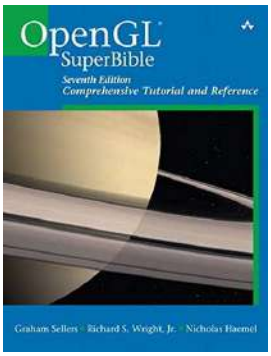


- Advanced Methods in Computer Graphics with examples in OpenGL
  - Ramakrishnan Mukudan
  - 2012
  - (Disponible digitalmente en catálogo de Bello)

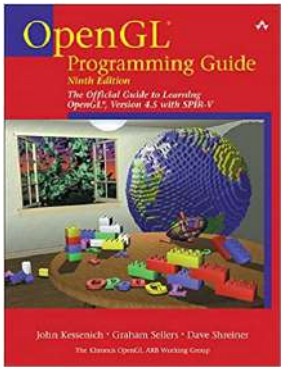
- Real-time rendering
  - Fourth edition
  - Tomas Akenine Muller, Eric Haines, Naty Hoffman

Bibliografía - OpenGL

- OpenGL SuperBible
  - Seventh Edition
  - Graham Seller, Richard S. Wriht, Nicholas Haemel
  - 2016

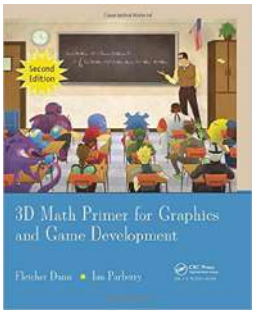


- OpenGL Programming Guide
  - Ninth edition
  - John Kessenich, Graham Sellers, Dave Shreiner
  - 2017

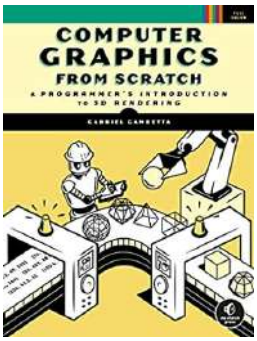


Bibliografía - OpenGL

- 3D Math Premier for Computer Graphics and Game Development
  - Second Edition
  - Fletcher Dunn, Ian Parbery
  - 2011
  - https://gamemath.com/

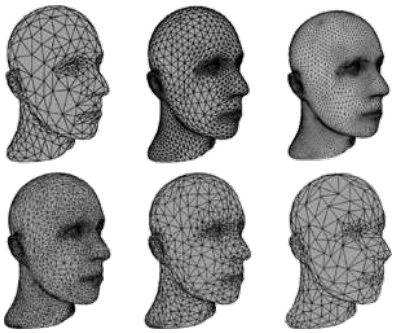


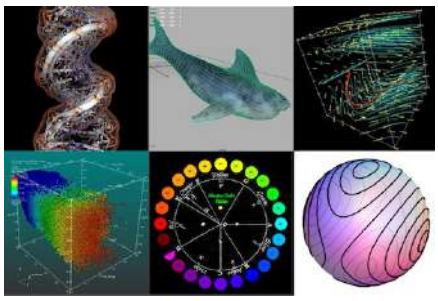
- Computer Graphics from Scratch
  - Gabriel Gambetta
  - 2020
  - https://gabrielgambetta.com/computer-graphics-from-scratch



CC3501 Roadmap

- Introducción a la Computación Gráfica
  - Computación gráfica en 2D
  - Pipeline de rendering
- Modelación geométrica
  - Transformación y jerarquías
  - Curvas, superficies y sólidos
- Computación gráfica en 3D
  - Proyecciones
  - Iluminación y sombreado
  - Texturas
- Visualización científica y simulaciones físicas
  - Visualización de campos escalares y vectoriales
  - Sistemas lineales y colisiones





Recursos

- http://www.opengl.org
- http://www.tomdalling.com/blog/category/modern-opengl/
- http://www.opengl-tutorial.org/
- https://en.wikibooks.org/wiki/OpenGL_Programming

Quién es el profe

- Profesor asistente en el DCC
  - Ingeniero Civil en Computación, FCFM, U. Chile
  - *Docteur en Informatique, Universite Joseph Fourier*, Francia
    - Creador del primer mundo virtual (MPEG-J) remoto
    - Creado del primer clone en video conferencias 3D
  - Perito, Asesor y con emprendimientos.
  - Esposo y Papá 😉

Color
Imagen y Tecnología de Display*

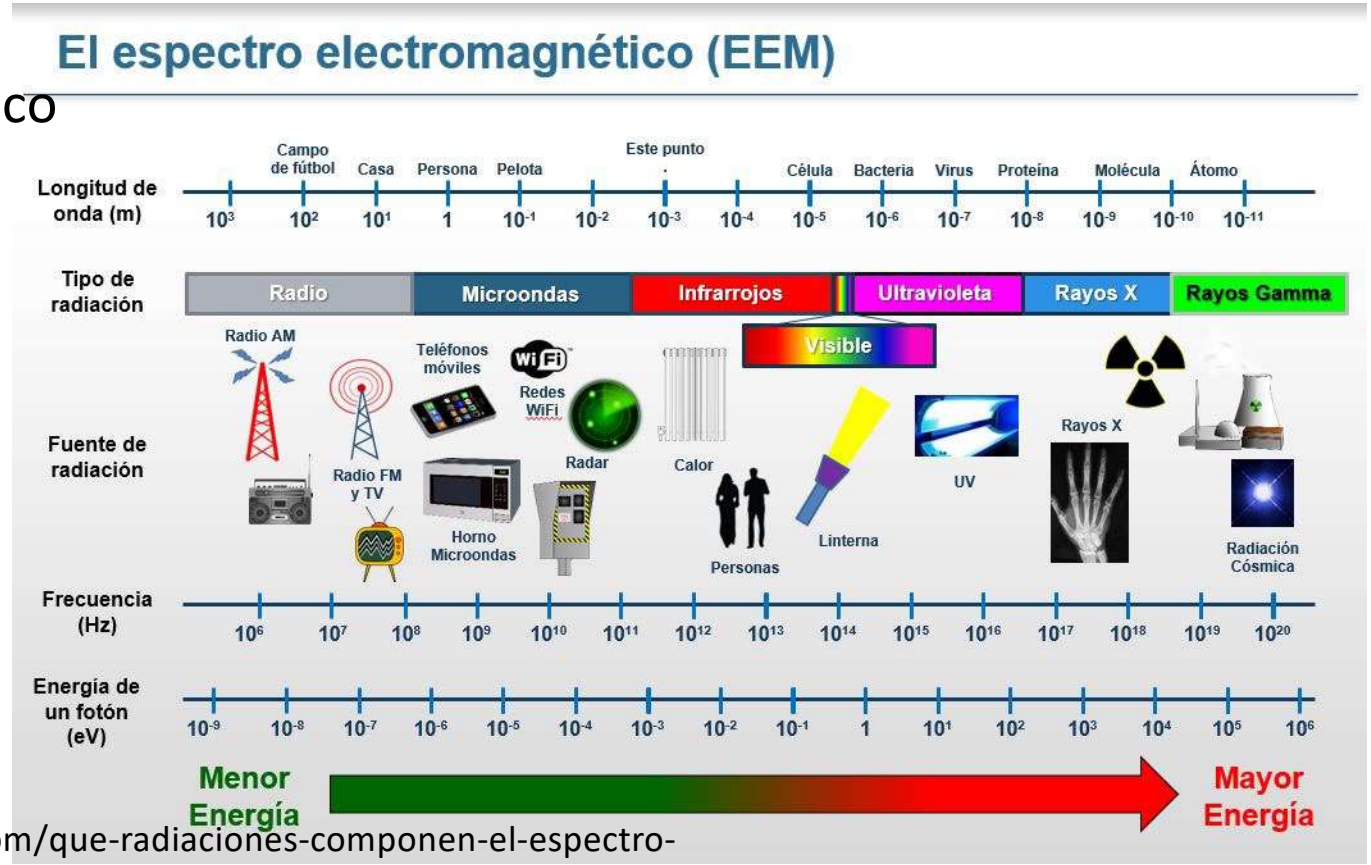
Patricio Inostroza, Dr.

dcc CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

*Material creado originalmente por Dr. Ivan Sipiran

Espectro Electromagnético

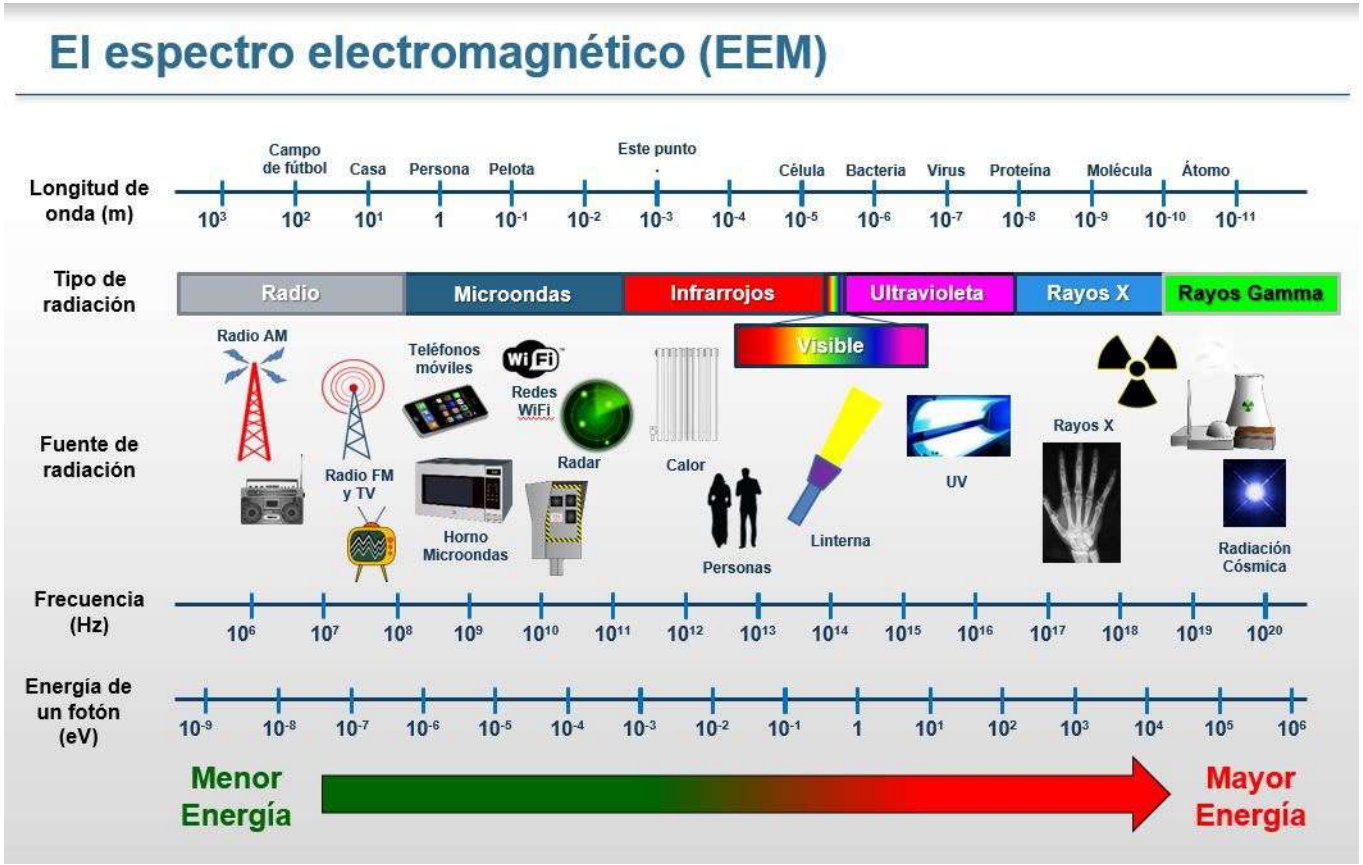
- Espectro Electromagnético relativamente grande
- Rango visible pequeño
  - 400nm (azul) a 700nm (rojo)



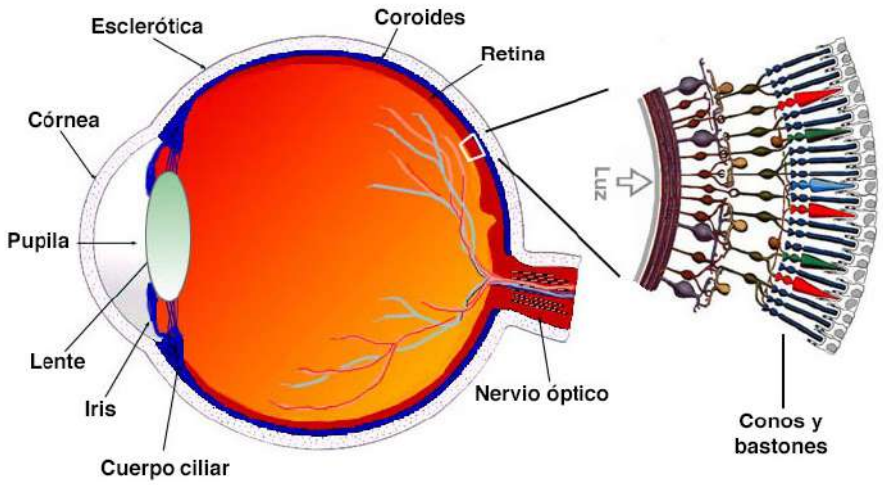
https://www.arizaadministraciones.com/que-radiaciones-componen-el-espectro-electromagnetico-cuales-y-como-afectan-a-la-salud-y-por-que-no-debemos-tenerles-miedo/

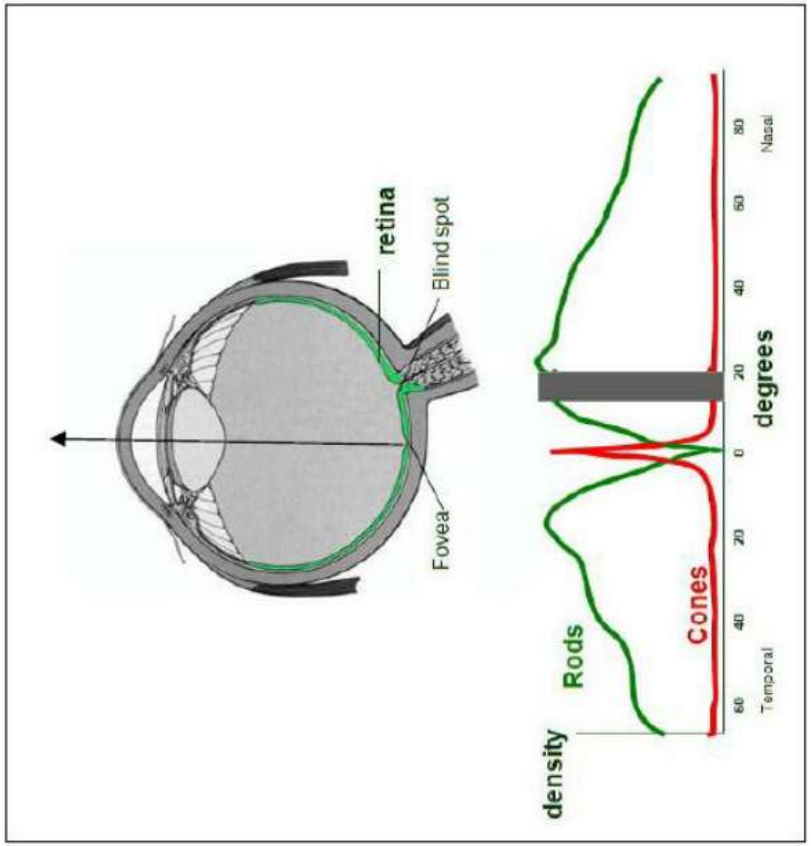
Espectro Electromagnético

- ¿qué pasaría si pudiésemos ver todo el espectro electromagnético?



¿Quién/qué percibe el color ?

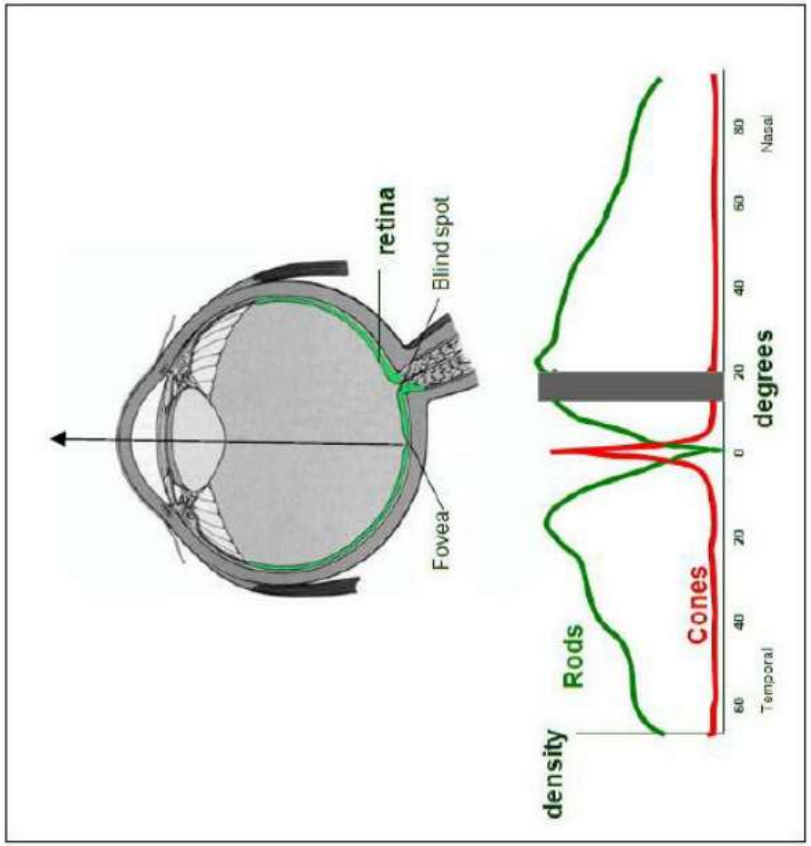




¿Quién/qué percibe el color ?

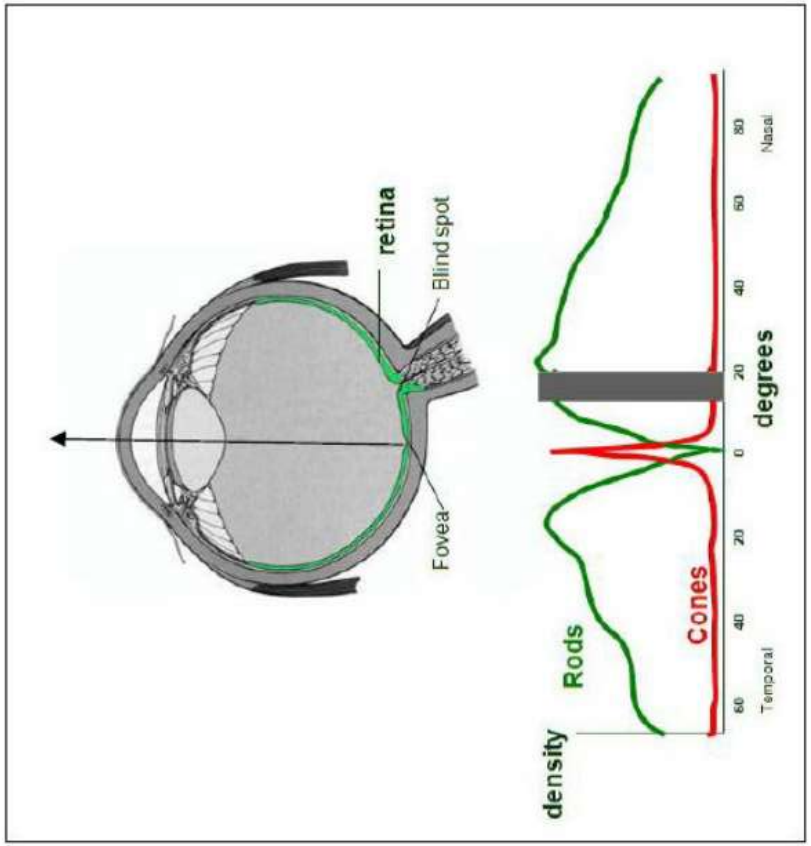
- Bastones:
  - Condiciones de baja iluminación
  - Se activan con la energía de un solo fotón
  - Muy buenos para distinguir formas y contrastes

- Conos:
  - Perciben los colores
  - Trabajar con alta luminosidad
  - Sensibles a la luz roja, a la luz azul y a la

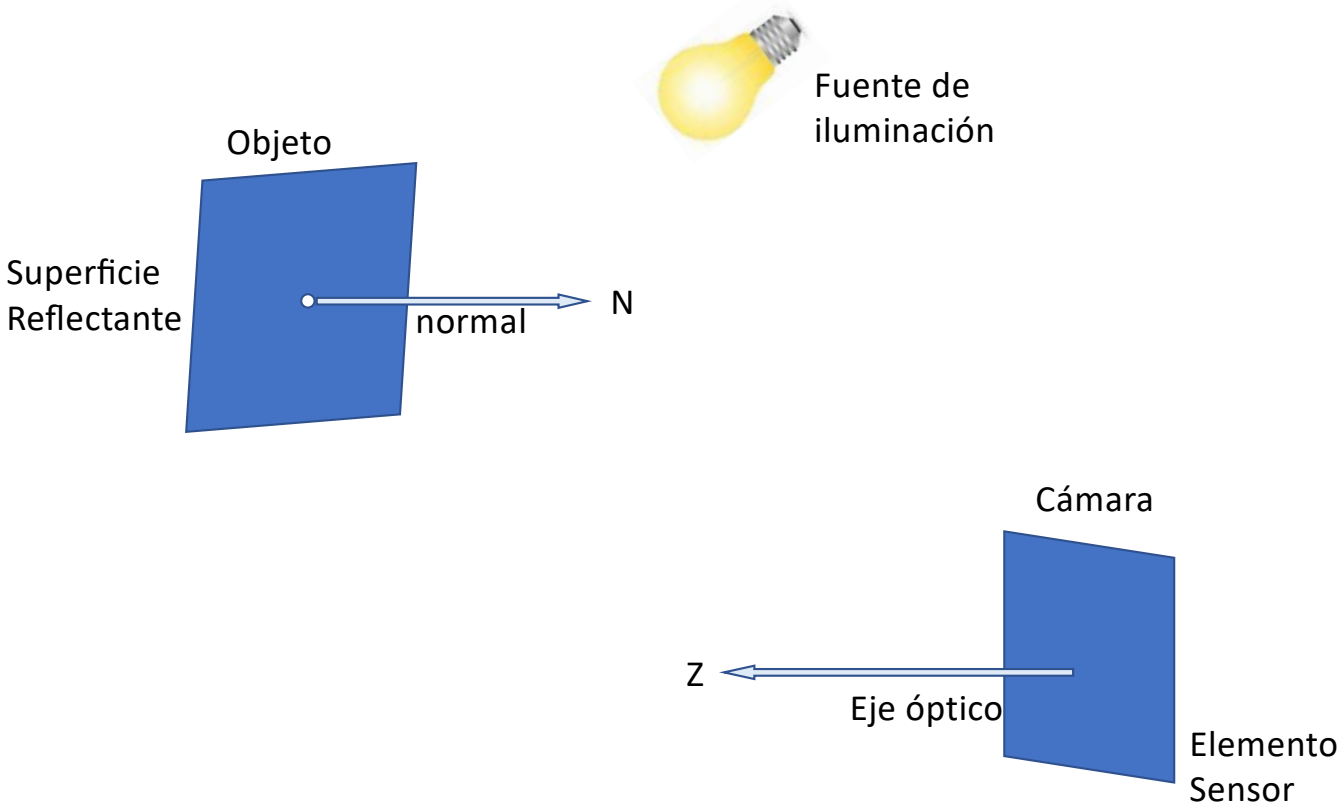


¿Quién/qué percibe el color ?

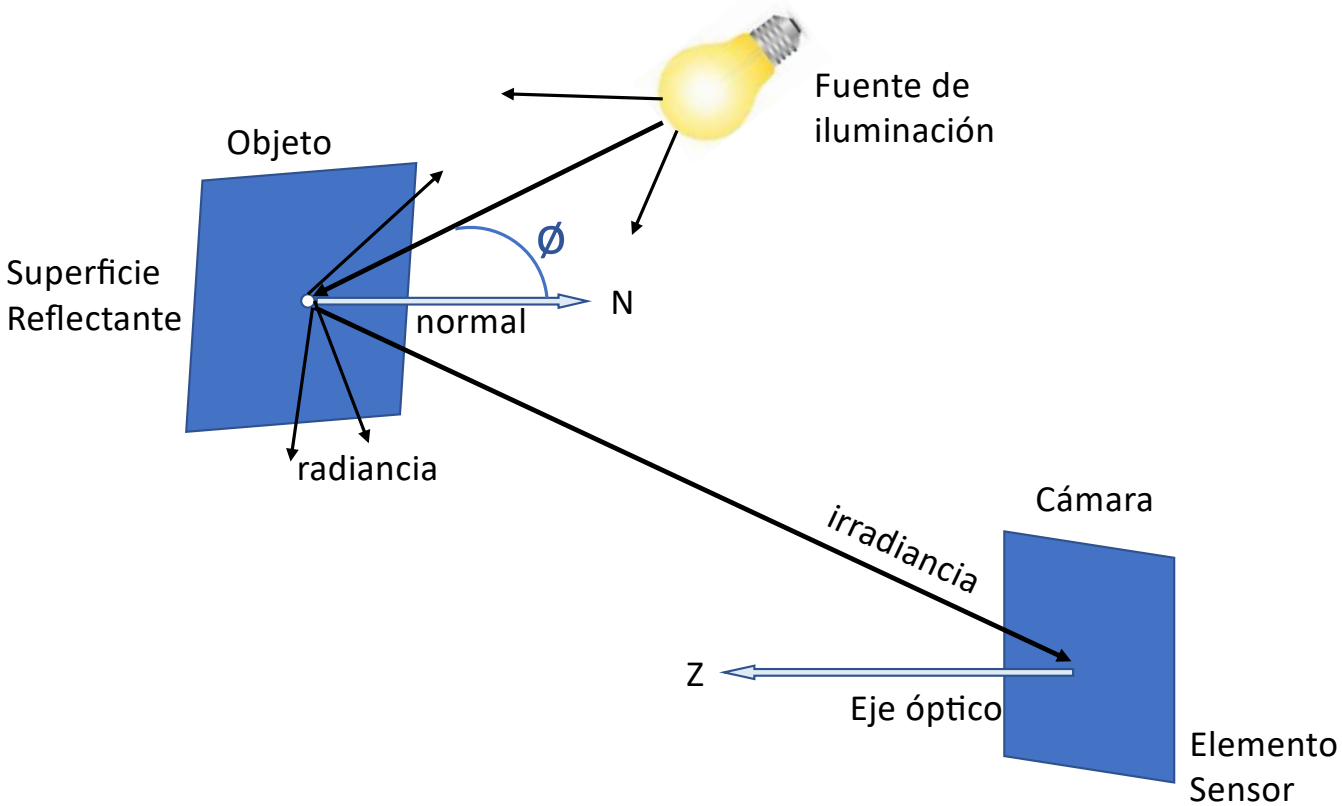
- ¿Por qué Homero **no** puede ver las estrellas en la noche?



iluminación



iluminación



iluminación

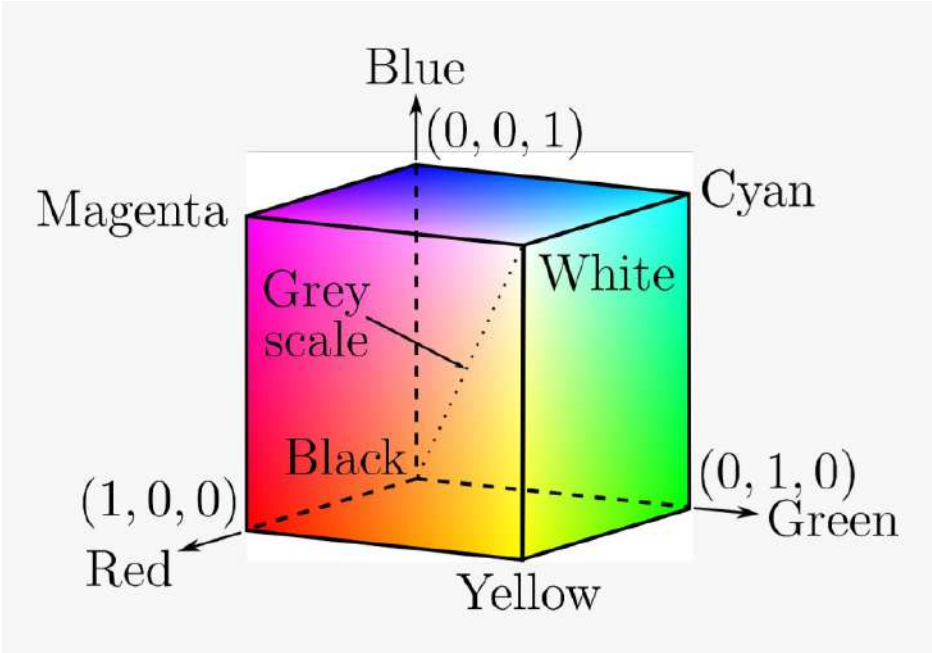
- **Luz:** el espectro de energía que ilumina la superficie del objeto
- **Reflectancia**: proporción de luz reflejada con respecto a luz entrante
- **Especularidad**: Especular vs. Mate
- **Distancia**: Distancia a la fuente de luz
- **Ángulo**: Ángulo entre la normal de la superficie y la fuente de luz
- **Sensitividad**: Cuán sensible es el sensor

Modelos del color

- **RGB** : Sistema aditivo; usado en pantallas
- **CMY[K]:** Sistema sustractivo ; usado en impresoras)
- **HSV:** eEspacio perceptual usado en arte, psicología y reconocimiento
- **YIQ:** usado para TV es bueno para compresión

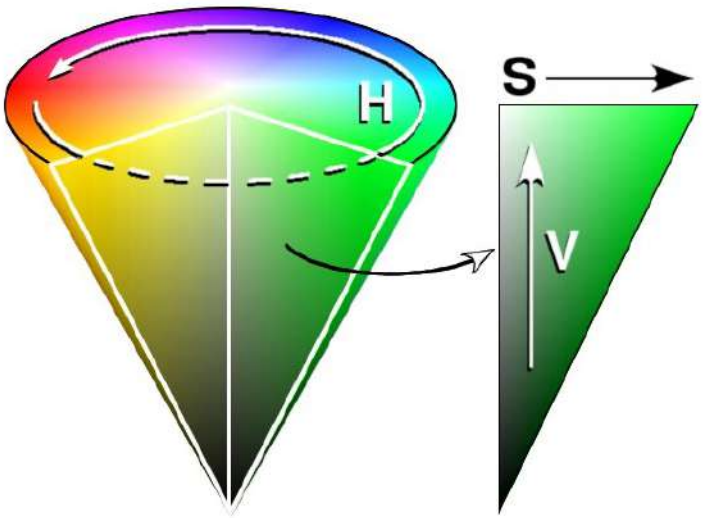
Cubo RGB

- Valores RGB normalizados a 0 – 1
- Gris son valores en la diagonal
- Colores "puros" en las esquinas



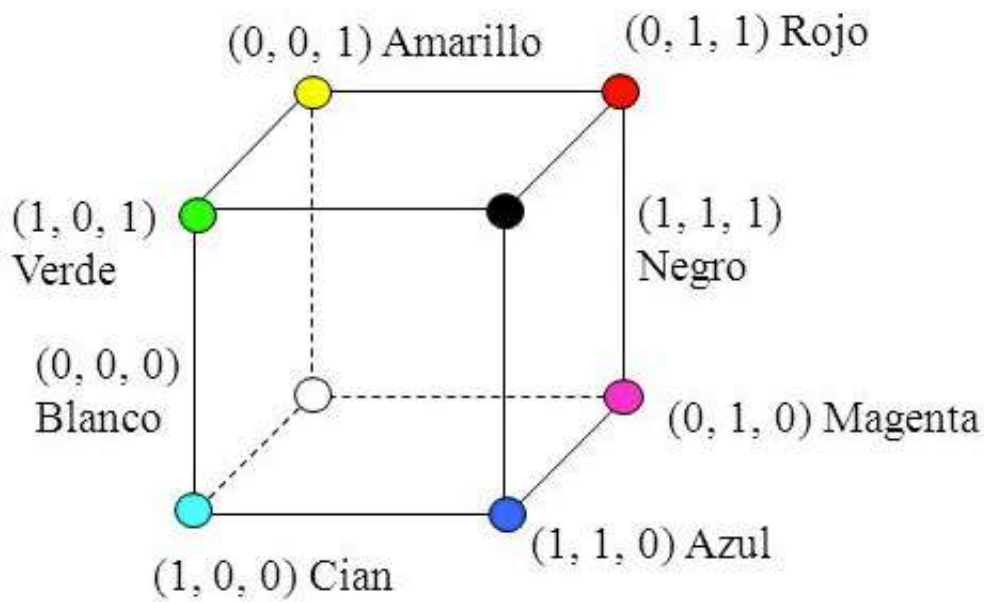
HSV: Hue, Saturation, Value

- Color codificado relativo a la diagonal del cubo de color.
- H: Es codificado como un ángulo.
  - 0 es rojo, 60 es amarillo y 120 es verde.
- S: Saturación es la distancia desde la diagonal
- V: Intensidad es la altura.



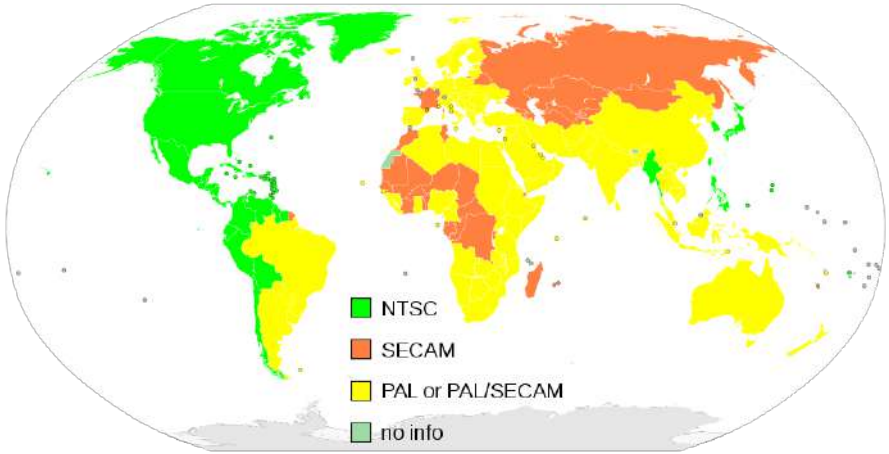
Modelo CMYK: Cyan, Magenta, Yellow, Black

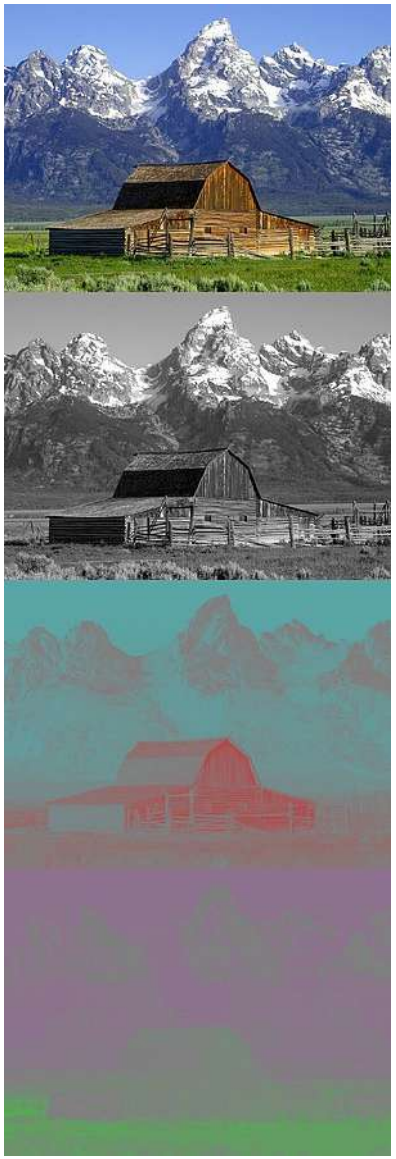
- Modelo sustractivo
- Color blanco en (0, 0, 0)
- Reflejo de RGB
- Se incorpora el negro



Modelo YIQ

- YIQ: Espacio de color, usado por el estándar de televisión NTSC
- Y representa la información de luminancia
  - Utilizado por los TV blaco y Negro
- I y Q representan la información de crominancia.





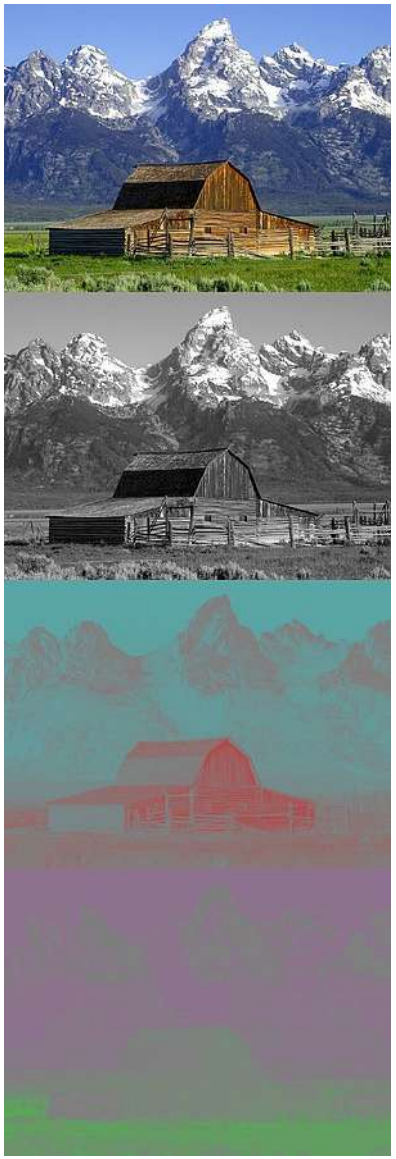
Modelo YIQ

De RGB a YIQ

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0,299 & 0,587 & 0,114 \\ 0,595716 & -0,274453 & -0,321263 \\ 0,211456 & 591 & 0,311135 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

De YIQ a RGB

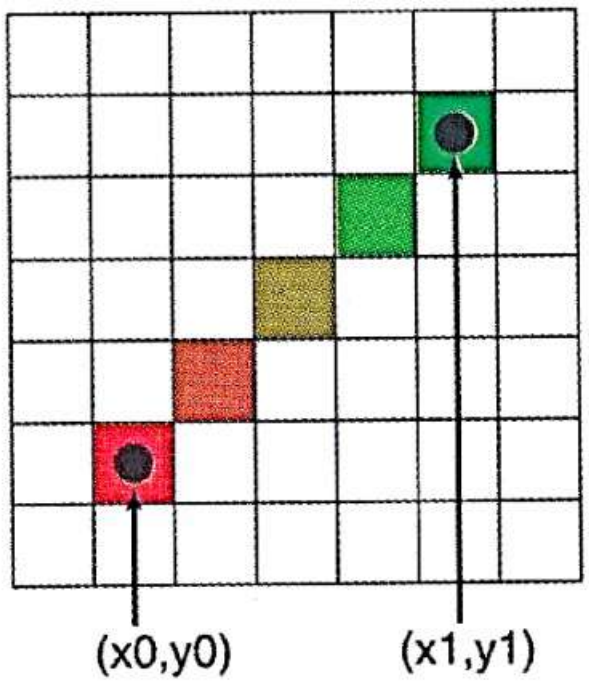
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0,9563 & 0,6210 \\ 1 & -0,2721 & -0,6474 \\ 1 & -1,1070 & 1,7046 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$



Todos los caminos llegan a … RGB

- Todos los modelos de color tiene una transformación a RGB
- ¿Por qué?

Interpolación de colores



(r,g,b) = (0.00, 1.00, 0.00), t = 1.00

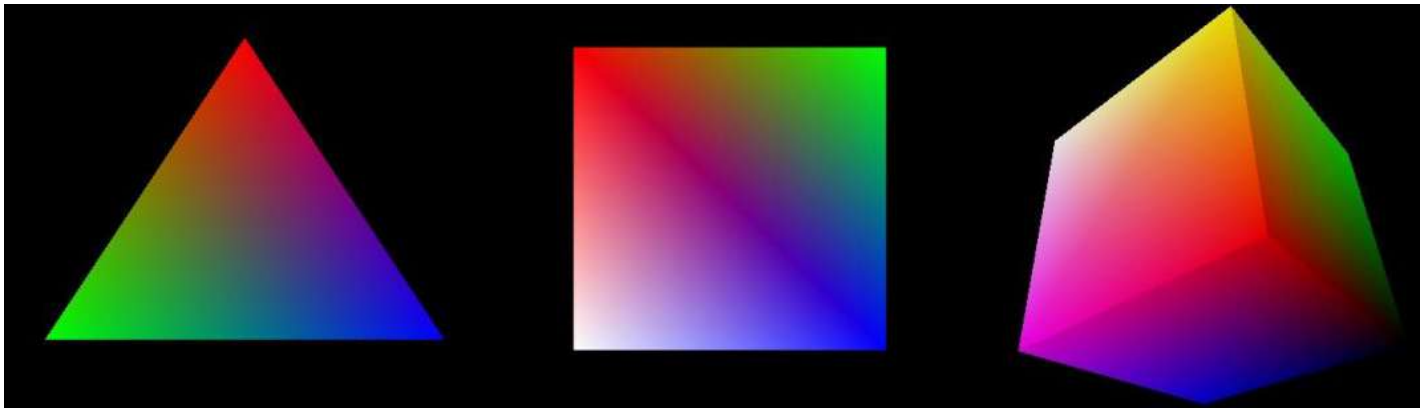
(r,g,b) = (0.25, 0.75, 0.00), t = 0.75

(r,g,b) = (0.50, 0.50, 0.00), t = 0.50

(r,g,b) = (0.75, 0.25, 0.00), t = 0.25

(r,g,b) = (1.00, 0.00, 0.00), t = 0.00

Interpolación de colores



Color en el computador

- Hay que almacenar cada color
- ¿cuánto espacio?
  - n bits por color
  - Hoy: 8 bits por color
- ¿cuál color?
  - Red
  - Green
  - Blue
- 24 bits

Color en el computador

- Hay que almacenar cada color
- ¿cuánto espacio?
  - n bits por color
  - Hoy: 8 bits por color
- ¿cuál color?
  - Red
  - Green
  - Blue
- 24 bits

- ¿es suficiente 24 bits?

El canal Alfa

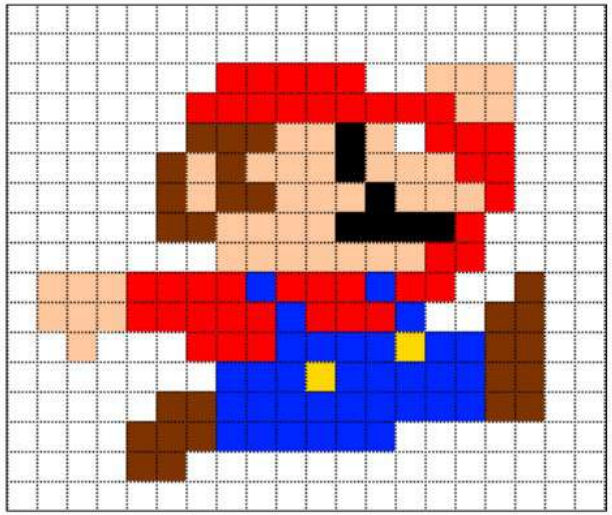
- Cada píxel necesita
  - Rojo: 8 bits
  - Verde: 8 bits
  - Azul: 8 bits
  - Alfa: 8 bits
  - **Total: 32 bits**

$$c = \alpha c_f + (1 - \alpha) c_b$$

foreground background

Imagen Raster

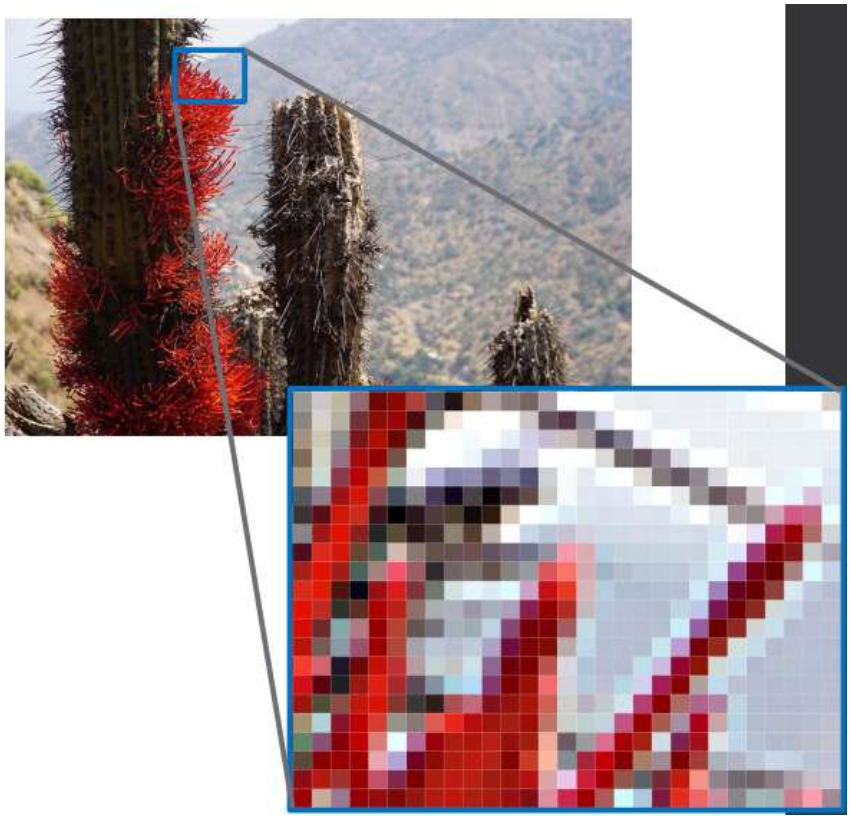
- Una matriz 2D
- Cada celda tiene asignado un color
- Cada celda se conoce como píxel
- Este modelo de visualización se conoce como ***raster***



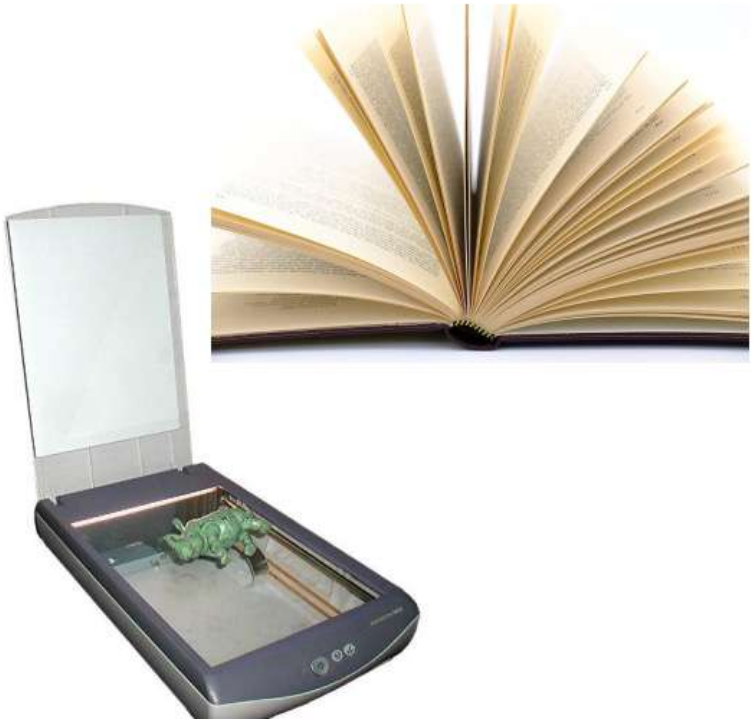
Créditos: Nintendo

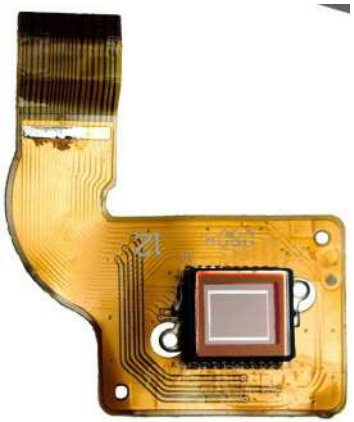
Imagen Raster

- Modelo simple
  - Un píxel por celda
- Resolución fija
  - Se pierde nitidez al hacer zoom
- Costoso de almacenar
  - Distintas estrategias de compresión



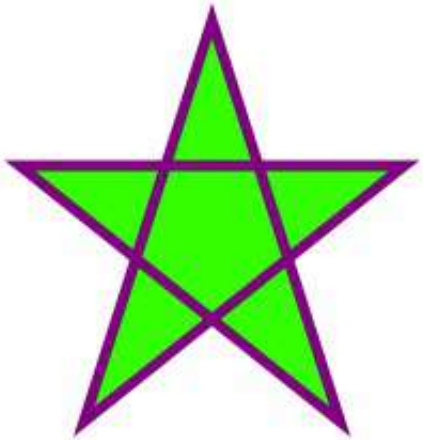
Dispositivos Raster





Sensor CMOS

Imágenes vectoriales



- Modelos paramétricos por cada figura representada
  - Líneas, cuadrados, círculos, curvas, etc
- Resolución infinita
  - No hay pérdida de nitidez al hacer zoom
- Tamaño depende de la complejidad del contenido

```
<svg height="210" width="500">
  <polygon points="100,10 40,198 190,78 10,78 160,198"
  style="fill:lime;stroke:purple;stroke-width:5;fill-rule:nonzero;" />
</svg>
```

https://www.w3schools.com/graphics/svg_polygon.asp

Formatos de Imágenes

- Vectoriales
  - SVG
  - EPS
  - VRML

- Formatos Raster
  - BMP
  - JPG
  - PNG
  - GIF

Aclaraciones…

- pdf: ¿es un formato vectorial?
  - "El PDF es ahora un estándar abierto, reconocido por la Organización Internacional para la Estandarización (ISO). Los documentos PDF pueden contener vínculos y botones, campos de formulario, audio, vídeo y lógica empresarial. Se pueden firmar electrónicamente y puedes ver con facilidad los archivos PDF en Windows o Mac OS utilizando el software gratuito Acrobat Reader DC." https://www.adobe.com/cl/acrobat/about-adobe-pdf.html

- Vectorial: resolución infinita…
  - Mandelbrot
  - https://math.hws.edu/eck/js/mandelbrot/MB.html

Arquitecturas de los Sistemas Raster Scan

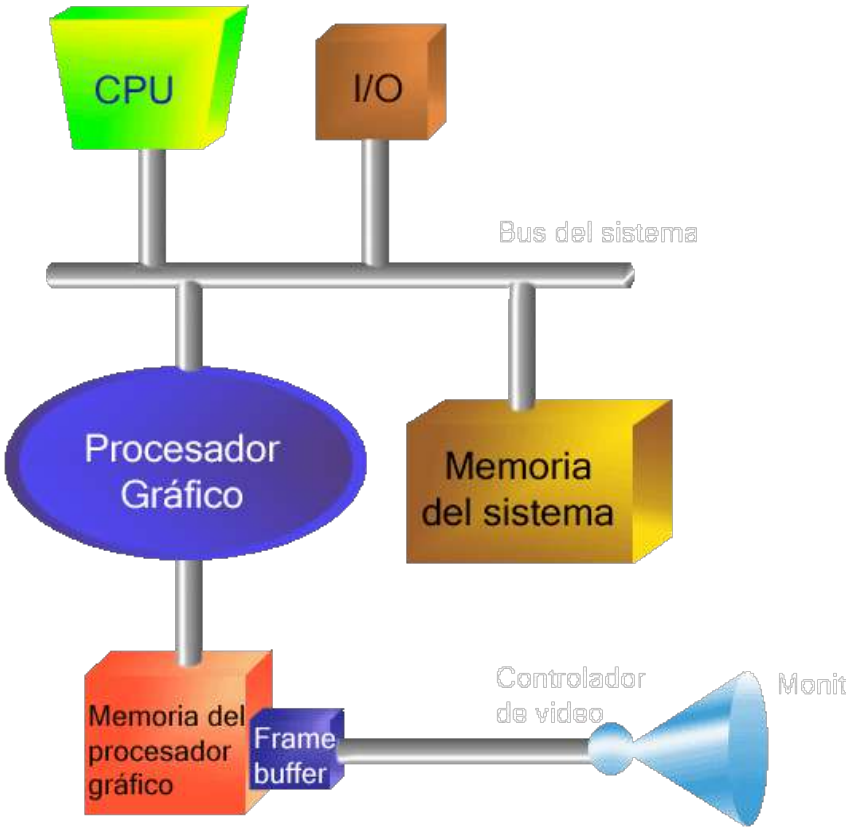
Patricio Inostroza, Dr.

dcc CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Sistema gráfico de barrido (raster)

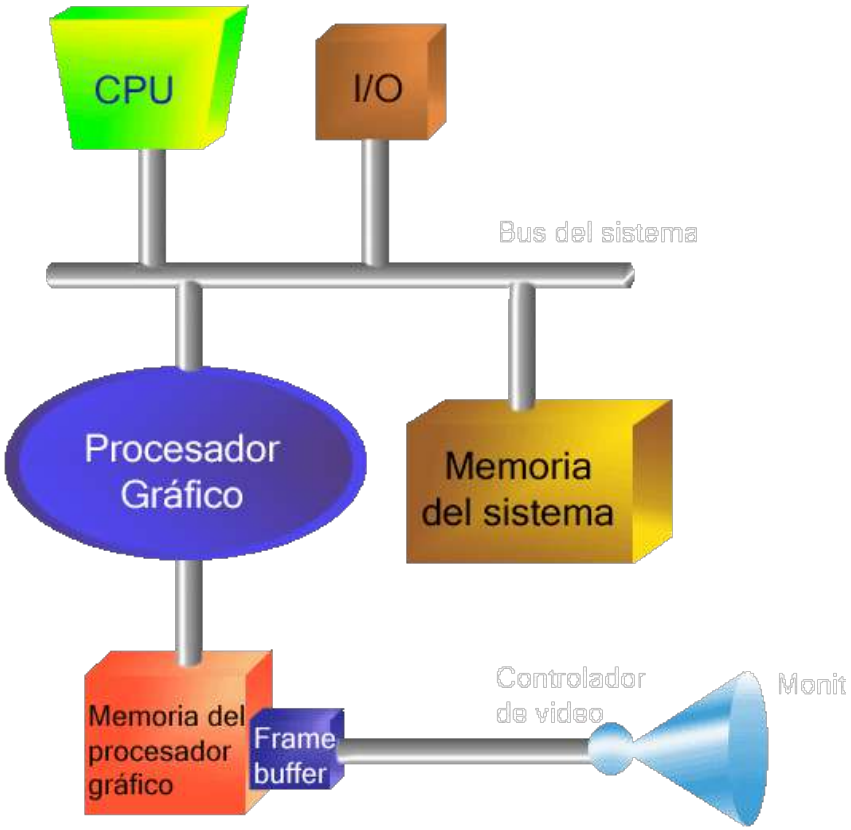
- Arquitectura
  - CPU y memoria principal .
  - Usados por todos los programas

- Un procesador especializado en gráfico:
  - Tiene su propia memoria
  - Mayor eficiencia



Sistema gráfico de barrido (raster)

- GPU: Procesador Gráfico
  - Calcula la siguiente imagen
  - Usan la memoria de la GPU
- Frame buffer
  - Contiene la imagen actual
  - Un controlador cambia el contenido
- Frame rate:
  - Frecuencia de cambio de imagen



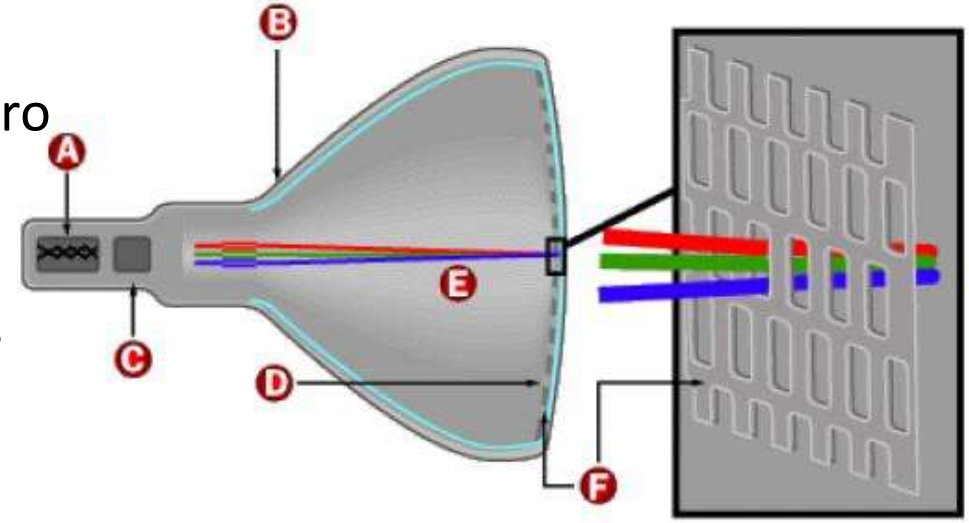
Doble buffer

- Un controlador de video muy rápido puede desplegar una imagen inconclusa

- Solución:
  - usar dos buffer
    - Uno para mostrar la imagen
    - Otro para crear la imagen
  - Creada la imagen, los roles se invierten

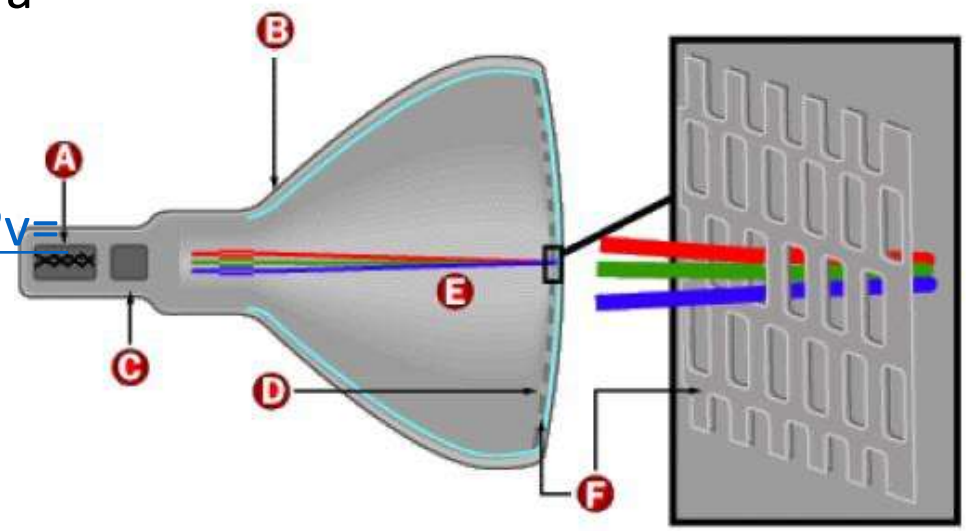
Sistema gráfico de barrido (raster)

- Monitores con tubo de rayos catódicos
- Pantalla de vidrio con una cubierta de diferentes compuestos de fósforo rojo, verde, azul
- El color aparece por la excitación producida por el haz de electrones



Sistema gráfico de barrido (raster)

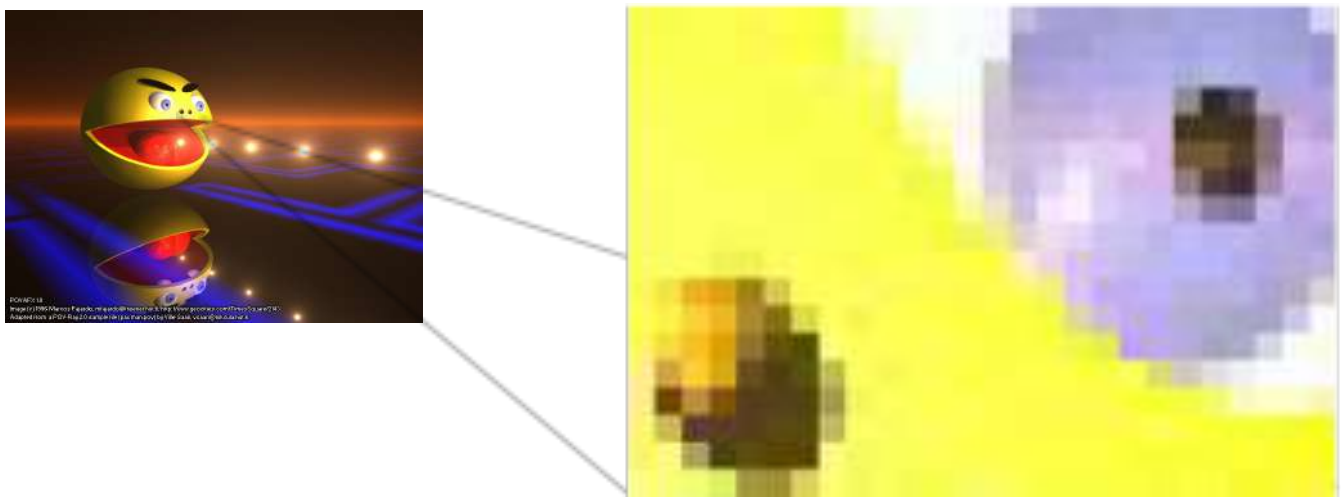
- La excitación no es continua
- El haz debe recorrer la pantalla para mantener la imagen
- Frecuencia de refresco
- https://www.youtube.com/watch?v=3BJU2drrtCM&t=463s



- 1:50 ; 7:40

Sistema gráfico de barrido (raster)

- Pixel
  - Pixel != Grupo de fósforos
  - Pixel : Número de puntos distinguibles. Depende de la resolución

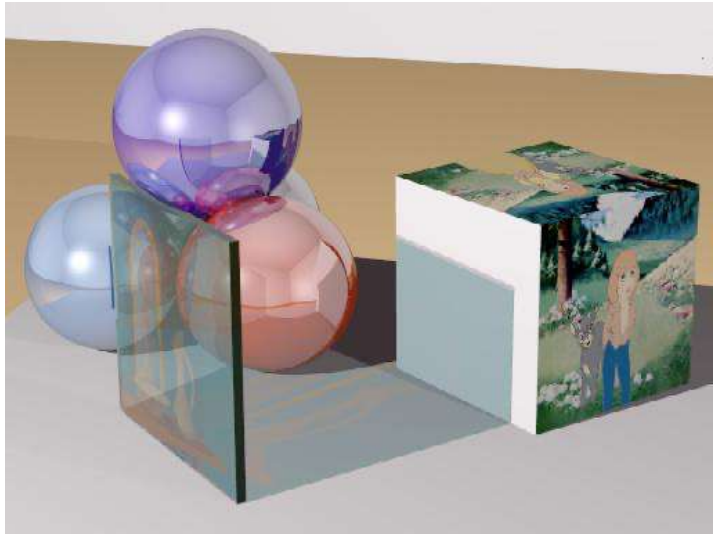


Frame buffer

- Memoria que almacena el color de cada pixel de la pantalla
- Ejemplo:
  - 8 bits por color
  - 3 bytes por pixel
  - Resolución 640 x 480 → 900 Kbytes
  - Más resolución → más memoria

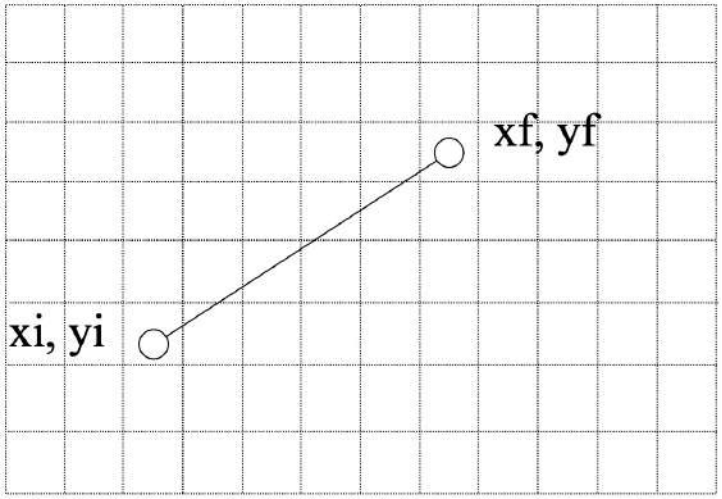
Componente Alfa

- La componente alfa permite definir la transparencia del objeto
- Es descrita mediante un byte
- Un color requerirá 4 bytes: 3 para el color y 1 para la transparencia



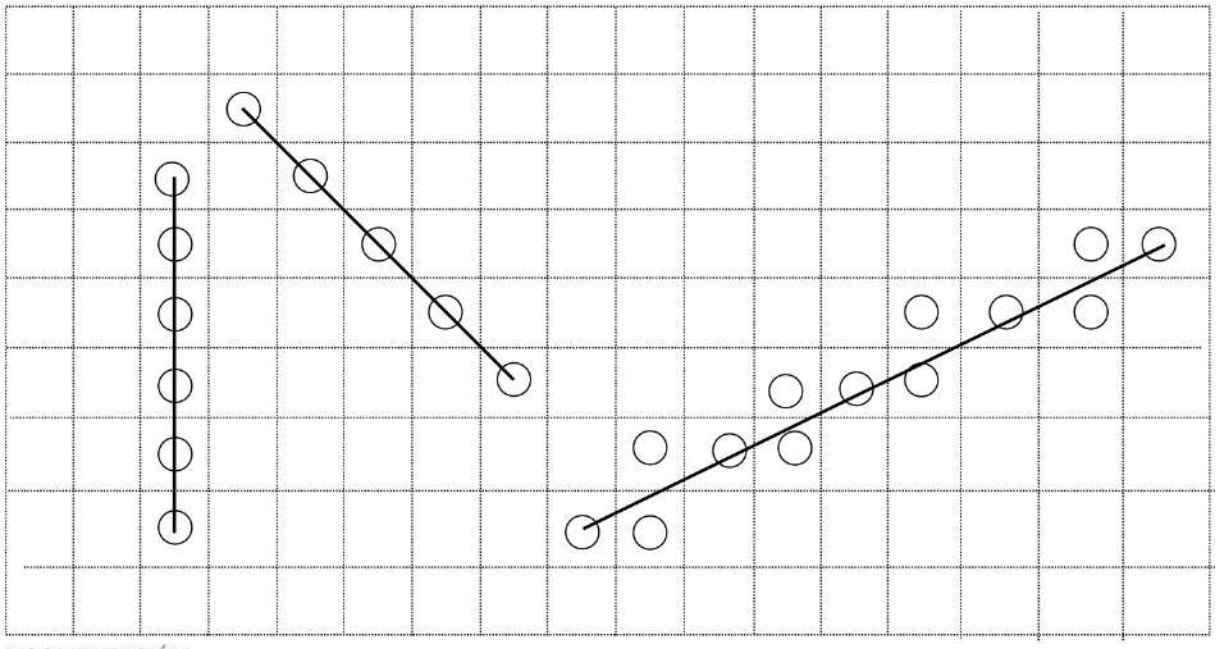
Pantalla raster

- Arreglo 2D de pixeles
- Pixeles centrados en coordenadas enteras
- **setPixel(x, y)**: función para iluminar un pixel



Segmento de línea

- Dado los puntos extremos de un segmento
  - ¿ Cómo identificar los pixeles que lo representan ?

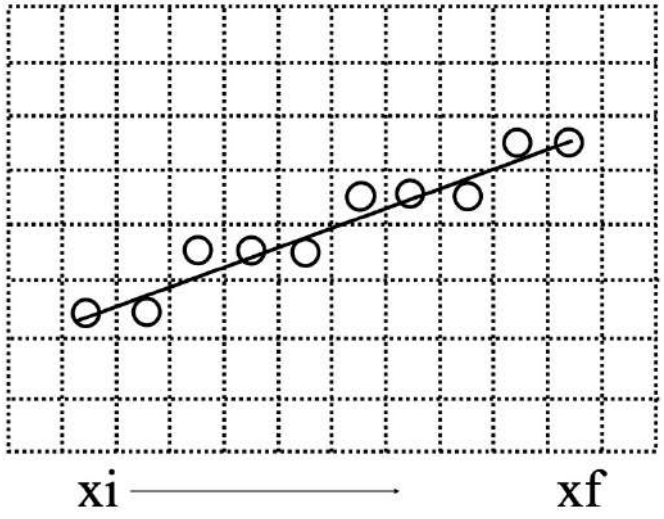


Segmento de línea

- Algoritmo “ingenuo”
  - Extremos del segmento: (xi, yi) – (xf, yf)
  - Sea f la función que describe el segmento: y = f(x)

    Para cada x en [xi, xf],

    setPixel(x, round(f(x))

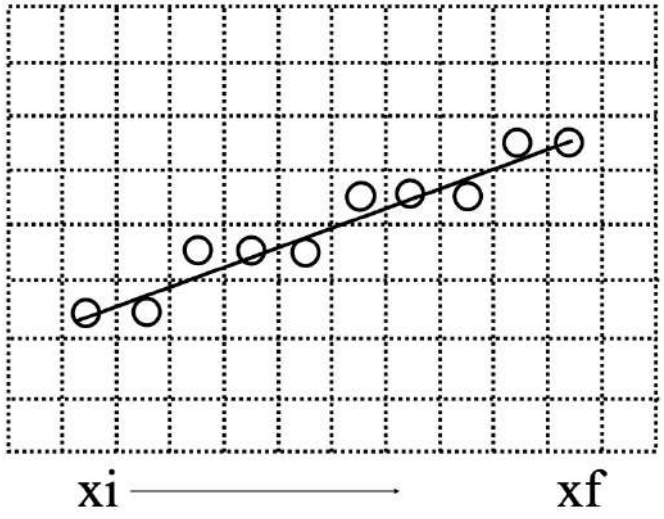


Segmento de línea

- Algoritmo “ingenuo”
  - Extremos del segmento: (xi, yi) – (xf, yf)
  - Sea f la función que describe el segmento: y = f(x)

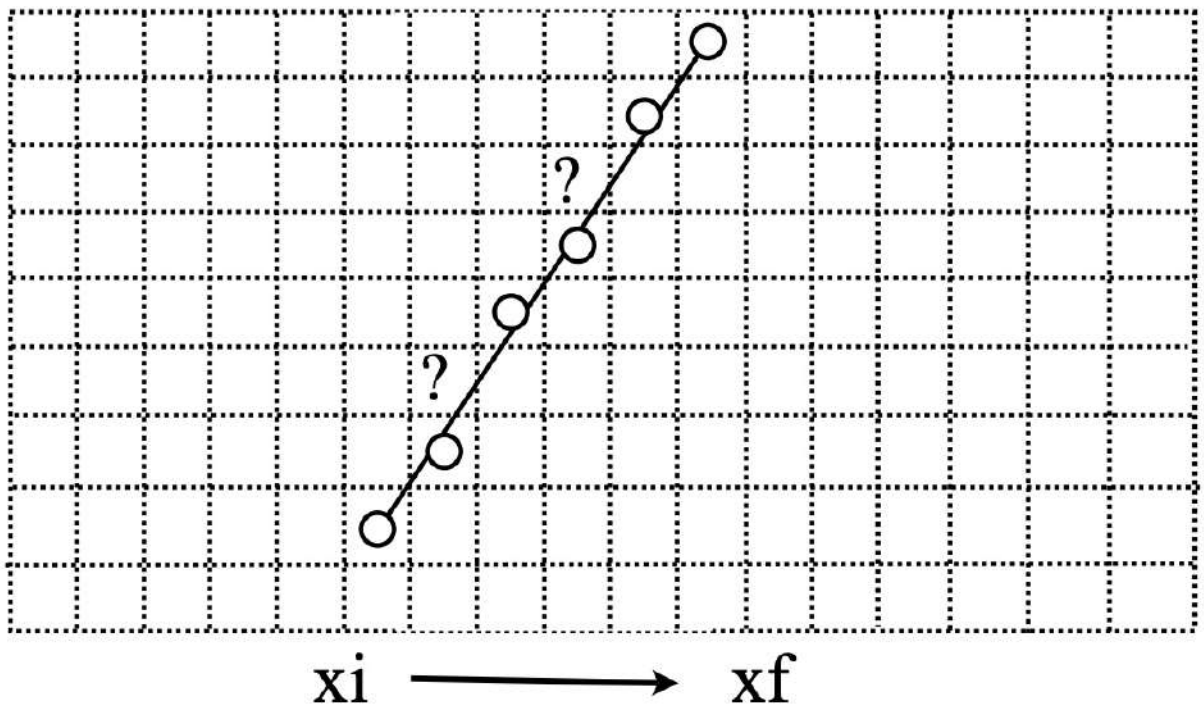
    Para cada x en [xi, xf],

    setPixel(x, round(f(x))

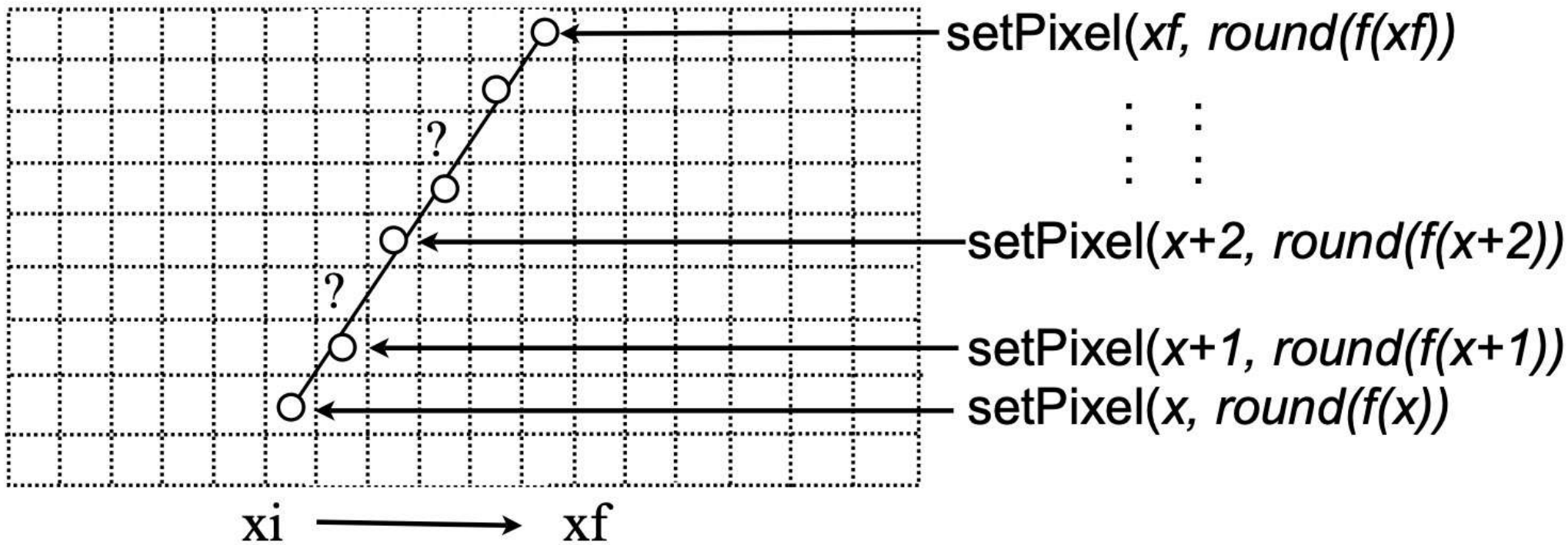


- ¿ Problema de este algoritmo ?

Algoritmo Ingenuo



Algoritmo Ingenuo



Algoritmo Incremental

$y_{k+1} = f(y_k)$

$x = x_i$; $y = y_i$

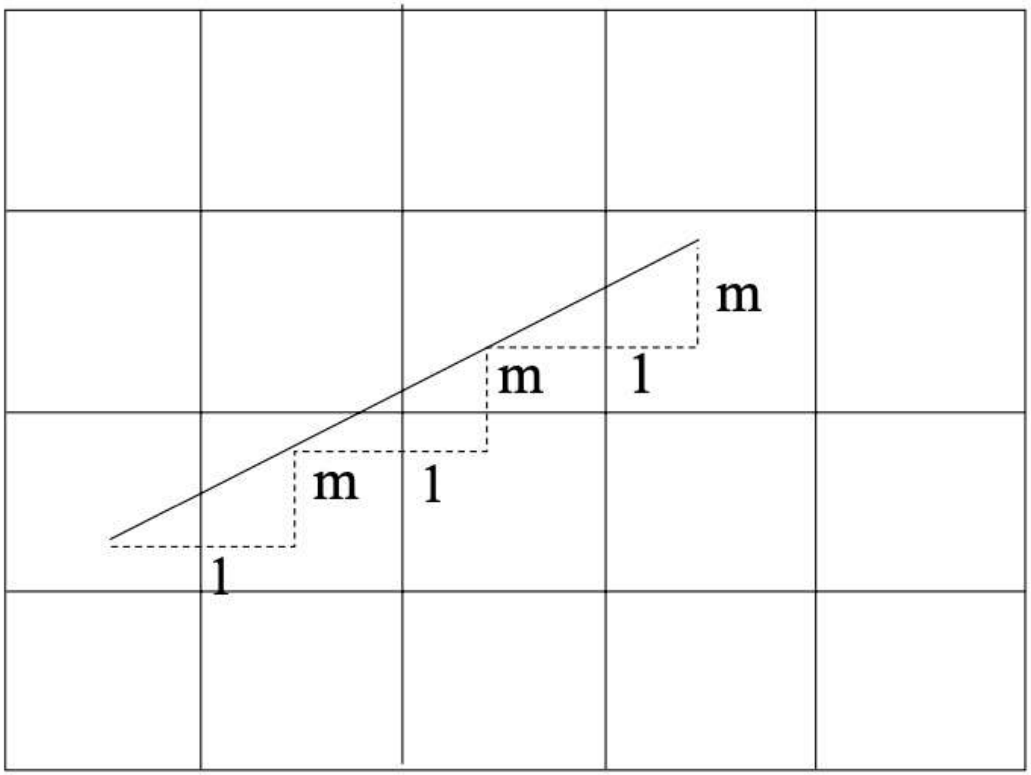
setPixel(x, round(y))

mientras $x < x_f$

  $x = x + 1$

  $y = y + m$

  setPixel(x, round(y))



¿ Problema de este algoritmo ?

Propuesto

- Qué es el algoritmo de Bresenham ?

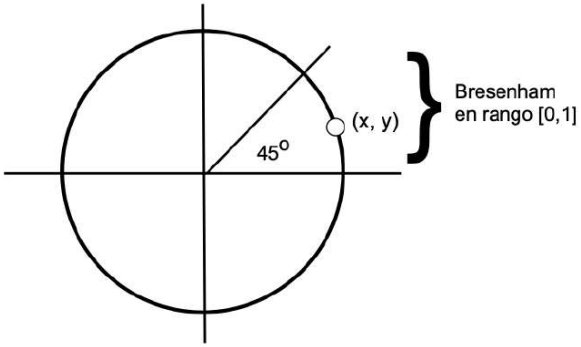
Bresenham

Patricio Inostroza, Dr.

dcc CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Dibujar un segmento de recta

- Bresenham
  - Un eficiente algoritmo para dibujar una línea en una pantalla raster (compuesta de pixeles).



- Características:
  - Usa aritmética entera
  - Supone la pendiente (m) en el rango [0, 1]
  - Parte del punto inferior-izquierdo al punto superior-derecho

- Referencia
  - Bresenham, J.E. "*Algorithm for Computer Control of a Digital Plotter*". IBM Systems Journal, 4(1), pp. 25-30, 1965.

Segmento de Recta

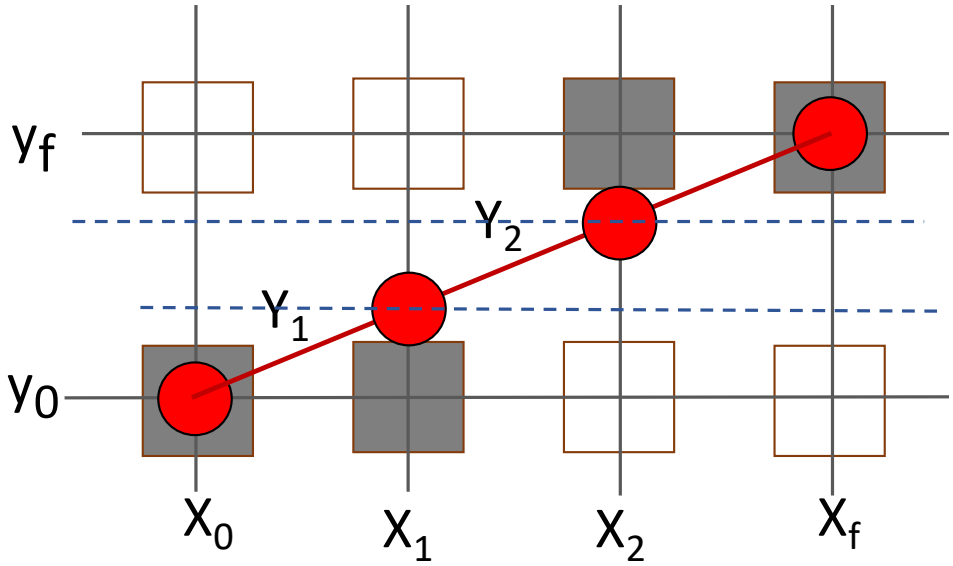
- Ecuación de la recta:
  - $y = f(x) = m*x + c$

- Puntos de la recta (valor exacto):
  - $(x, f(x)) = (x, m*x + c)$

- Punto de la recta (valor en pantalla):
  - $(x, \text{round}(f(x))) = (x, \text{round}(m*x + c))$

Segmento de Recta

- : Pixel apagado
- : Pixel encendido
- : Punto de la recta (real)



Segmento de Recta

- Dado que:

  $x_2 = x_1 + \Delta x$

  $y_1 = m * x_1 + b$

  $y_2 = m * x_2 + b$

- Luego

  $y_2 = m * x_2 + b$

  $y_2 = m * (x_1 + \Delta x) + b$

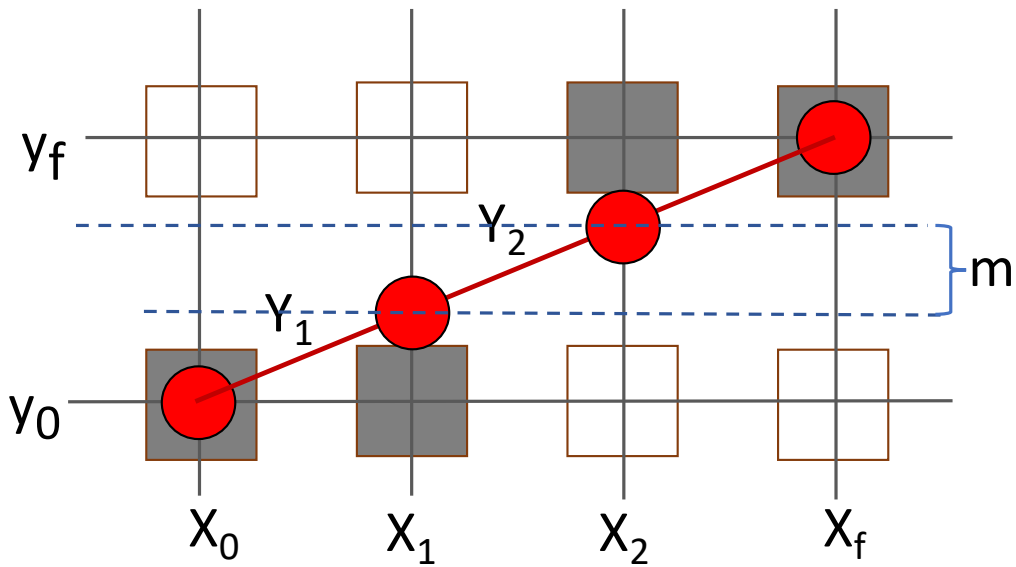
  $y_2 = m * x_1 + m * \Delta x + b$

- Pero

  $\Delta x = x_2 - x_1 = 1$; distancia unitaria

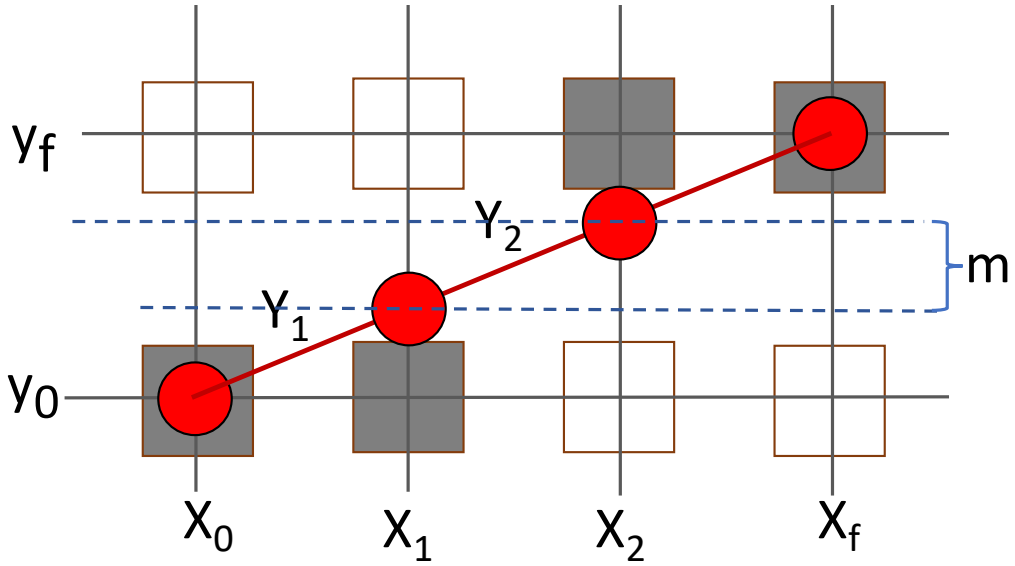
- Luego
  - $y_2 = m * x_1 + b + m$

    $y_2 = y_1 + m$



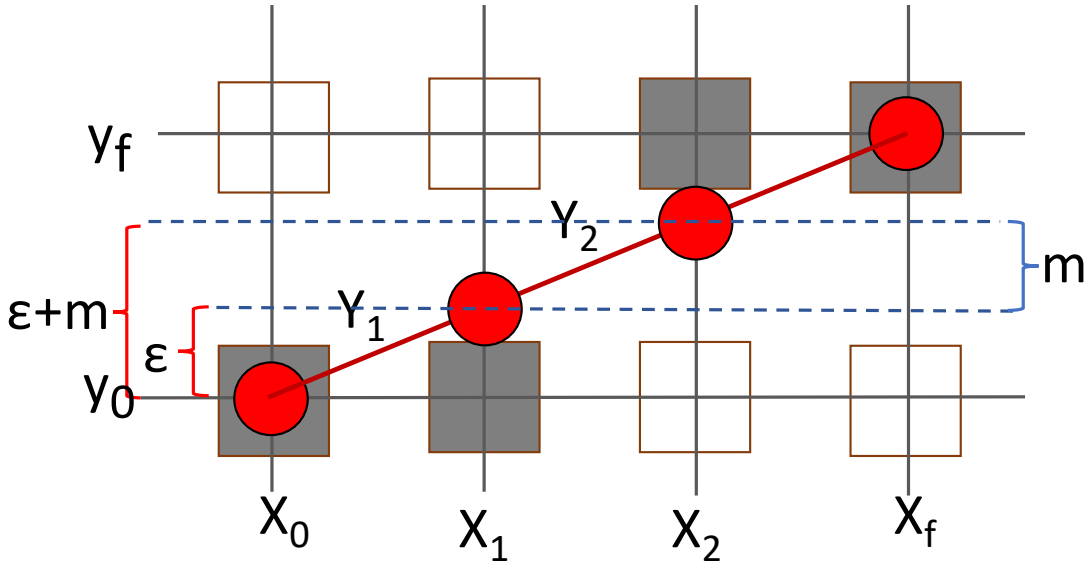
Segmento de Recta

- : Pixel apagado
- : Pixel encendido
- : Punto de la recta (real)
- m : $(y_f - y_0)/(x_f - x_0) = d_y/d_x$



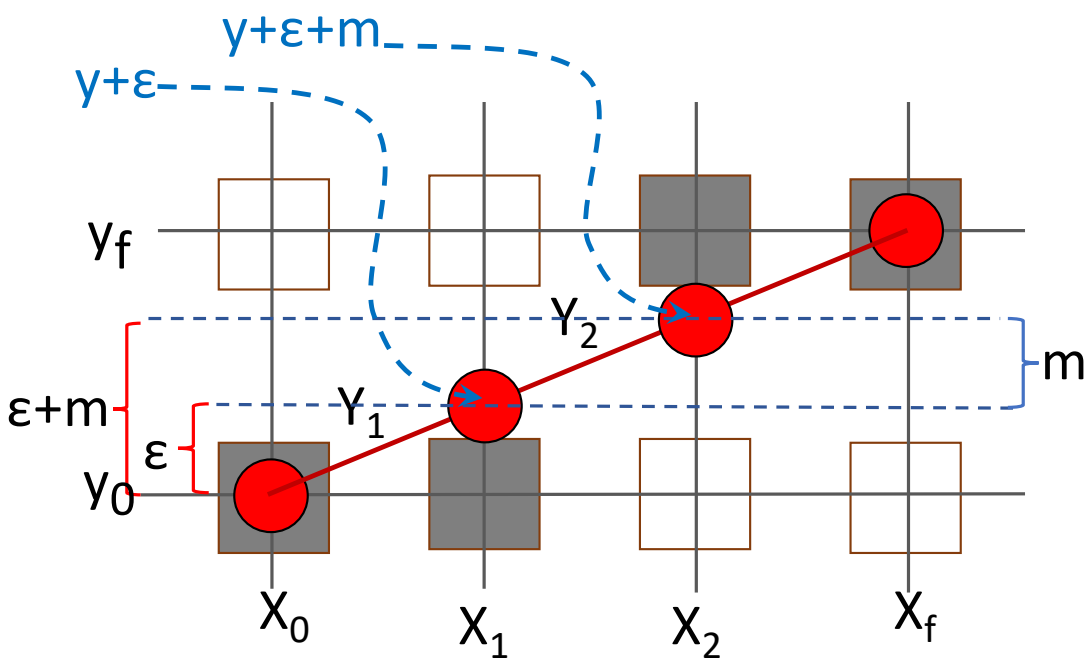
Segmento de Recta

- : Pixel apagado
- : Pixel encendido
- : Punto de la recta (real)
- m : $(y_f - y_0)/(x_f - x_0) = d_y/d_x$
- ε : "error" (distancia) a la ordenada del pixel



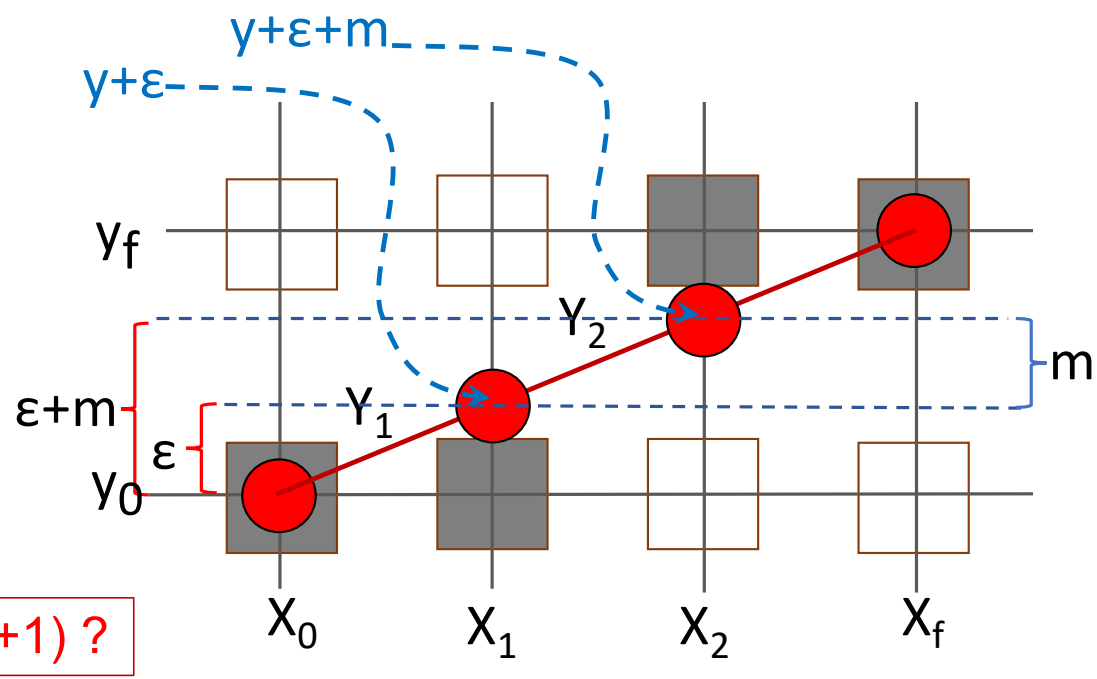
Segmento de Recta

- : Pixel apagado
- : Pixel encendido
- : Punto de la recta (real)
- m : $(y_f - y_0)/(x_f - x_0) = d_y/d_x$
- ε : "error" (distancia) a la ordenada del pixel



Segmento de Recta

- : Pixel apagado
- : Pixel encendido
- : Punto de la recta (real)
- m : $(y_f - y_0)/(x_f - x_0) = d_y/d_x$
- ε : "error" (distancia) a la ordenada del pixel



¿Cuál pixel pintar (x+1, y) o (x+1, y+1) ?

Bresenham

si $y + \varepsilon + m < y + 0.5$

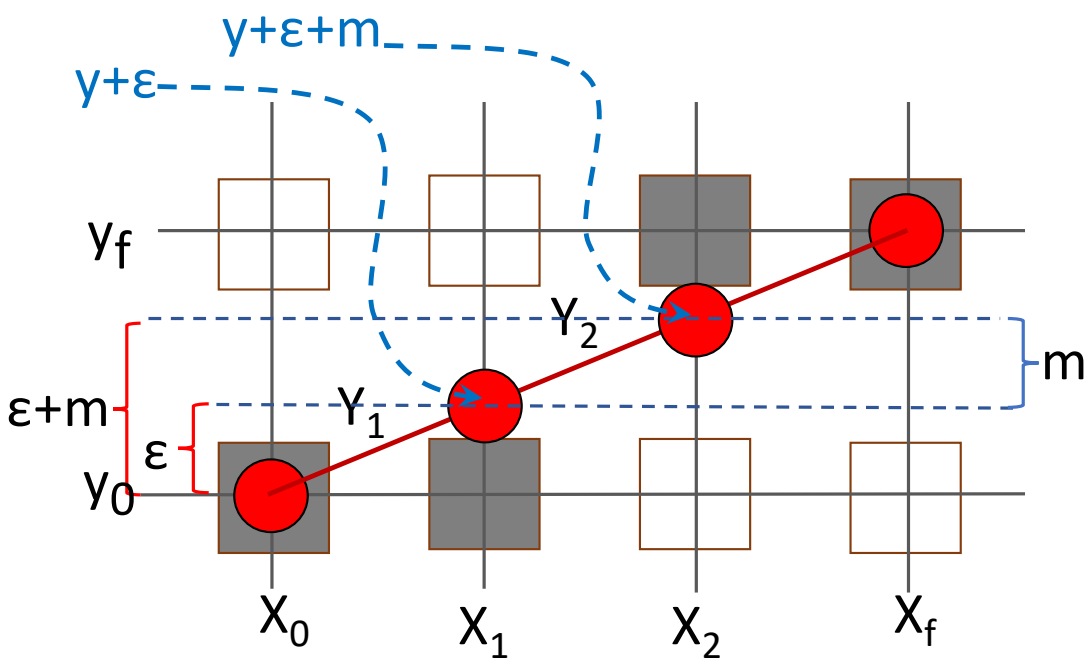
se dibuja $(x+1, y)$

$\varepsilon = (y + \varepsilon + m) - y$ **$= \varepsilon + m$**

de lo contrario

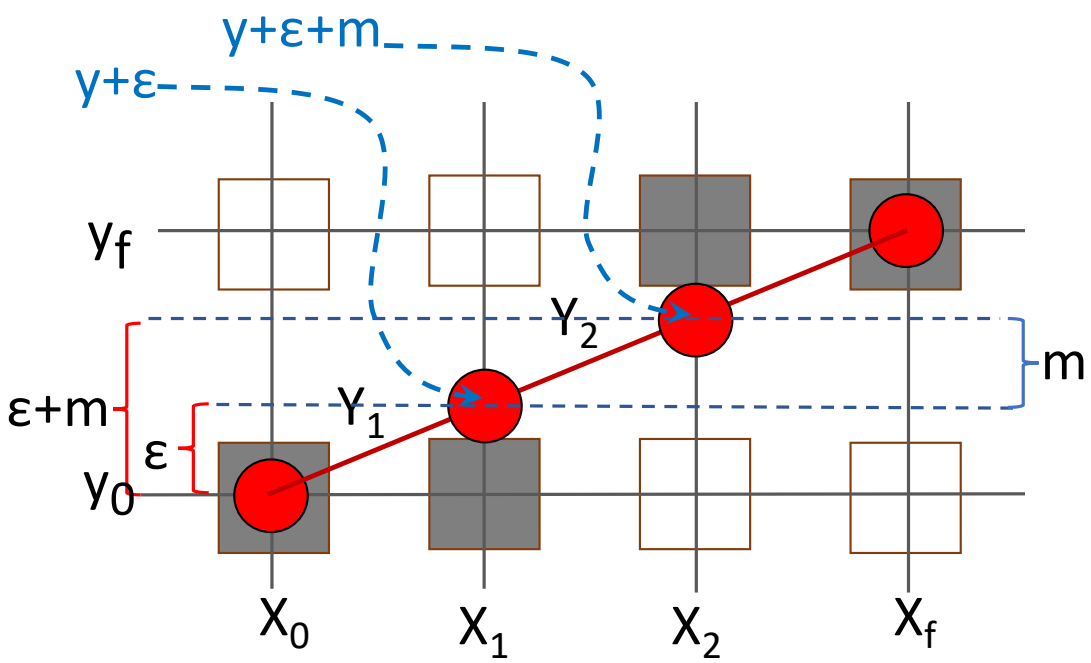
se dibuja $(x+1, y + 1)$

$\varepsilon = (y + \varepsilon + m) - (y + 1) =$ **$\varepsilon + m - 1$**



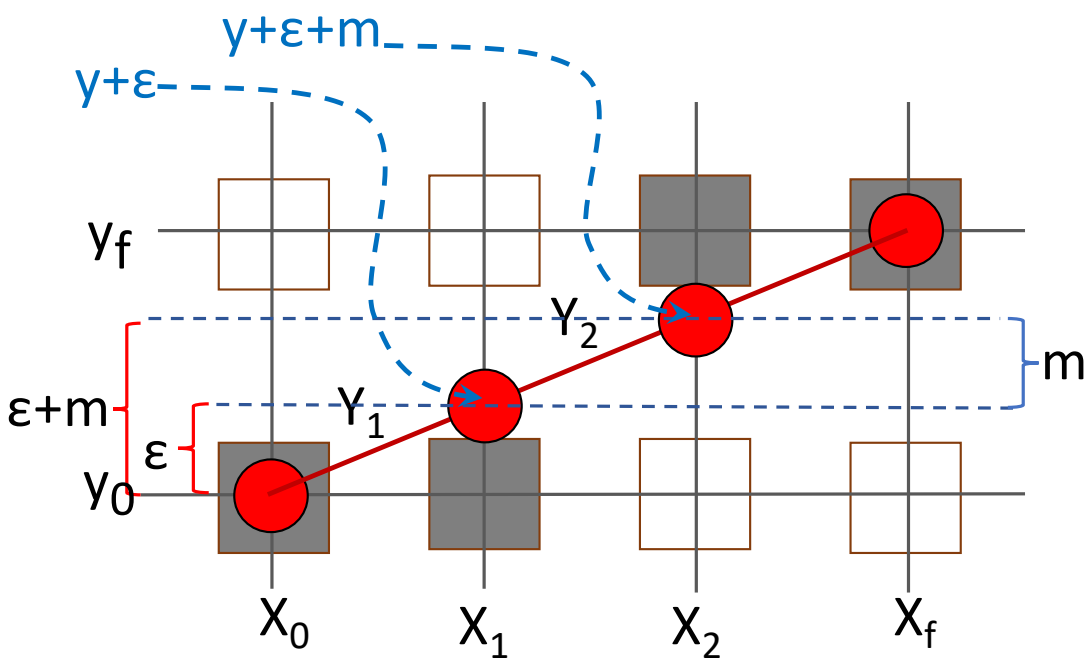
Bresenham Versión 1

```
ε = 0
y = y0
para x = x0 hasta xf
    setPixel(x, y)
    si ε + m < 0.5
        ε  = ε + m
    sino
        y = y + 1
        ε = ε + m - 1
```



Bresenham Versión 1

```
ε = 0
y = y0
para x = x0 hasta xf
    setPixel(x, y)
    si ε + m < 0.5
        ε  = ε + m
    sino
        y = y + 1
        ε = ε + m - 1
```



"m" es un valor no entero 😳

Bresenham … corrección

- Recordar que:

  $m = d_y / d_x$
- entonces:

  $\varepsilon + m < 0.5$

  $\varepsilon + d_y / d_x < 0.5$
- Multiplicando por $2*d_x$

  $(\varepsilon + d_y / d_x) * 2 * dx < 0.5 * 2 * d_x$

  $\varepsilon * 2 * d_x + d_y * 2 < d_x$
- Sustituyendo $\varepsilon * d_x$; reordenando, el test pasa a:

  $2* (\varepsilon' + d_y) < d_x$

Bresenham … corrección

- Hay que actualizar el error
  - $\varepsilon = \varepsilon + m$
  - $\varepsilon = \varepsilon + m - 1$
- Se multiplica por d_x
  - $\varepsilon * d_x = (\varepsilon + m) * d_x \quad = (\varepsilon + (d_y / d_x)) * d_x \quad = \varepsilon * d_x + d_y$
  - $\varepsilon * d_x = (\varepsilon + m - 1) * d_x = (\varepsilon + (d_y / d_x) - 1) * d_x = \varepsilon * d_x + d_y - d_x$
- Sustituyendo $\varepsilon * d_x$ por ε'
  - $\varepsilon' = \varepsilon' + d_y$
  - $\varepsilon' = \varepsilon' + d_y - d_x$

Bresenham V2

$\varepsilon' = 0$

$y = y_0$

para $x = x_0$ hasta x_f

  setPixel(x, y)

  si $2*(\varepsilon' + d_y) < d_x$

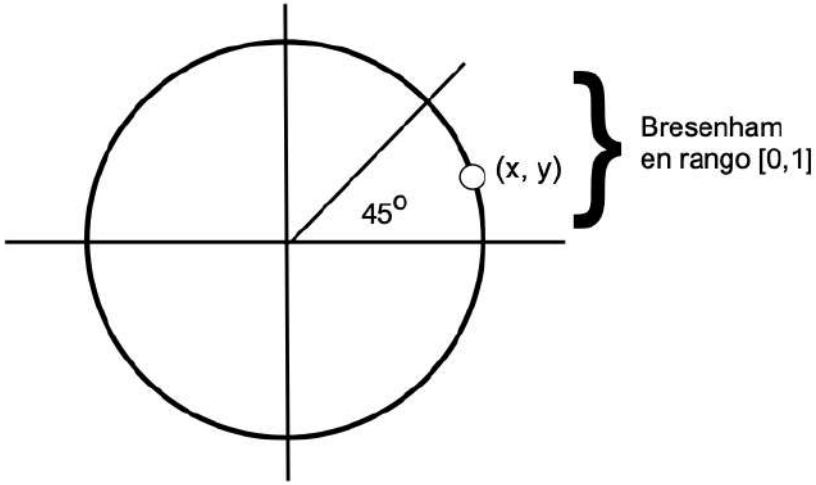
    $\varepsilon' = \varepsilon' + d_y$

  sino

    $y = y + 1$

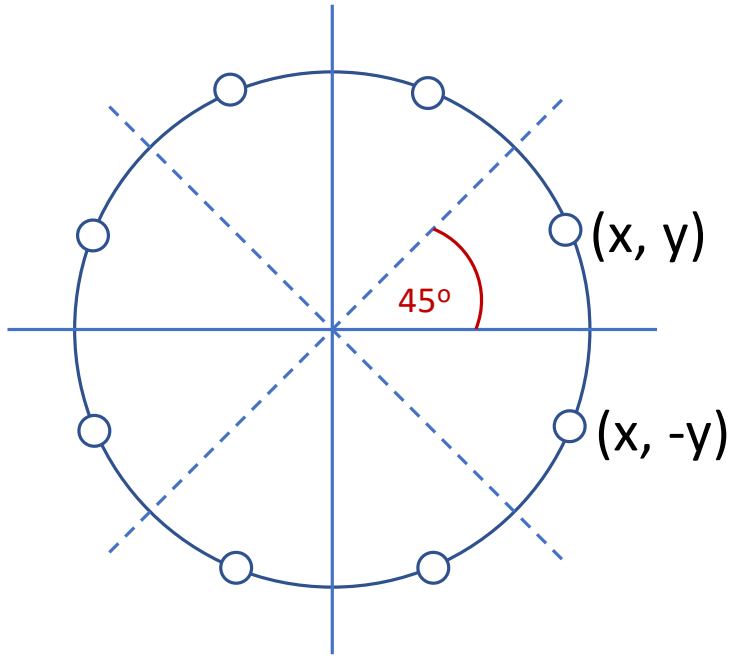
    $\varepsilon' = \varepsilon' + d_y - d_x$

Sólo valores enteros!



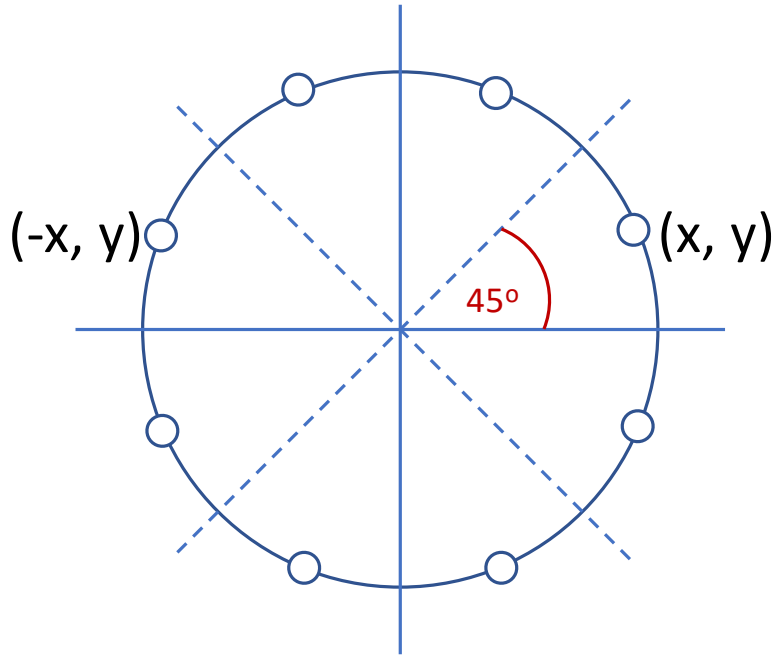
Simetría

- Para Bresenham en pendientes distintas al rango [0, 1] se usa simetría:



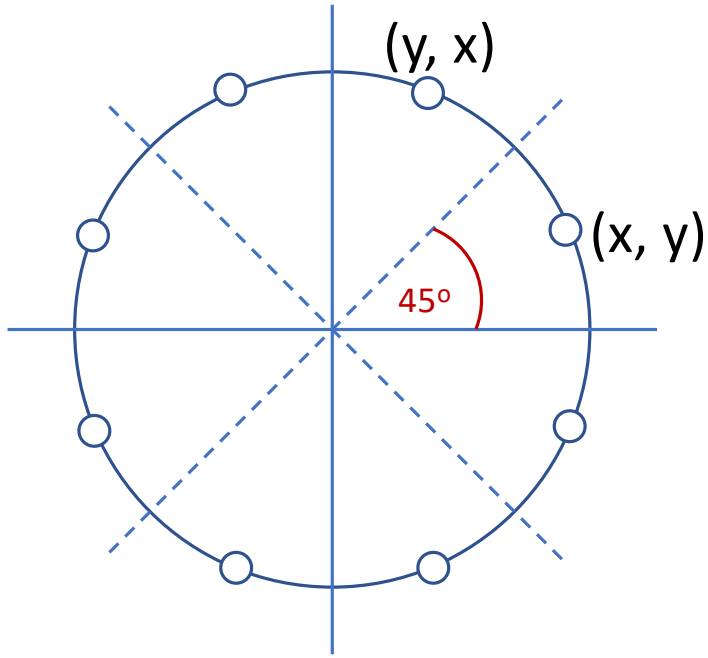
Simetría

- Para Bresenham en pendientes distintas al rango [0, 1] se usa simetría:



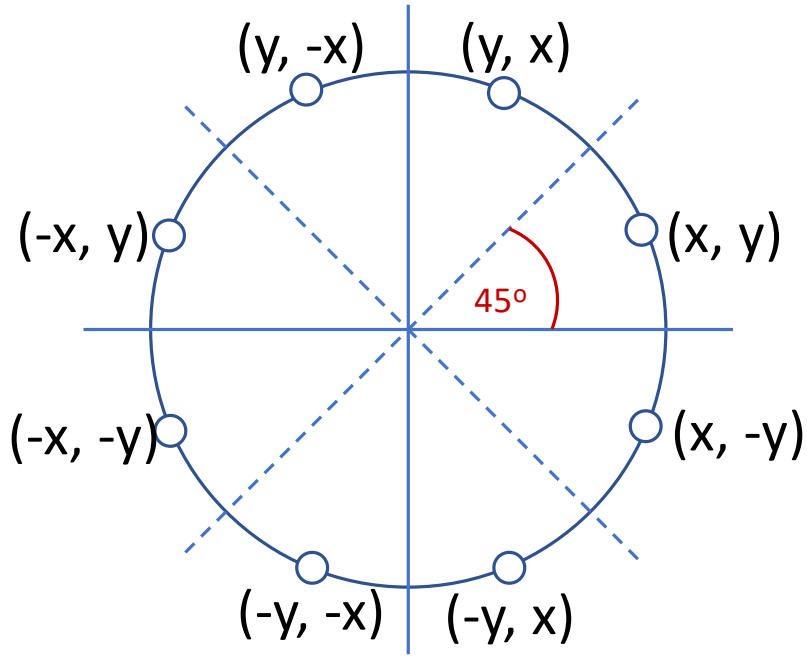
Simetría

- Para Bresenham en pendientes distintas al rango [0, 1] se usa simetría:



Simetría

- Para Bresenham en pendientes distintas al rango [0, 1] se usa simetría:



Bresenham

- Bresenham para cualquier pendiente puede ser implementado usando la implementación de la pendiente en rango [0, 1]
  - Hay que analizar la pendiente de la recta
  - Hay que analizar la posición de (x, y) c/r a (x0, y0)
  - Luego, hay que cambiar los signos de abscisa y ordenada y / o intercambiar abscisa por ordenada

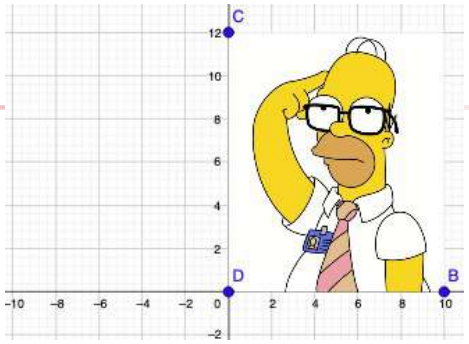
¡¡Propuesto!!

Transformaciones 2D

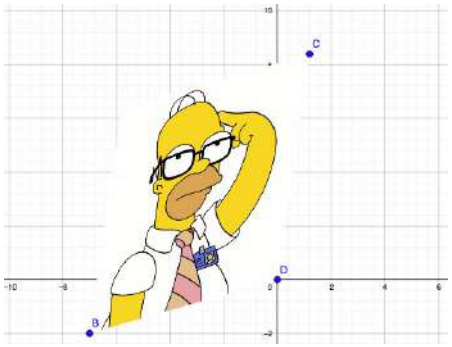
Patricio Inostroza, Dr.

dcc CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

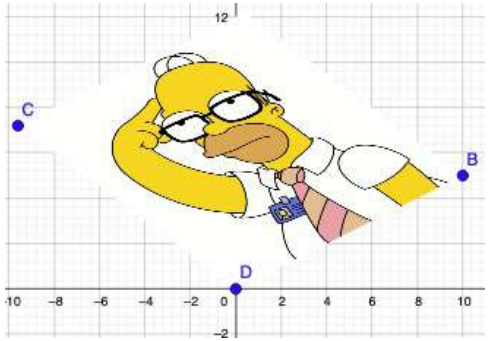
Geometría bidimensional



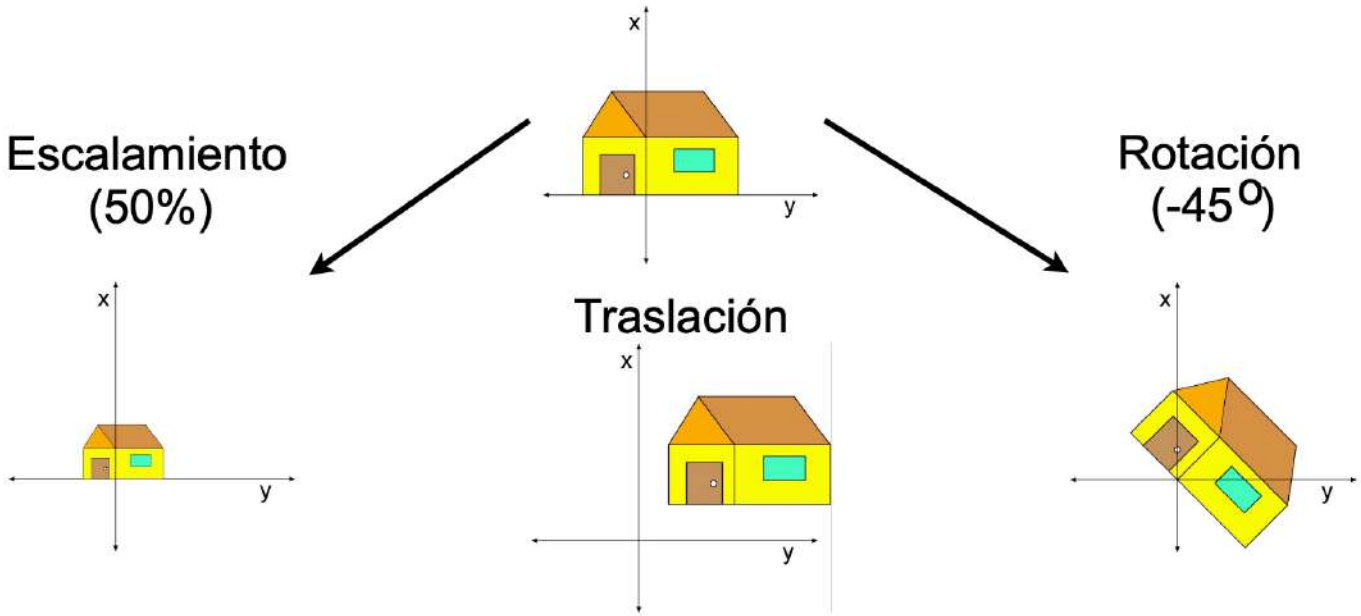
- Problemas de Ingeniería en 2D
  - Uso de modelos bidimensionales



- La visualización requiere de:
  - Un sistema gráfico
  - La creación de entidades geométricas:
    - Líneas, círculos, rectángulos, etc.
  - Transformaciones geométricas:
    - Translaciones, Escalamientos, Rotaciones, ...



Geometría bidimensional



- Las transformaciones geométricas:
  - Permiten la creación y vistas de modelos
  - Son una parte importante de los CAD

Representación

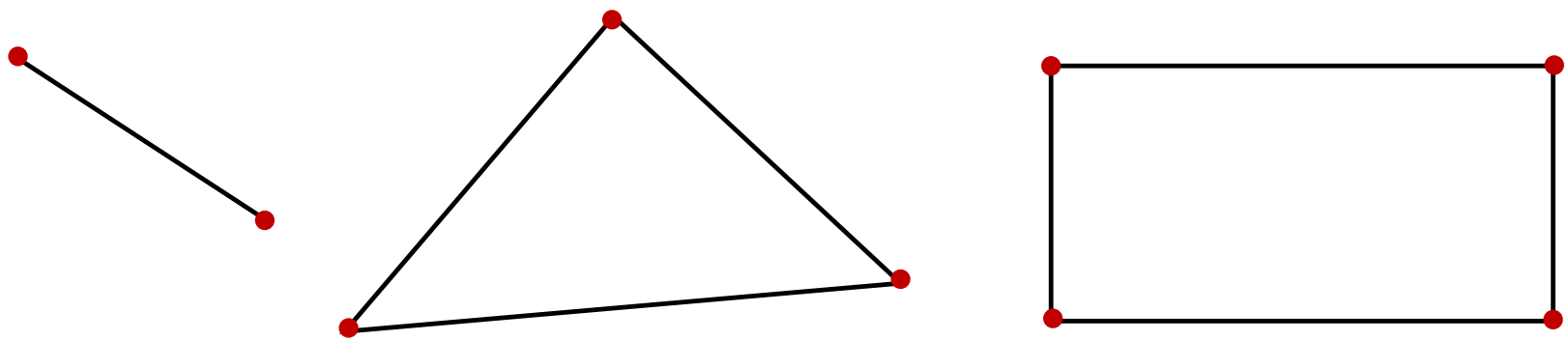
- ¿Cómo representar un objeto 2D ?
  - Mediante una especificación numérica
  - Dentro de un sistema de coordenadas,
    - Ejemplo, cartesiano x, y

- El sistema de coordenadas permite la aplicación de transformaciones capaces de:
  - Trasladar a los objetos
  - Rotar los objetos
  - etc.

Representación

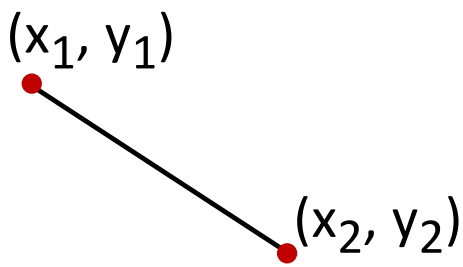
- Punto: elemento básico para la representación de objetos

- Todo objeto puede ser representado a través de un conjunto de puntos



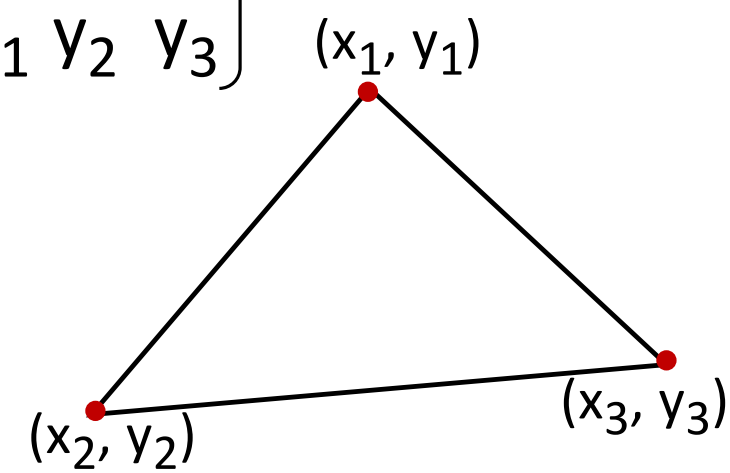
Representación

- Ejemplo:
  - Una línea es representada por sus puntos extremos
  - Un triángulo queda representado por la matriz de 3x2:



$$\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix}$$

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}$$



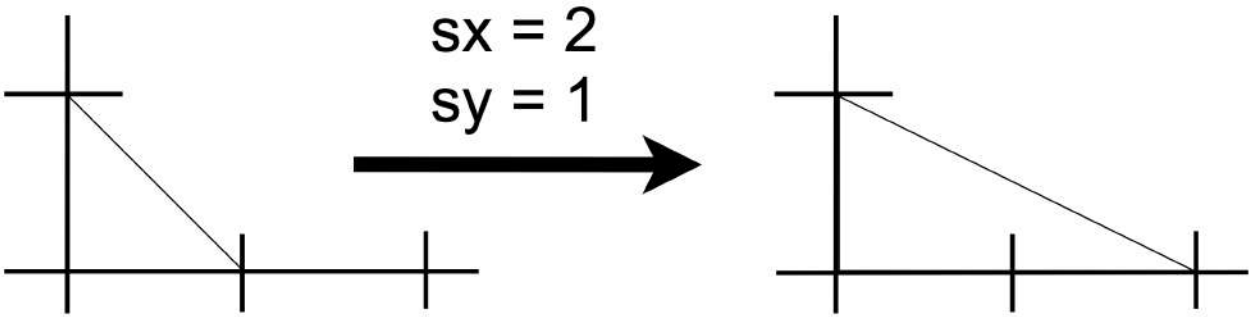
Transformaciones

- En geometría hay transformaciones de interés
  - Escalar
  - Rotar
  - Trasladar
  - Otras

- Representación Matricial
  - Para simplificar la vida…

Escalamiento

- Factores (sx, sy)
- Permiten incrementar / decrementar el valor de las coordenas (x, y) del objeto
  - x’ = sx * x
  - y’ = sy * y
- Ejemplo



Escalamiento

- Escalamiento
  - x' = sx * x
  - y' = sy * y

- Representación matricial

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix} \begin{pmatrix} x1 & x2 & \dots & xn \\ y1 & y2 & \dots & yn \end{pmatrix}$$

- P' = S * P

Escalamiento

- Escalamiento
  - x' = sx * x
  - y' = sy * y

¿Ventajas?

- Representación matricial

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix} \begin{pmatrix} x1 & x2 & \dots & xn \\ y1 & y2 & \dots & yn \end{pmatrix}$$

- P' = S * P

Representación Matricial

- Representación Matricial

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- Simplificación de las operaciones
  - Incrementar la figura en 20% en (x, y) y luego reducir 30% solo x

$$S1 = \begin{pmatrix} 1.2 & 0.0 \\ 0.0 & 1.2 \end{pmatrix}$$

$$S2 = \begin{pmatrix} 0.7 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$

Representación Matricial

- Simplificación de las operaciones
  - Incrementar la figura en 20% en (x, y) y luego reducir 30% solo x

$$S1=\begin{pmatrix}1.2 & 0.0\\ 0.0 & 1.2\end{pmatrix} \qquad \begin{pmatrix}x'\\ y'\end{pmatrix}=\begin{pmatrix}1.2 & 0\\ 0 & 1.2\end{pmatrix}\begin{pmatrix}x1 & x2 & \dots & xn\\ y1 & y2 & \dots & yn\end{pmatrix}$$

$$S2=\begin{pmatrix}0.7 & 0.0\\ 0.0 & 1.0\end{pmatrix} \qquad \begin{pmatrix}x''\\ y''\end{pmatrix}=\begin{pmatrix}0.7 & 0\\ 0 & 1.0\end{pmatrix}\begin{pmatrix}x'1 & x'2 & \dots & x'n\\ y'1 & y'2 & \dots & y'n\end{pmatrix}$$

Representación Matricial

- Simplificación de las operaciones
  - Incrementar la figura en 20% en (x, y) y luego reducir 30% solo x

$$S1=\begin{pmatrix}1.2 & 0.0\\ 0.0 & 1.2\end{pmatrix} \qquad \begin{pmatrix}x'\\ y'\end{pmatrix}=\begin{pmatrix}1.2 & 0\\ 0 & 1.2\end{pmatrix}\begin{pmatrix}x1 & x2 & \dots & xn\\ y1 & y2 & \dots & yn\end{pmatrix}$$

$$S2=\begin{pmatrix}0.7 & 0.0\\ 0.0 & 1.0\end{pmatrix} \qquad \begin{pmatrix}x''\\ y''\end{pmatrix}=\begin{pmatrix}0.7 & 0\\ 0 & 1.0\end{pmatrix}\begin{pmatrix}x'1 & x'2 & \dots & x'n\\ y'1 & y'2 & \dots & y'n\end{pmatrix}$$

- Asociación de las operaciones

$$\begin{pmatrix}x'\\ y'\end{pmatrix}=\begin{pmatrix}0.7 & 0\\ 0 & 1.0\end{pmatrix}\begin{pmatrix}1.2 & 0\\ 0 & 1.2\end{pmatrix}\begin{pmatrix}x1 & x2 & \dots & xn\\ y1 & y2 & \dots & yn\end{pmatrix}$$

Representación Matricial

- Simplificación de las operaciones
  - Incrementar la figura en 20% en (x, y) y luego reducir 30% solo x

$$P=\begin{pmatrix} x1 & x2 & \dots & xn \\ y1 & y2 & \dots & yn \end{pmatrix} \qquad S1=\begin{pmatrix} 1.2 & 0.0 \\ 0.0 & 1.2 \end{pmatrix} \qquad S2=\begin{pmatrix} 0.7 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$

- Asociación de las operaciones

  $P' = S2 * S1 * P$

- Definiendo

  $S = S2 * S1$

- Entonces

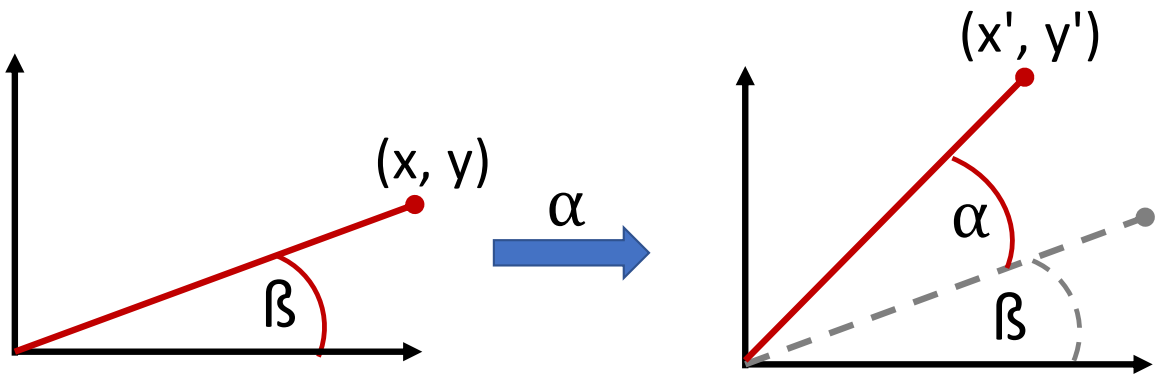
  $P' = S * P$

Rotación

- Rotación

  $x' = x * \cos(\alpha) - y * \mathrm{sen}(\alpha)$

  $y' = x * \mathrm{sen}(\alpha) + y * \cos(\alpha)$

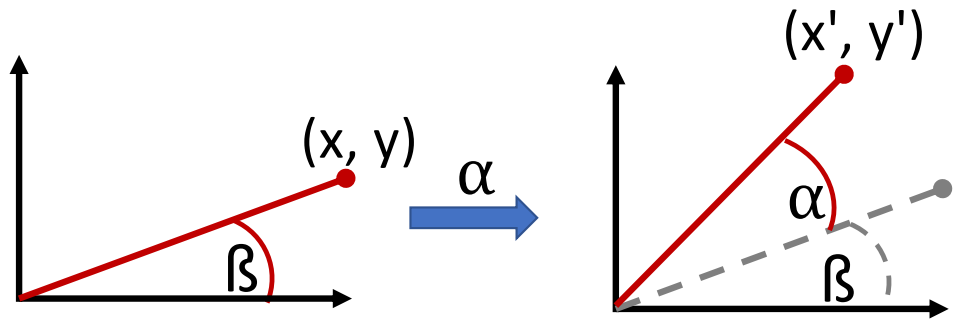


Rotación

- Rotación

  $x' = x * \cos(\alpha) - y * \text{sen}(\alpha)$

  $y' = x * \text{sen}(\alpha) + y * \cos(\alpha)$



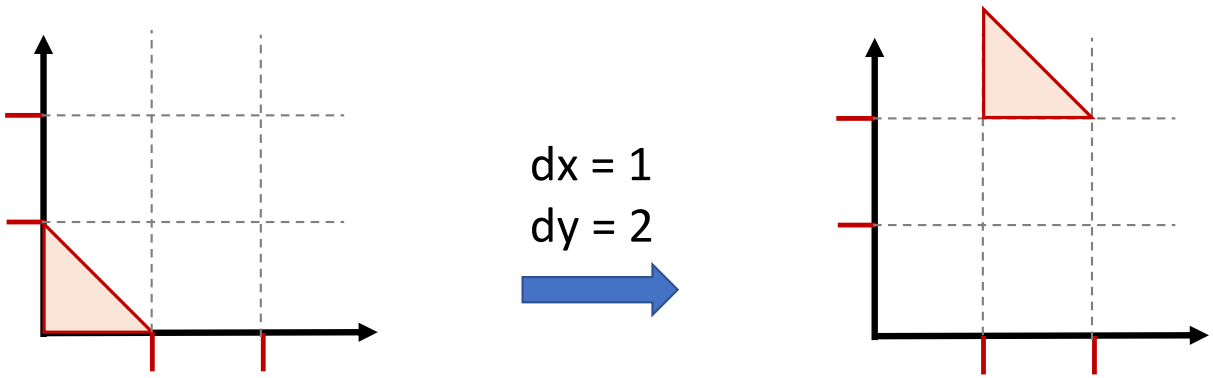
- Representación Matricial

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\text{sen}(\alpha) \\ \text{sen}(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- $P' = R * P$

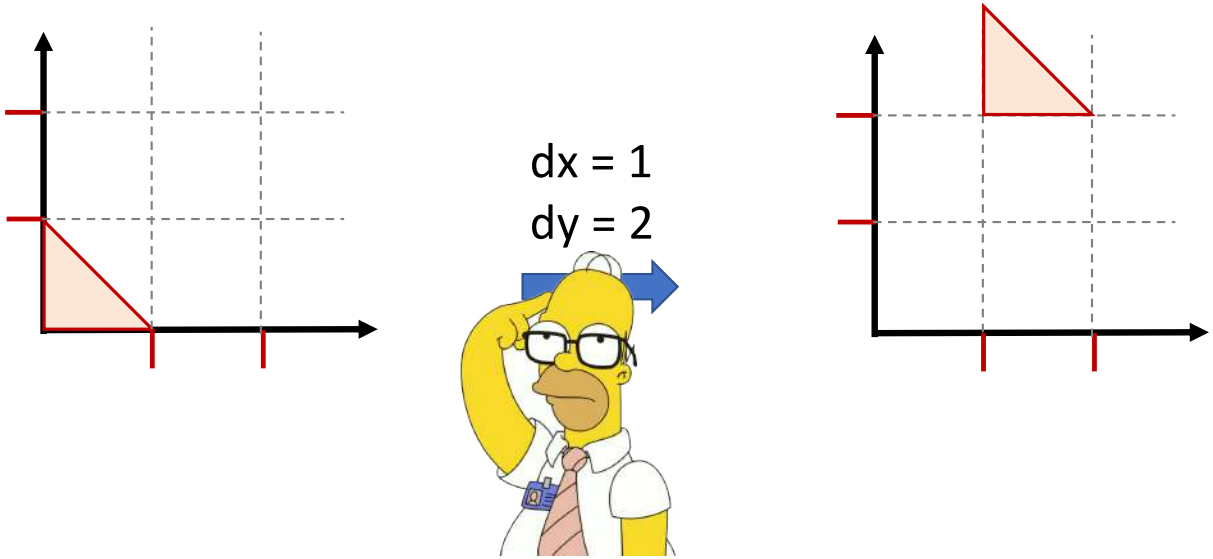
Traslación

- Traslación
  - x' = dx + x
  - y' = dy + y



Traslación

- Traslación
  - x' = dx + x
  - y' = dy + y

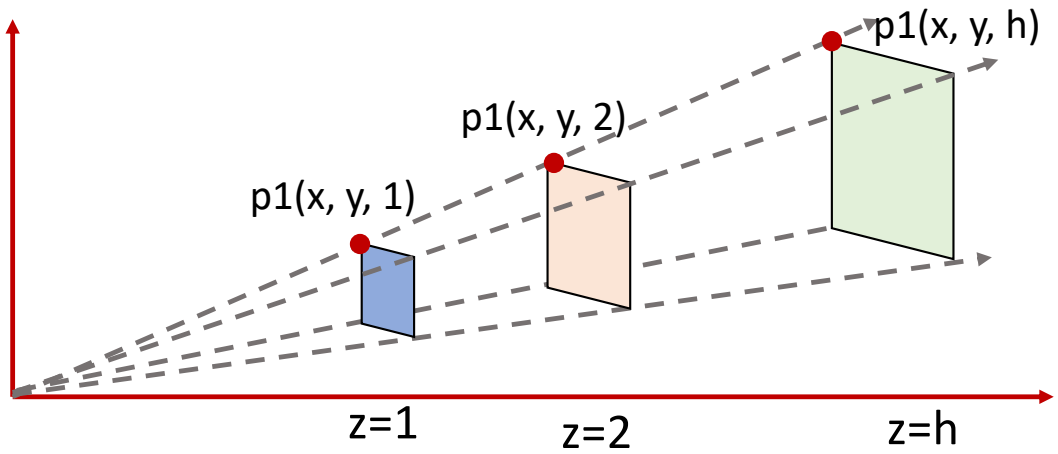


¿Representación Matricial?

Coordenadas Homogéneas

- Coordenadas homogéneas
  - Punto 2D en el espacio tridimensional

- Todo punto 2D (p) puede ser representado por cualquier punto a lo largo de rayo 3D (llamado espacio homogéneo)



- Coordenadas homogéneas → z = 1

Coordenadas Homogéneas

- Usando Coordenadas homogéneas
- Cambia la representación de cada punto

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} xh \\ yh \\ h \end{pmatrix}$$

- Donde:

  xh = x / h

  yh = y / h

- No está permitido h sea cero
- Para el curso: h = 1 😋

Coordenadas Homogéneas

- Usando Coordenadas homogéneas
- Cambia la representación de cada punto

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} xh \\ yh \\ h \end{pmatrix}$$

- Donde:

  xh = x / h

  yh = y / h

¿y que pasa con las transformaciones anteriores?

- No está permitido h sea cero
- Para el curso: h = 1 😋

Coordenadas Homogéneas

- Coordenadas homogéneas
  - Escalamiento

$$\begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

  - Rotación

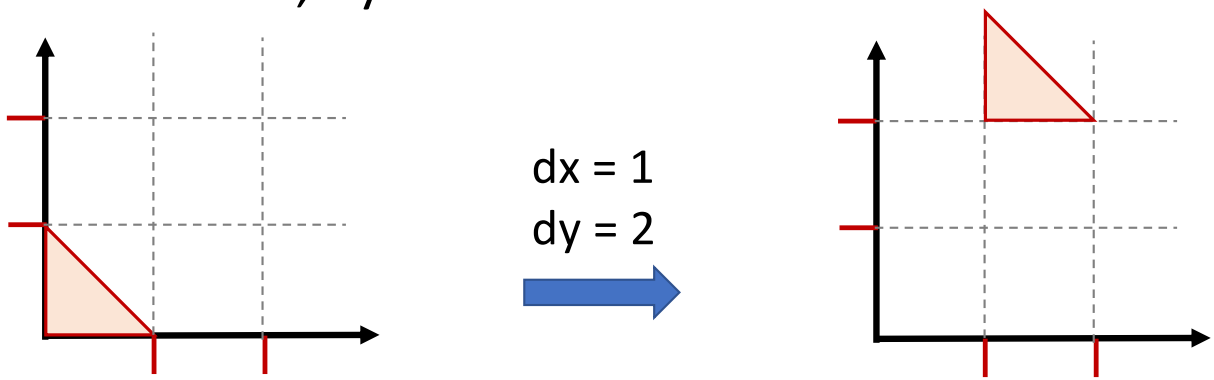
$$\begin{pmatrix} \cos(\alpha) & -\text{sen}(\alpha) & 0 \\ \text{sen}(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Coordenadas Homogéneas

- Coordenadas Homogéneas
  - Traslación

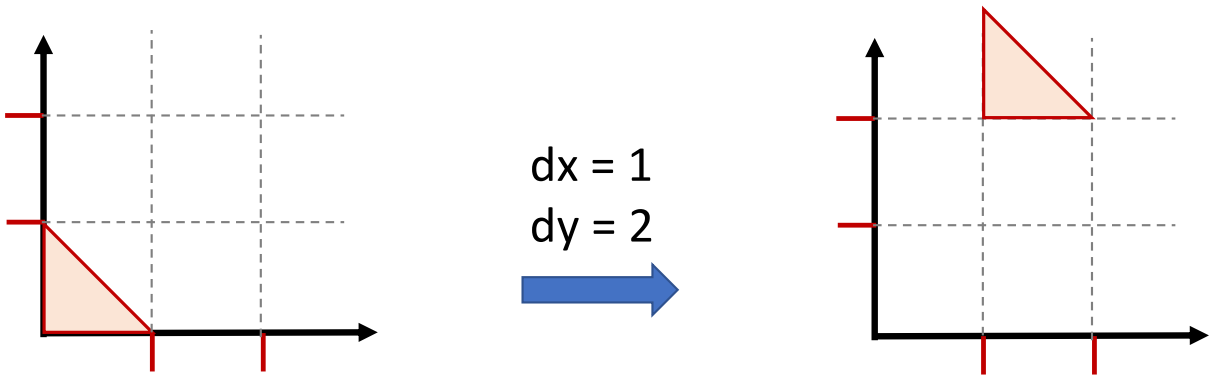
$$\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix}$$

- Trasladar en dx = 2 , dy = 1



Coordenadas Homogéneas

- Trasladar en dx = 2 , dy = 1



$$T' = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x0 & x1 & x2 \\ y0 & y1 & y2 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} x0+2 & x1+2 & x2+2 \\ y0+1 & y1+1 & y2+1 \\ 1 & 1 & 1 \end{pmatrix}$$

Composición de Transformaciones

- Cuando una transformación no es suficiente….

- Primera transformación: M1

  P1 = M1 * P

- Segunda transformación: M2

  P2 = M2 * P1

Composición de Transformaciones

- Primera transformación: M1

  P1 = M1 * P

- Segunda transformación: M2

  P2 = M2 * P1

- : :

- n-ésima transformación: Mn

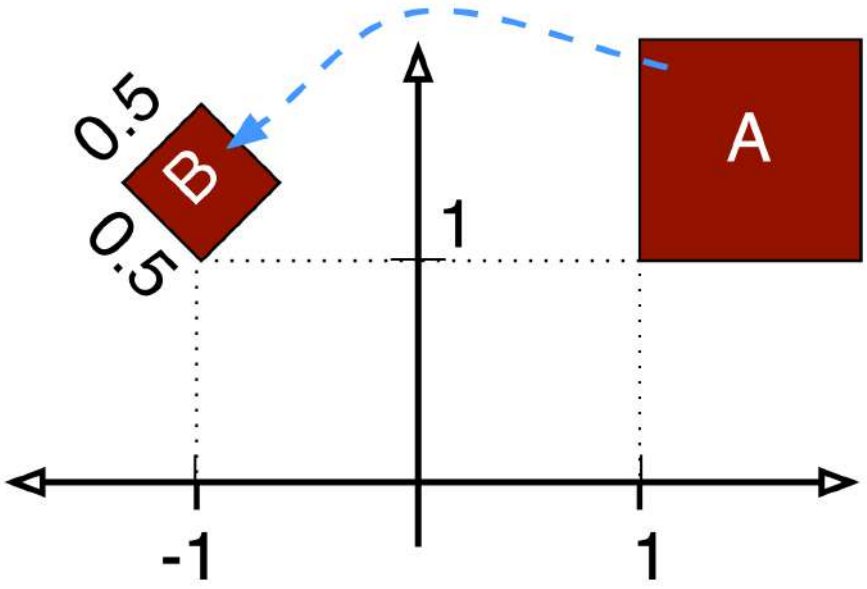
  Pn = Mn * Pn

- Reordenando:

  Pn = (Mn * … * M2 * M1) * P = Mn1 * P

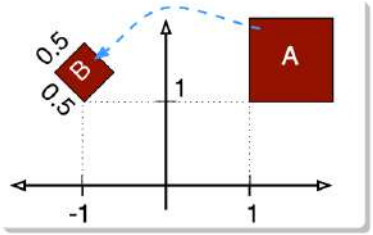
Ejemplo

- Transformar el objeto A en B



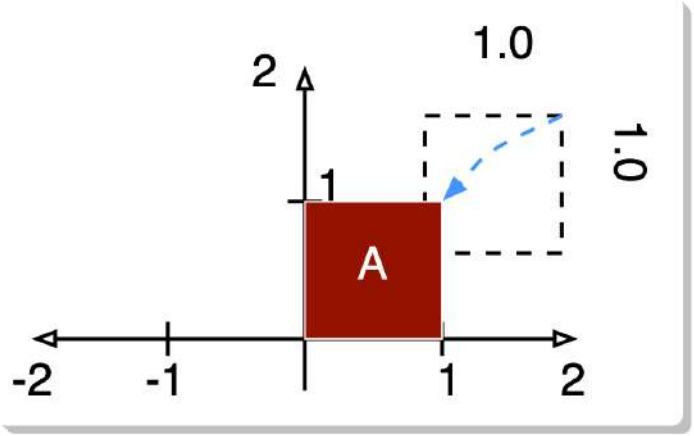
Ejemplo

- Transformar el objeto A en B



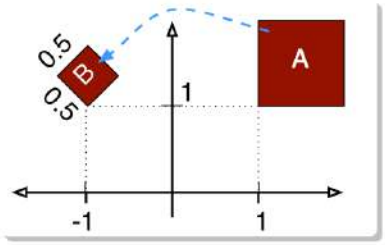
- Paso 1: Trasladar al origen

$$T1 = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$



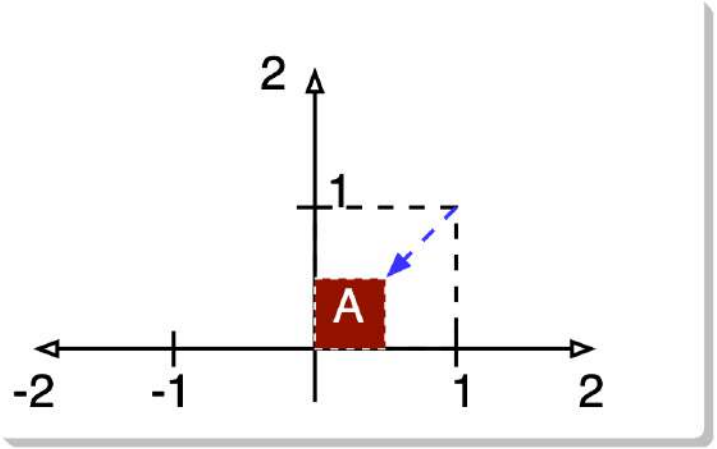
Ejemplo

- Transformar el objeto A en B



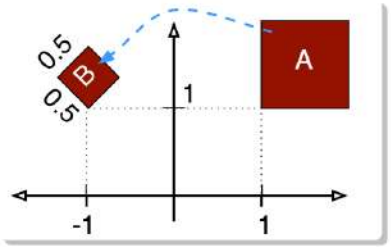
- **Paso 2: Escalar el objeto**

$$E = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



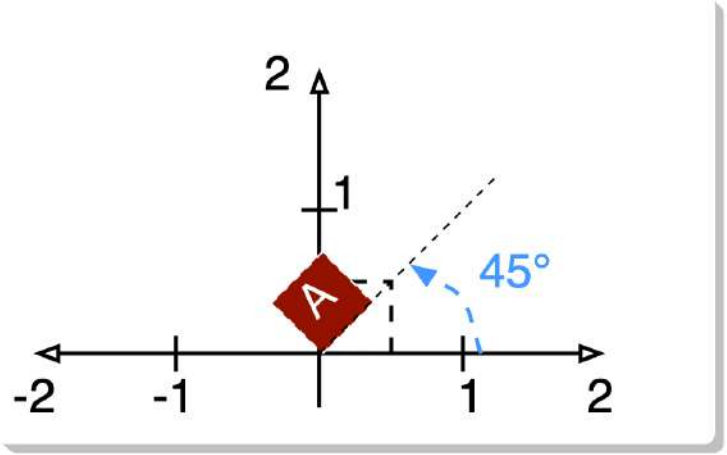
Ejemplo

- Transformar el objeto A en B



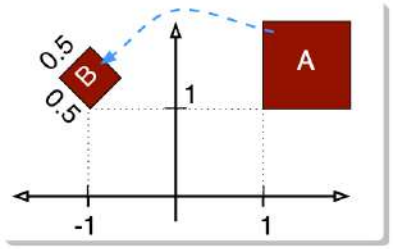
- **Paso 3: Rotar el objeto**

$$R = \begin{pmatrix} \cos(\alpha) & -\text{sen}(\alpha) & 0 \\ \text{sen}(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



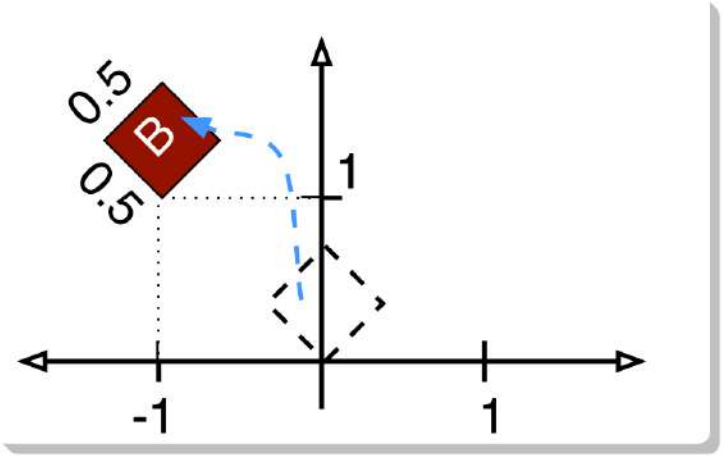
Ejemplo

- Transformar el objeto A en B



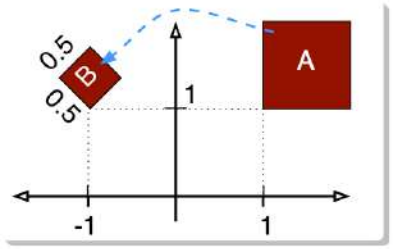
- Paso 4: Trasladar el objeto

$$T2 = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$



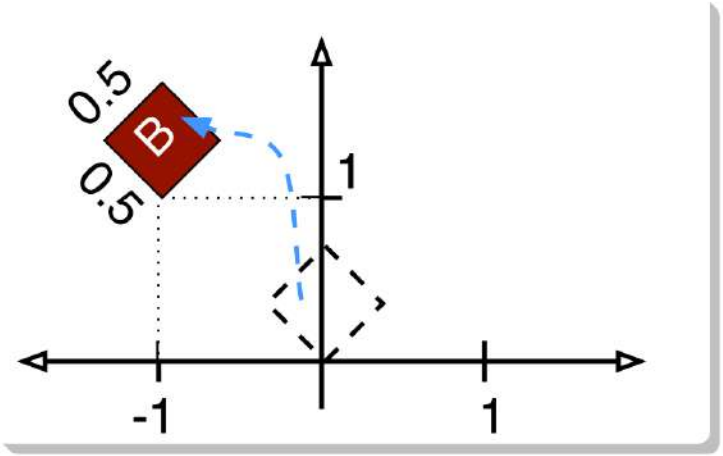
Ejemplo

- Transformar el objeto A en B



- Paso 4: Trasladar el objeto

$$T2 = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$



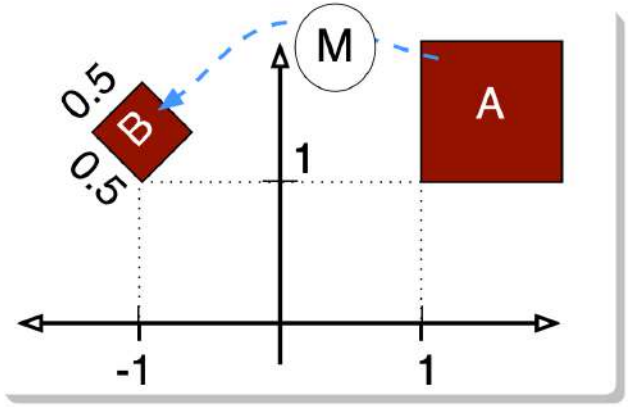
Ejemplo

- Transformar el objeto A en B
  - Matriz de transformación final

    M = T2 * R * E * T1

  - Objeto transformado

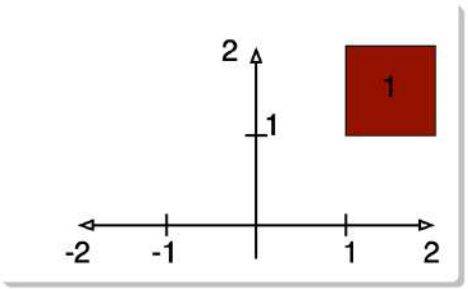
    P' = M * P

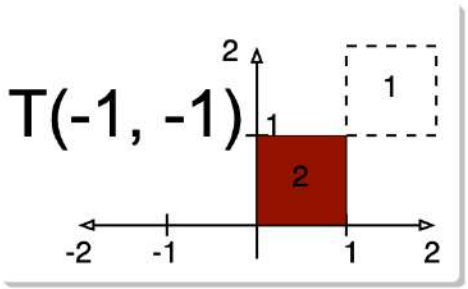


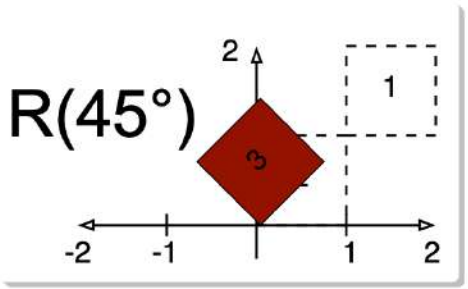
$$M = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\alpha) & -\text{sen}(\alpha) & 0 \\ \text{sen}(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

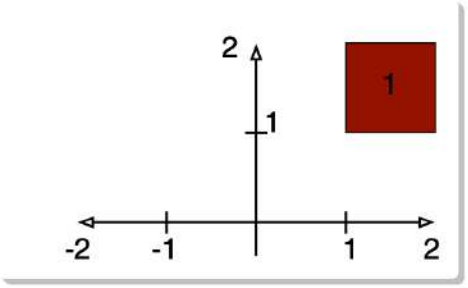
Composición de Transformaciones

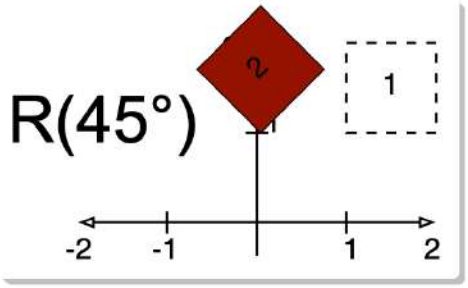
- El orden SI importa

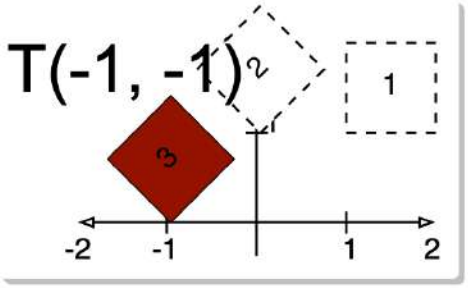






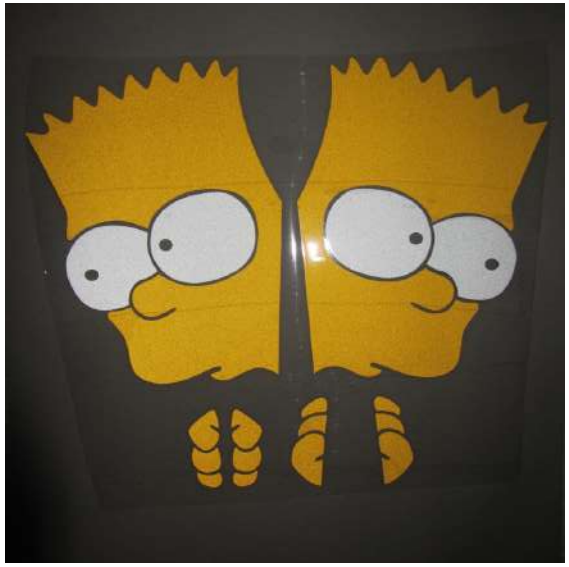






Reflexión

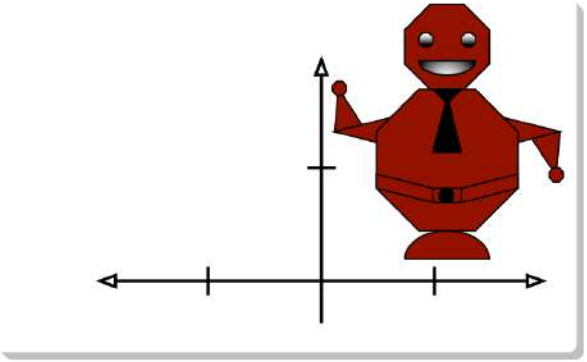
- Reflexión respecto a un eje

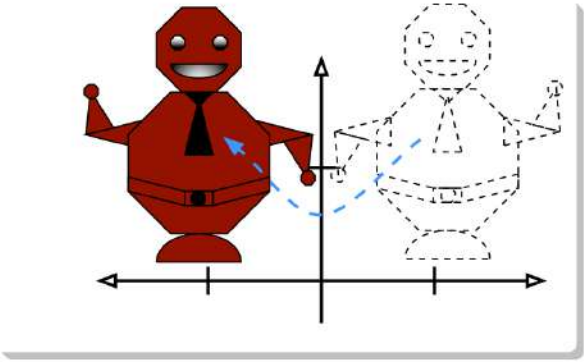


Reflexión

- Reflexión c/r al eje Y

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

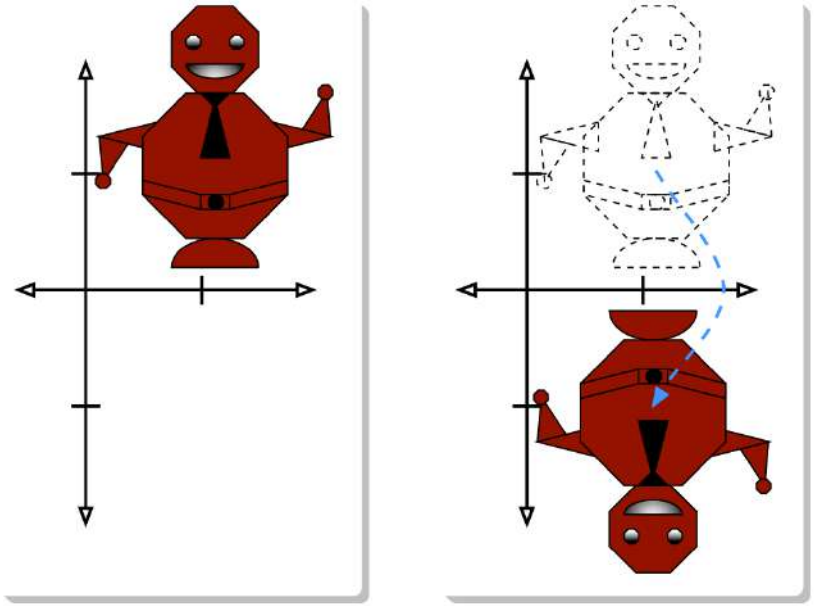




Reflexión

- Reflexión c/r al eje X

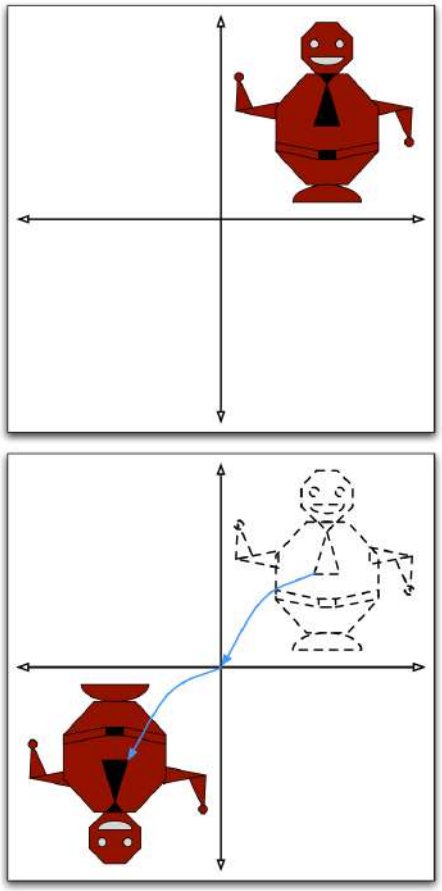
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflexión

- Reflexión c/r al origen

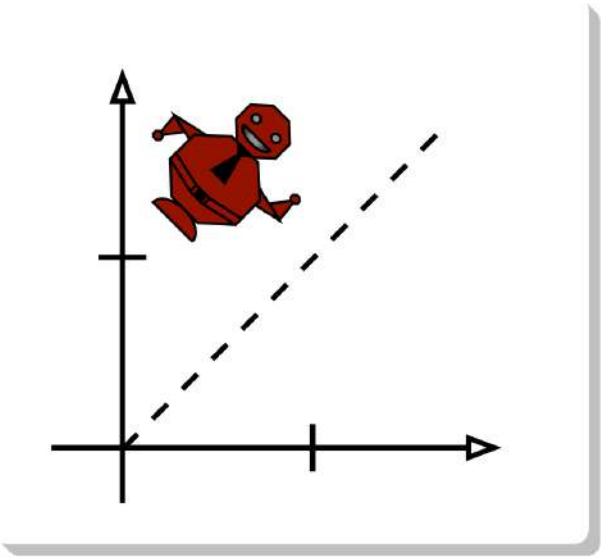
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

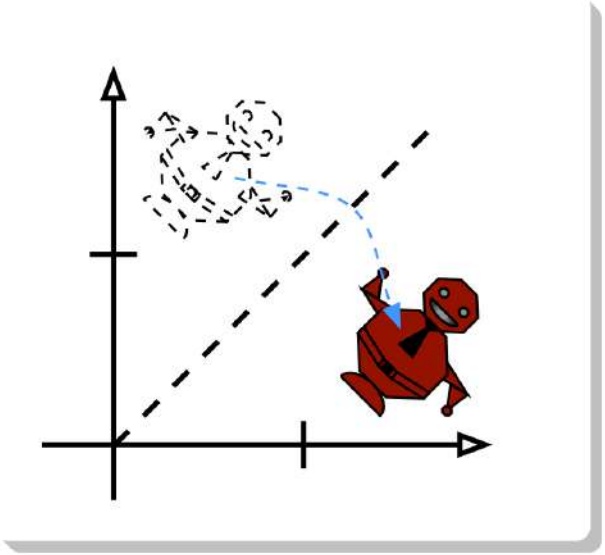


pinostro@dcc.uchile.cl

Reflexión

- Reflexión c/r a un eje arbitrario





Propuesto!!

Shearing

- Transformación que adiciona a una coordenada un valor lineal de la otra coordenada

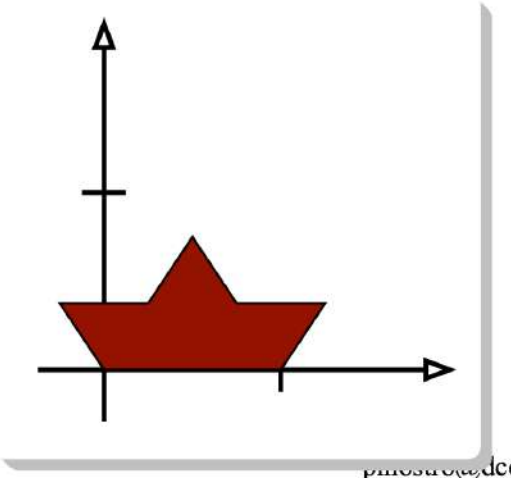
- Por ejemplo:
  - $x' = x + 5*y$
  - $y' = y$

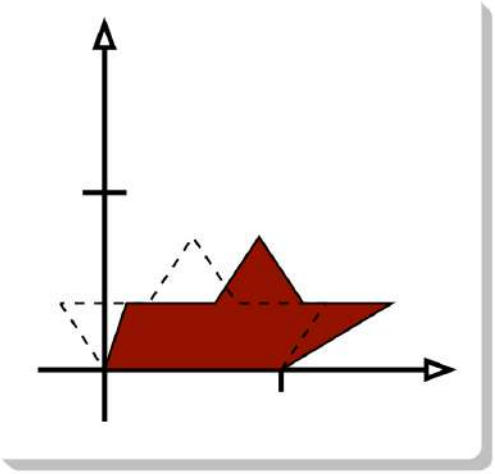
Valor lineal de y

Shearing

- Shearing en X

$$\begin{bmatrix} 1 & Shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

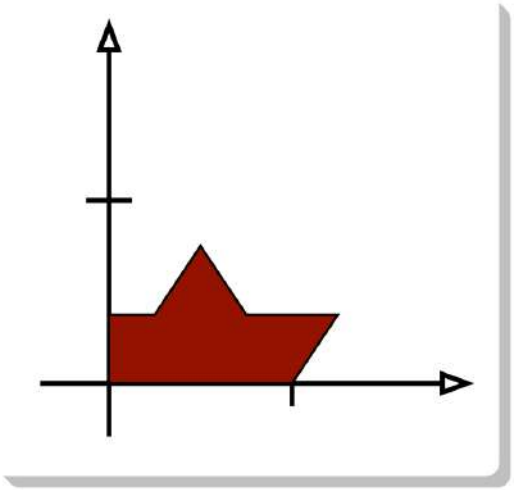


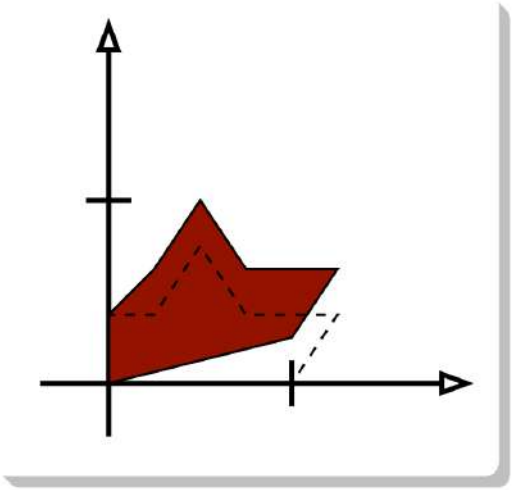


Shearing

- Shearing en Y

$$\begin{bmatrix} 1 & 0 & 0 \\ Shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

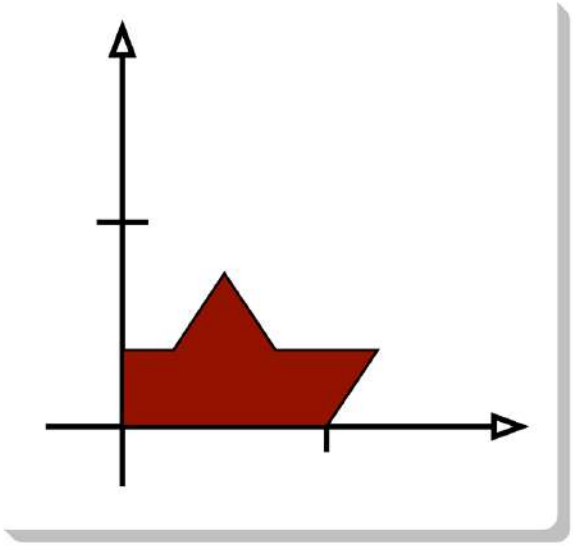


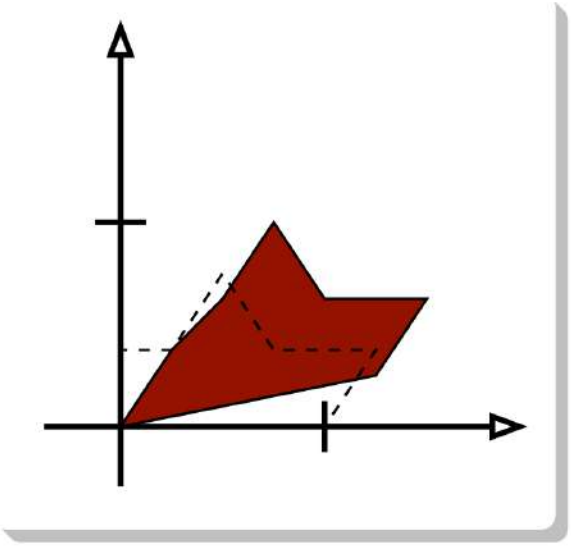


Shearing

- Shearing en X e Y

$$\begin{bmatrix} 1 & Shx & 0 \\ Shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



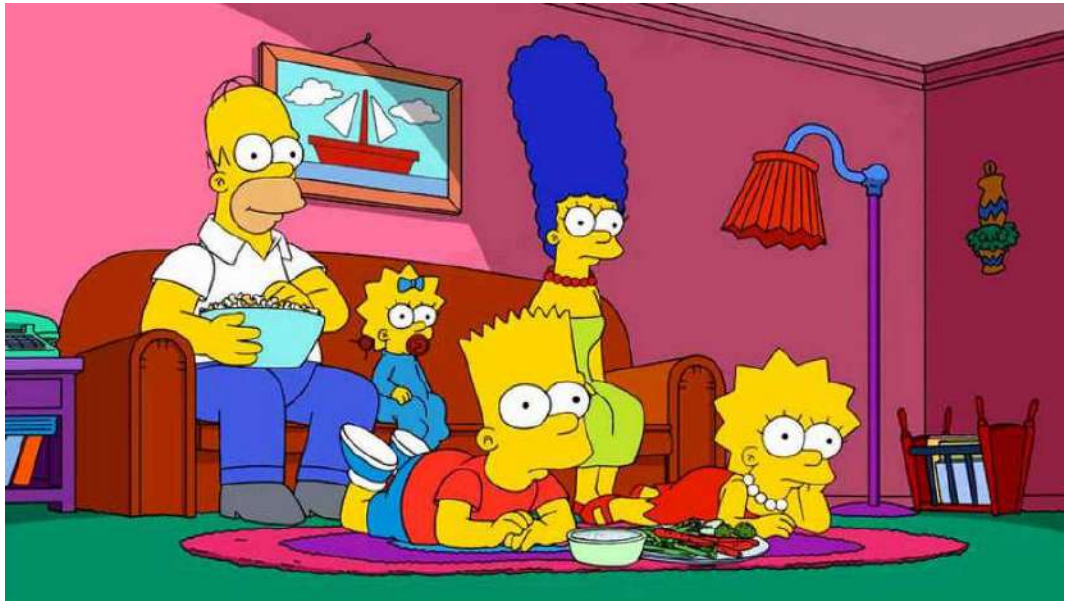


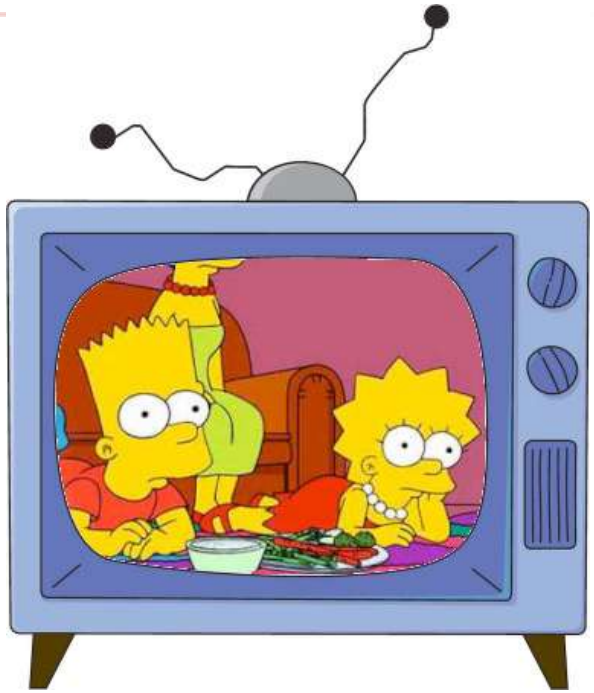
Viewing 2D

Patricio Inostroza, Dr.

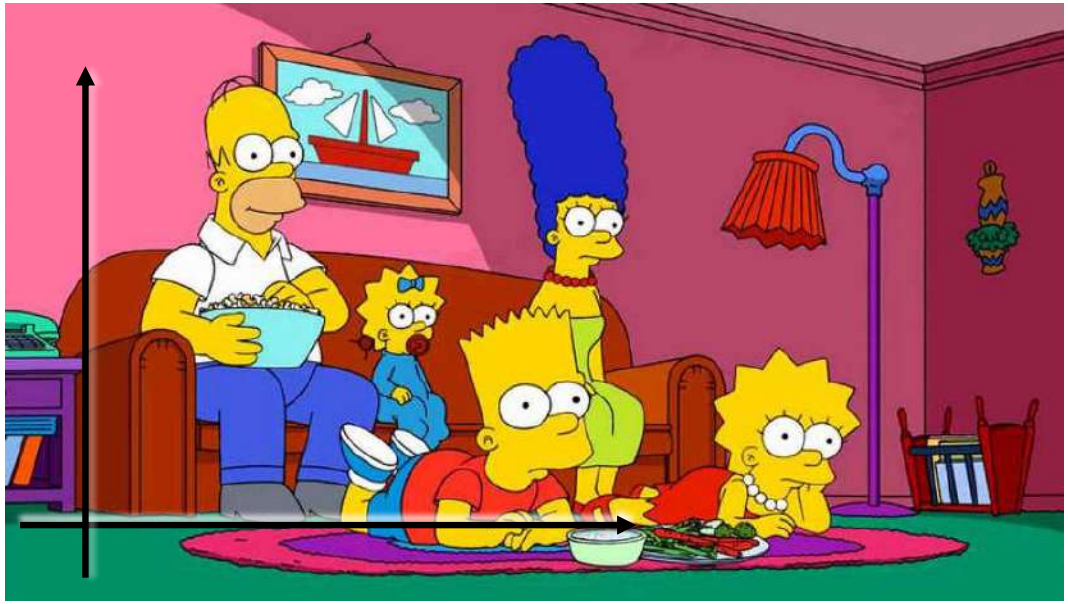
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Del mundo al dispositivo

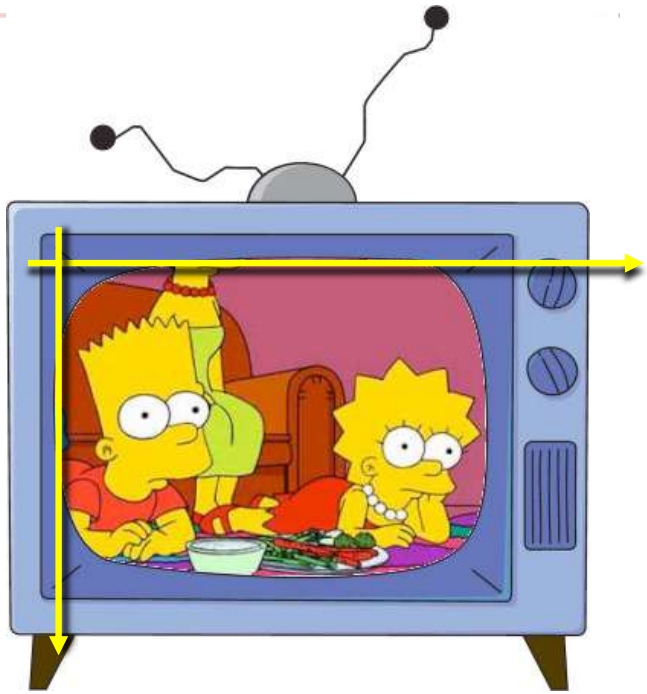




Del mundo al dispositivo

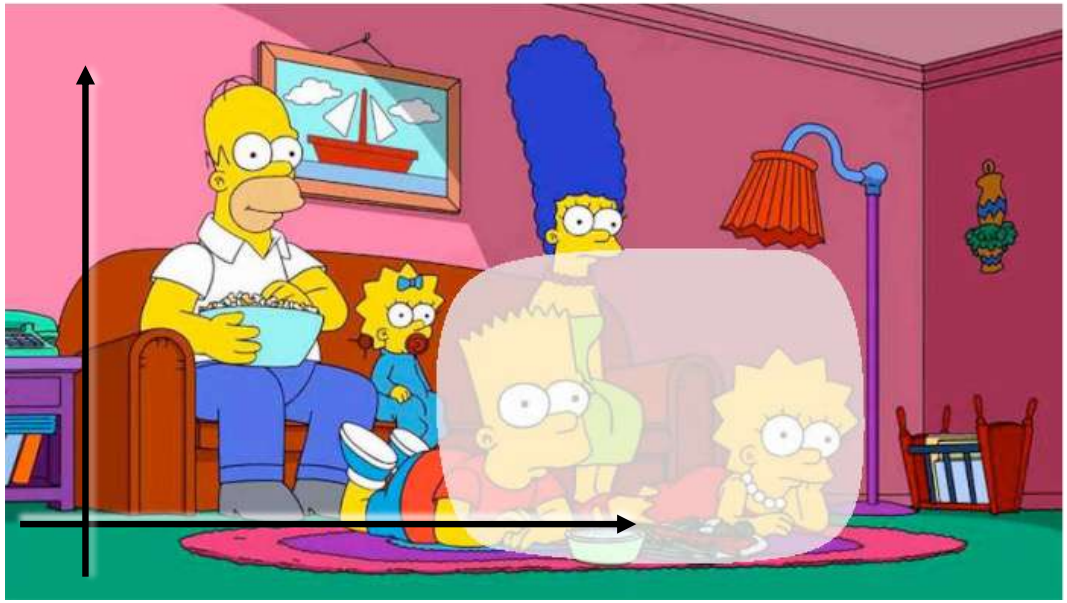


- Sistema coordenadas del mundo

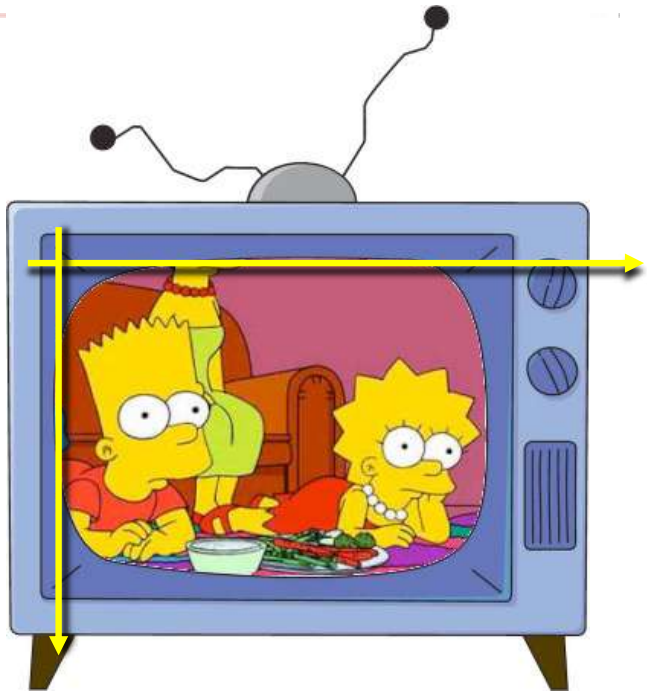


- Sistema coordenadas del dispositivo

Del mundo al dispositivo



- Sistema coordenadas del mundo
- Establecer la zona de interés



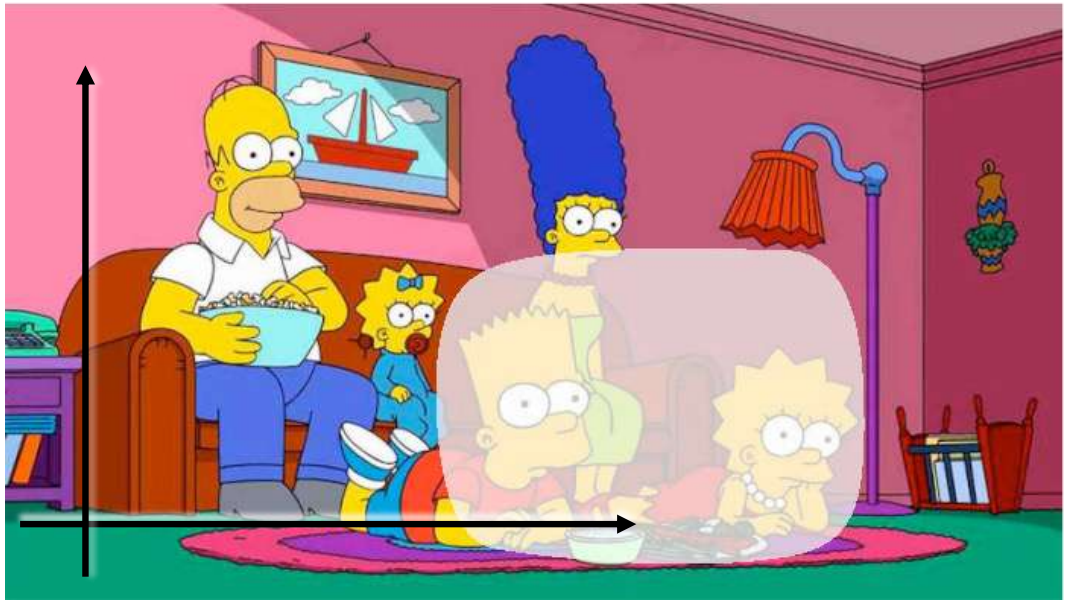
- Sistema coordenadas del dispositivo

Viewing 2D

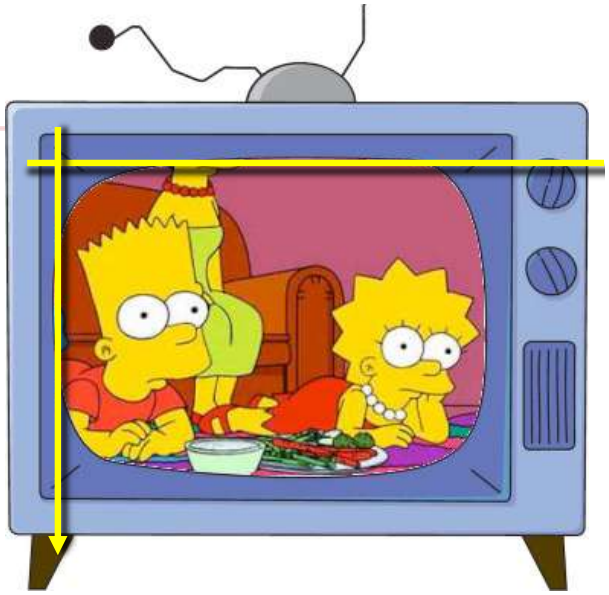
- Todo dispositivo de salida (pantalla) tienen su propio sistema de coordenadas
- Cada objeto tienen un sistemas de coordenadas
  - generalmente corresponde a las coordenadas del mundo
- Por lo general, sólo una parte del objeto tiene interés en ser proyectada (window)

- ¿ Cómo realizar la proyección ?

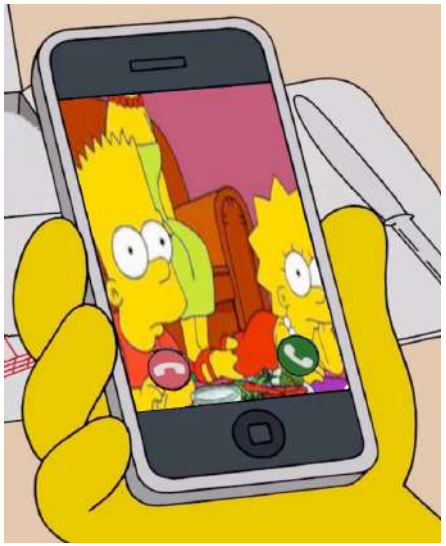
Del mundo al dispositivo



f1(x, y)

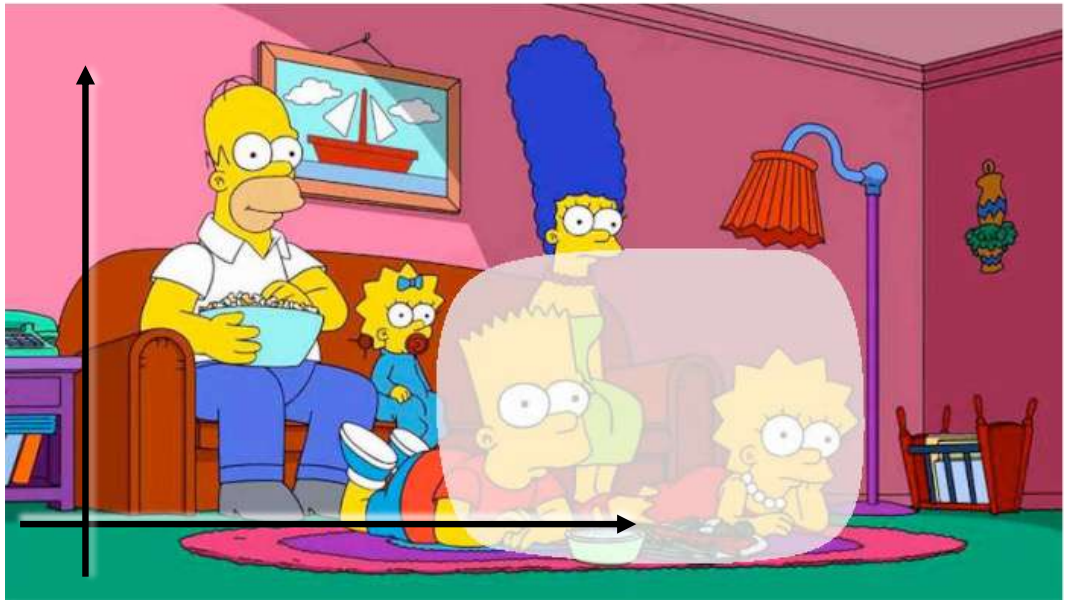


f2(x, y)

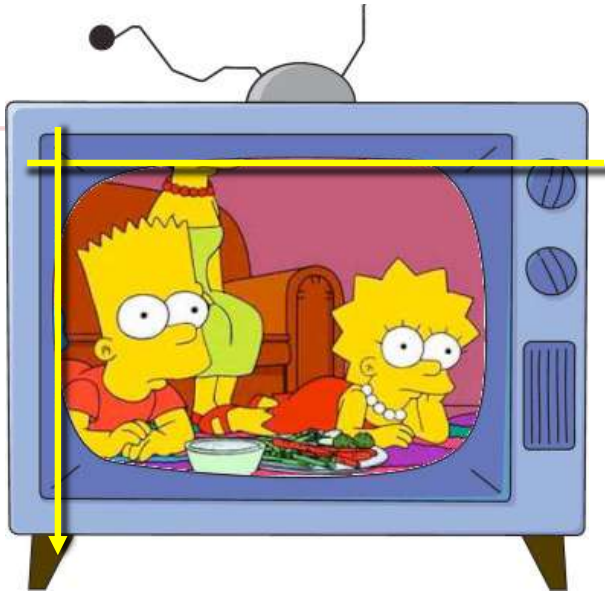


- Convertir la zona de interés a las coordenadas del dispositivo

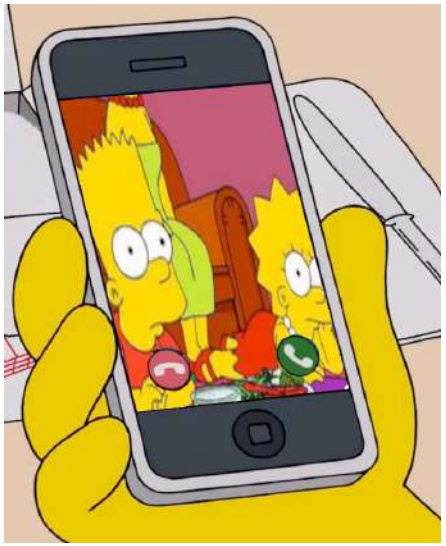
Del mundo al dispositivo



f1(x, y)



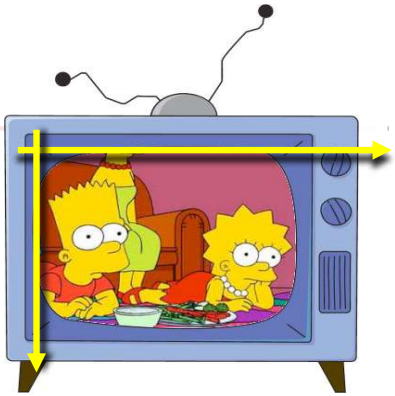
f2(x, y)



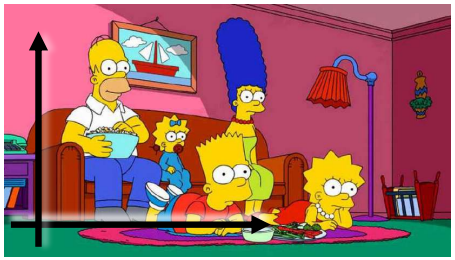
- Convertir la zona de interés a las coordenadas del dispositivo

Problema:
Depende del dispositivo!

Definiciones



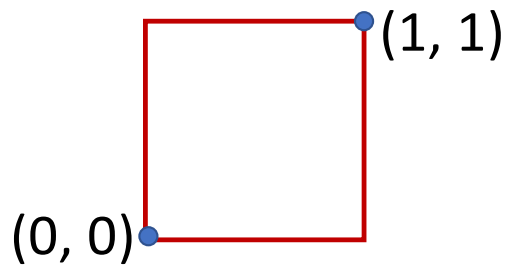
- Coordenadas del dispositivo (DC):
  - Corresponden a las coordenadas del actual dispositivo
  - Especifican en que parte de la pantalla aparecerá el objeto



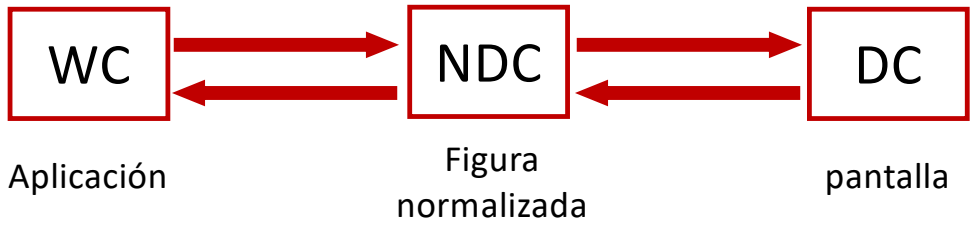
- Sistema de coordenadas universal (WC):
  - Sistema cartesiano utilizado por las coordenadas del objeto
  - Las transformaciones antes vistas se basan en este sistema

Coordenadas normalizadas

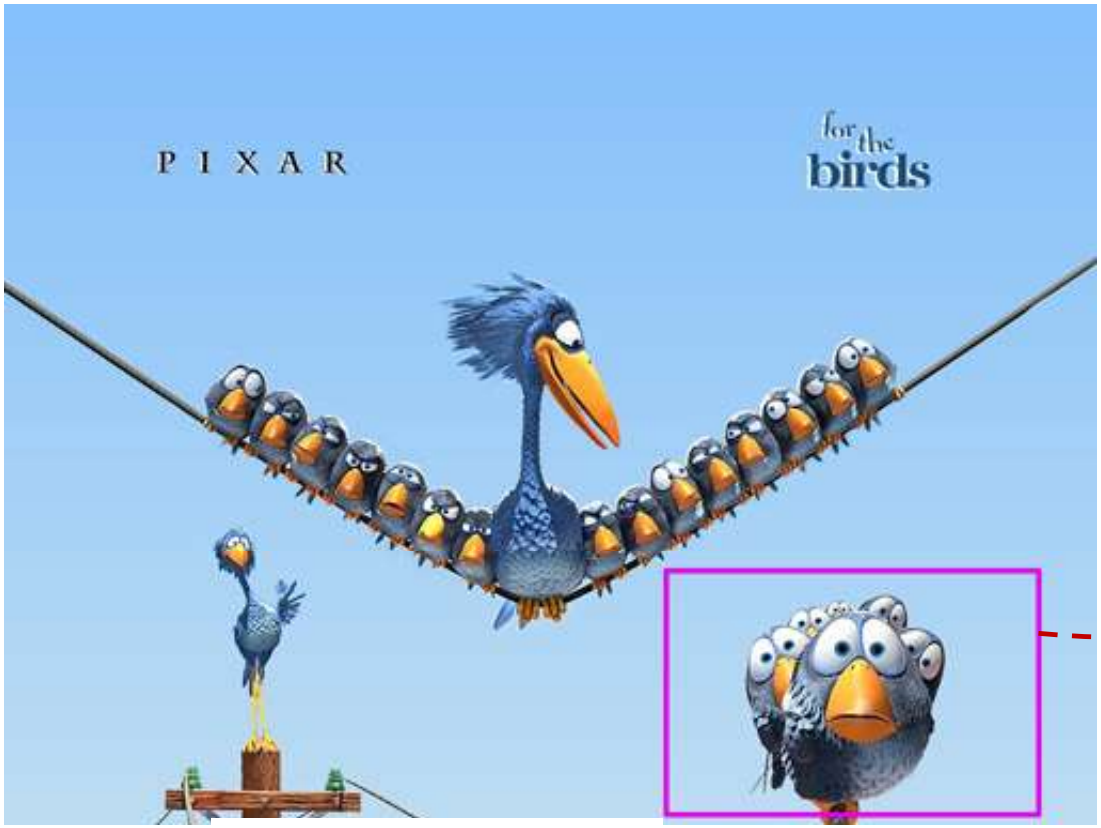
- Para no depender del tipo de dispositivo, se utiliza un sistema de coordenadas normalizado (NDC)
  - NCD: sistema de proyección de tamaño unitario



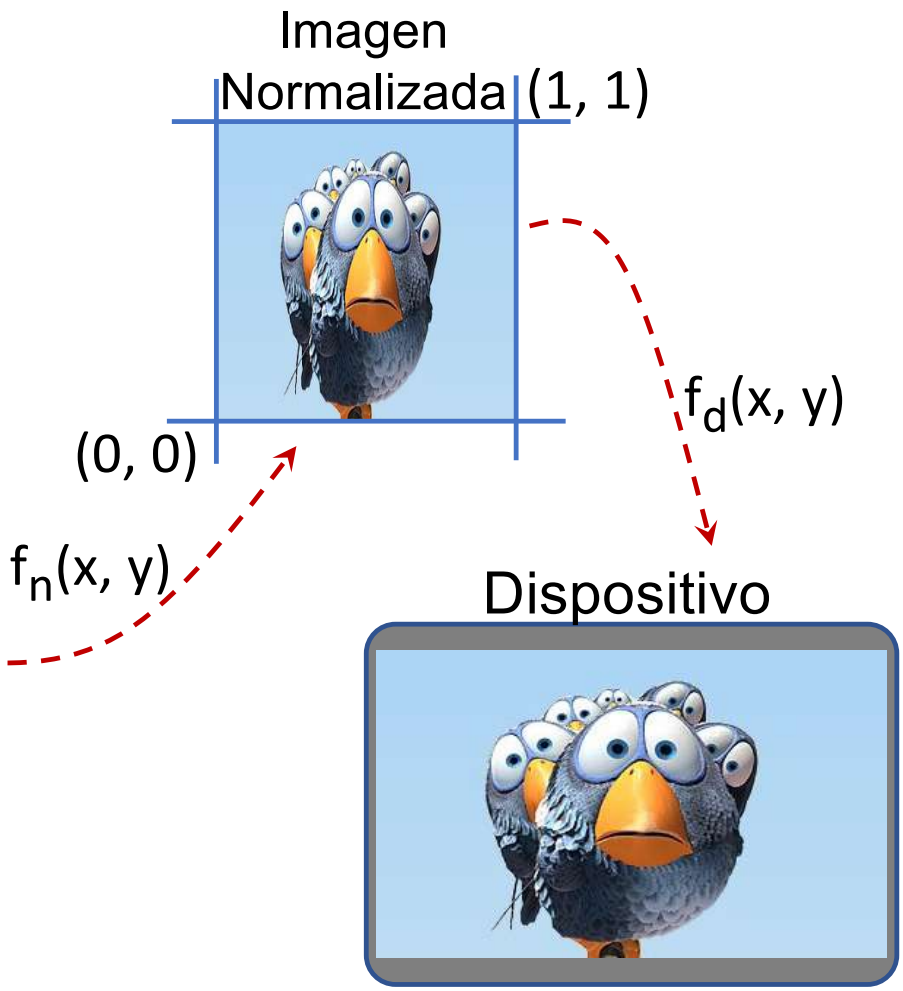
- Una window, en coordenadas WC, es llevada a NDC y luego a DC:
  - Permite independizar la aplicación del dispositivo



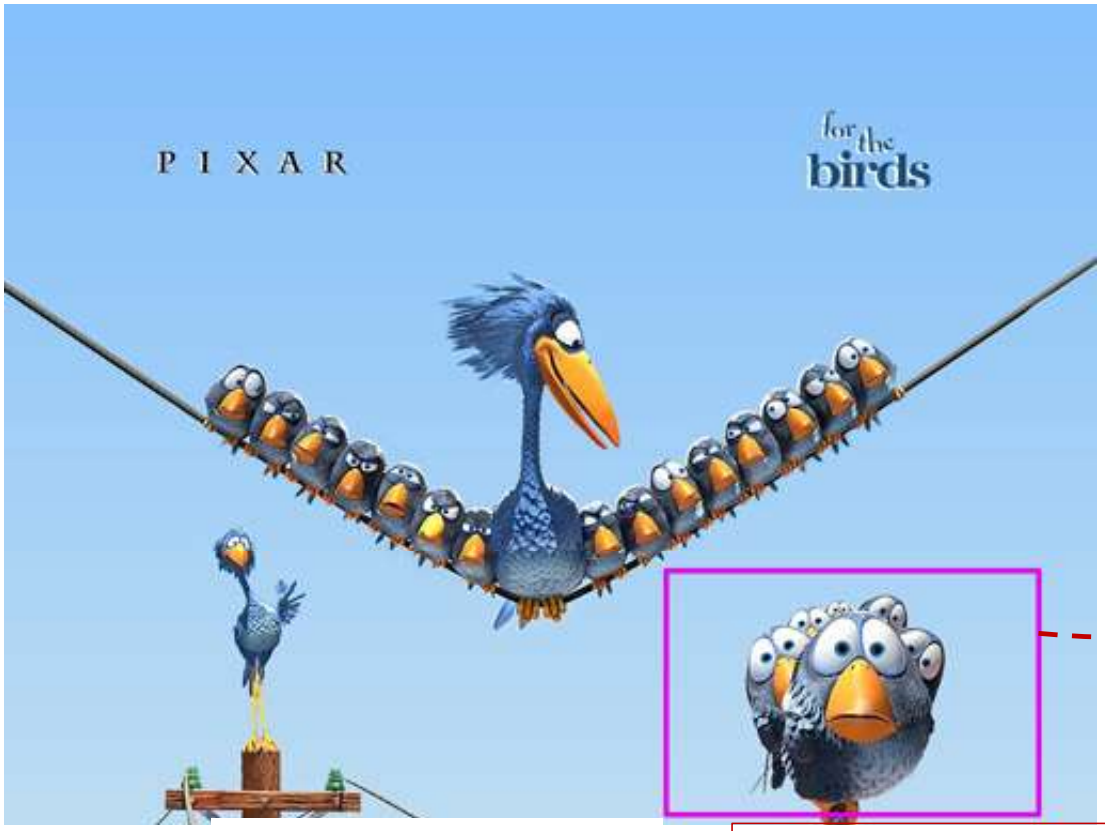
Coordenadas normalizadas



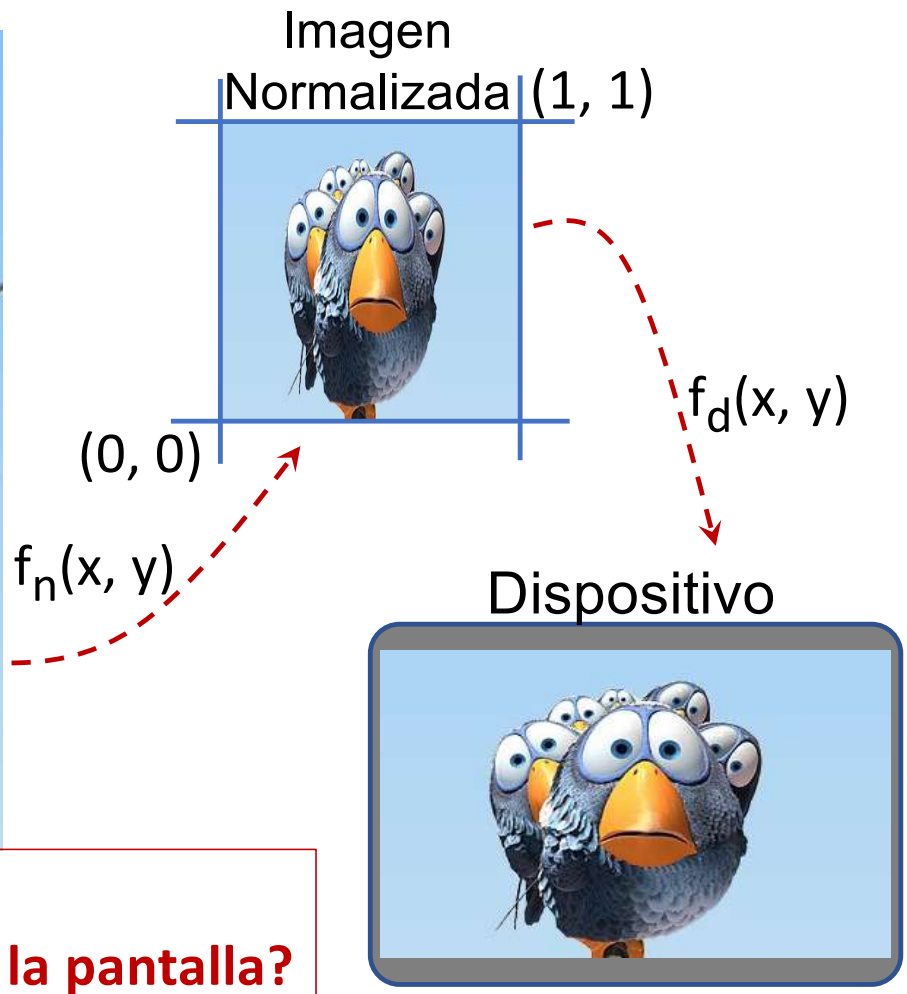
Objeto (mundo)



Coordenadas normalizadas



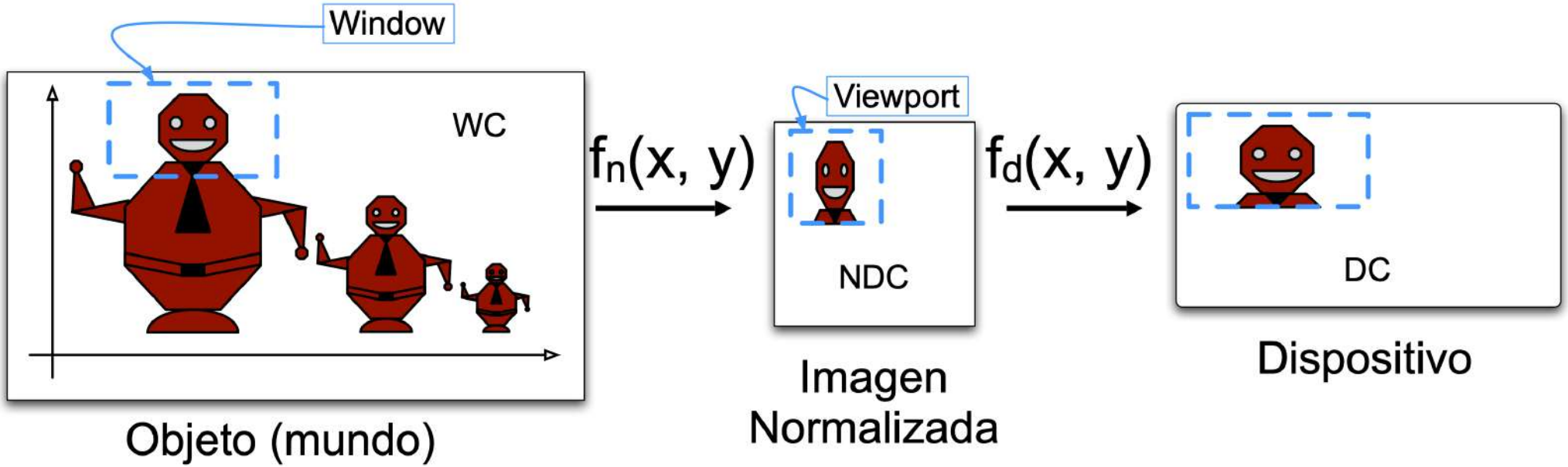
Objeto (mundo)



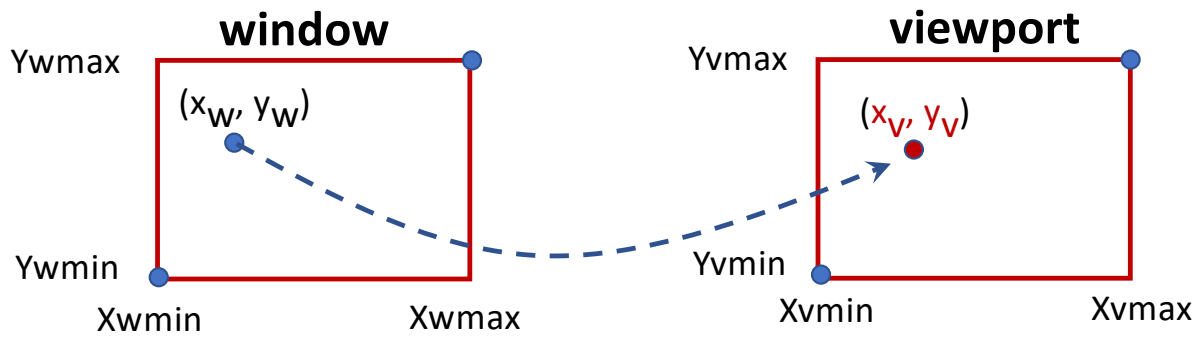
Problema:
¿Se usa toda la pantalla?

Window / Viewport

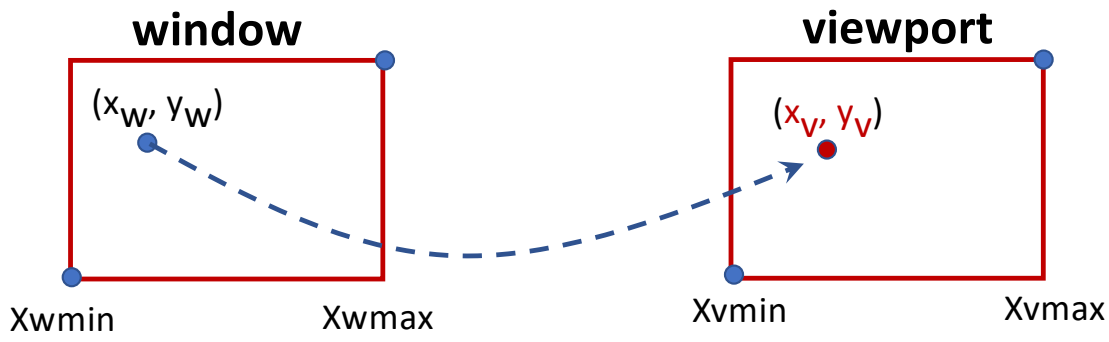
- Window: Zona de interés del WC
- Viewport: Zona de NDC que especifica donde será proyectada la *window*



Window / Viewport



Window / Viewport



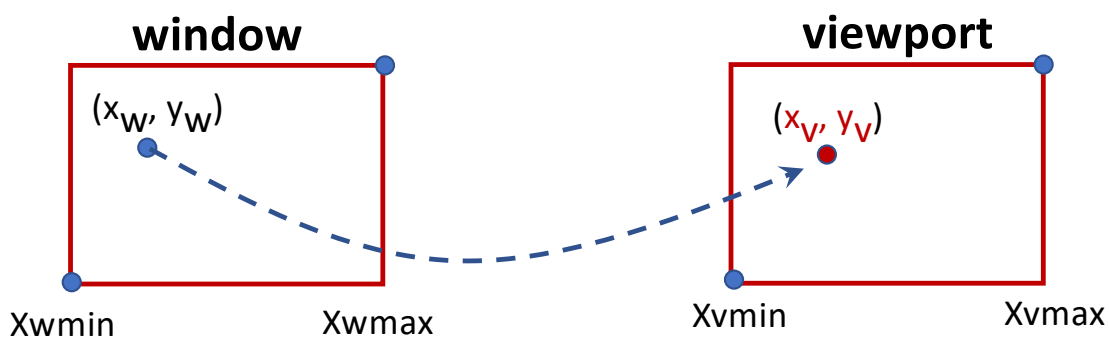
- Usando proporcionalidad

$$\frac{(Xw - Xwmin)}{Xwmax - Xwmin} = \frac{(Xv - Xvmin)}{Xvmax - Xvmin}$$

- Despejando:

$$Xv = (Xw - Xwmin) * (Xvmax - Xvmin) / (Xwmax - Xwmin) + Xvmin$$

Window / Viewport



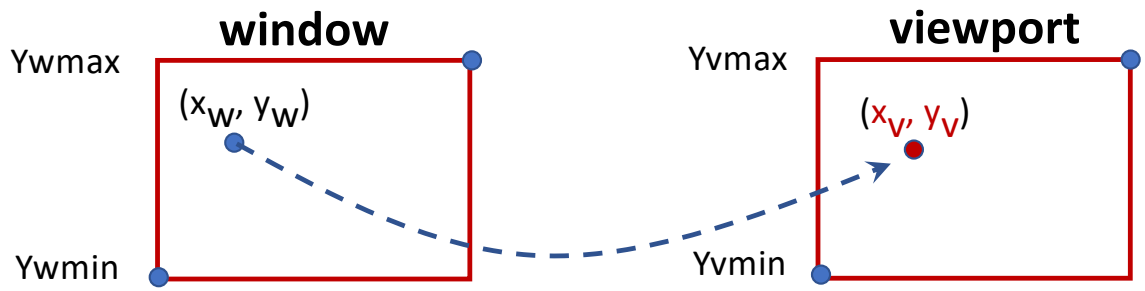
- Usando proporcionalidad

$$\frac{(Xw - Xwmin)}{Xwmax - Xwmin} = \frac{(Xv - Xvmin)}{Xvmax - Xvmin}$$

- Despejando:

$$Xv = (Xw - Xwmin) * \underbrace{(Xvmax - Xvmin) / (Xwmax - Xwmin)}_{Sx \leftarrow \text{constante}} + Xvmin$$

Window / Viewport



- Usando proporcionalidad

$$\frac{(Yw - Ywmin)}{Ywmax - Ywmin} = \frac{(Yv - Yvmin)}{Yvmax - Yvmin}$$

- Despejando:

$$Yv = (Yw - Ywmin) * \underbrace{(Yvmax - Yvmin) / (Ywmax - Ywmin)}_{Sy \leftarrow \text{constante}} + Yvmin$$

Viewing 2D: Proyección

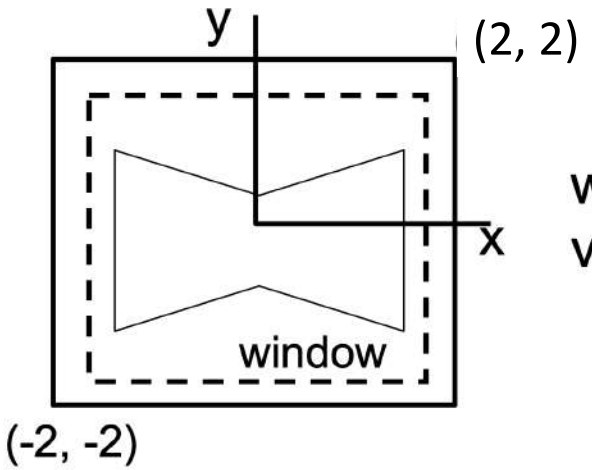
- Sx y Sy: Constante para todo los punto a mapear
- Si Sx es distinto de Sy se observan distorsiones en la figura

- Aspect ratio = (Xmax – Xmin) / (Ymax – Ymin)

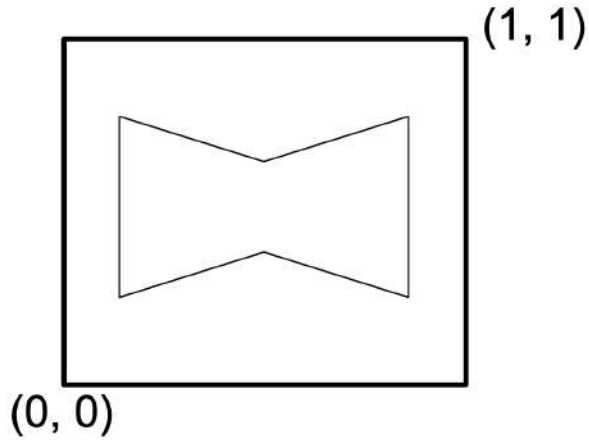
- Aspect Ratio de Window == Aspect Ratio viewport
  →No hay deformaciones 😃

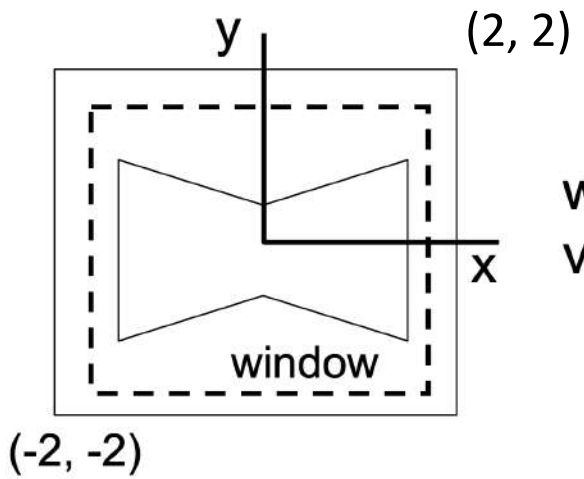
- Propuesto:
  - Determine la matriz de transformación que permite llevar un punto p desde el window al viewport

Ejemplo 1

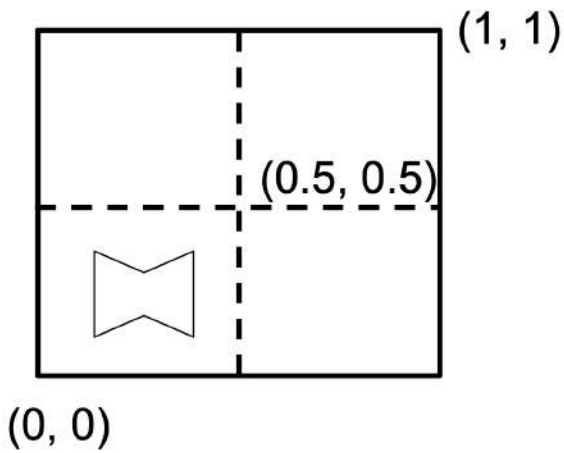


window (-2, -2, 2, 2)
viewport (0, 0, 1, 1)

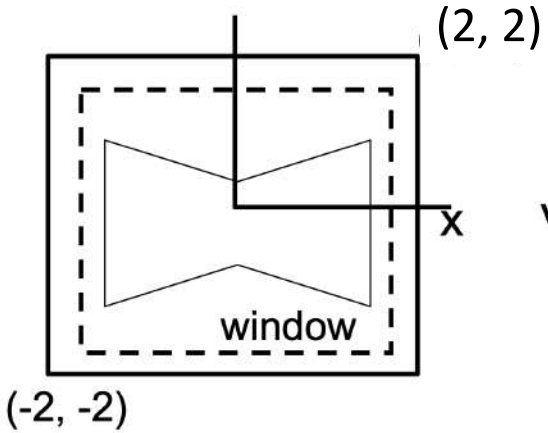




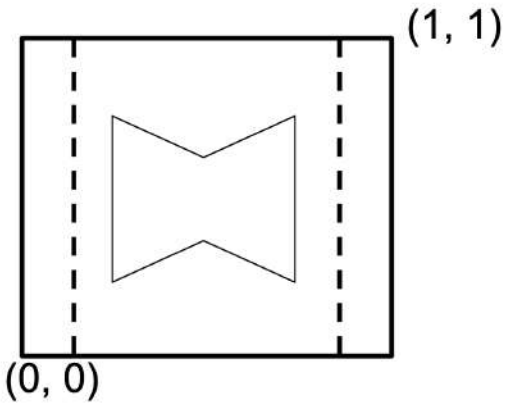
window (-2, -2, 2, 2)
viewport (0, 0, 0.5, 0.5)

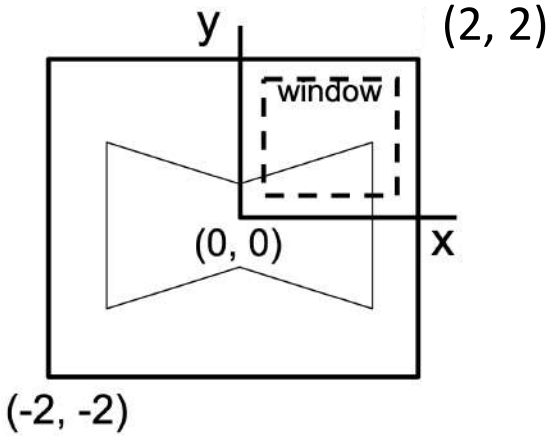


Ejemplo 2:

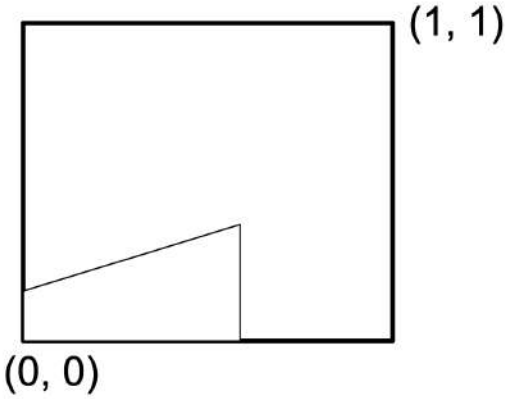


window (-2, -2, 2, 2)
viewport (0.25, 0, 0.75, 1)
(distorsión)





window (0, 0, 2, 2)
viewport (0, 0, 1, 1)
(zoom)



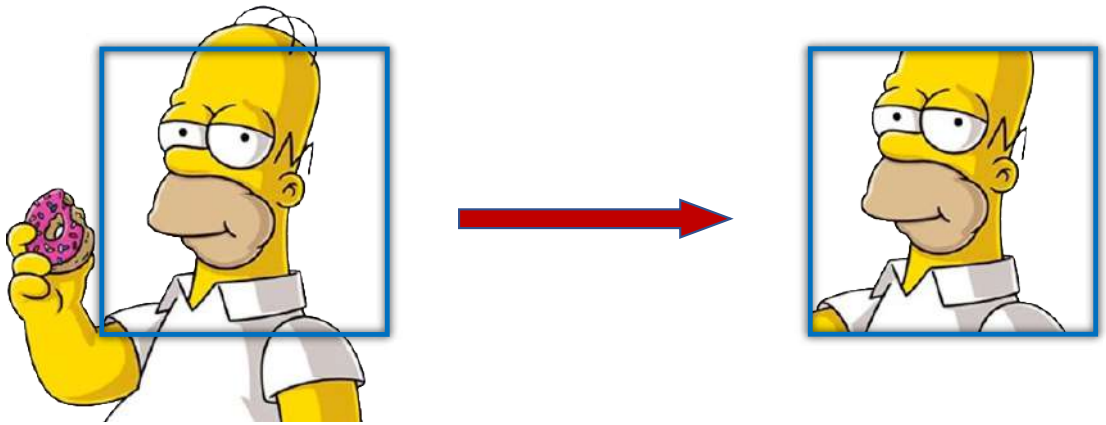
Clipping 2D

Patricio Inostroza, Dr.

dcc CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Clipping 2D

- Por lo general, sólo una parte de una figura es visualizada
- *Window*: zona seleccionada para proyección

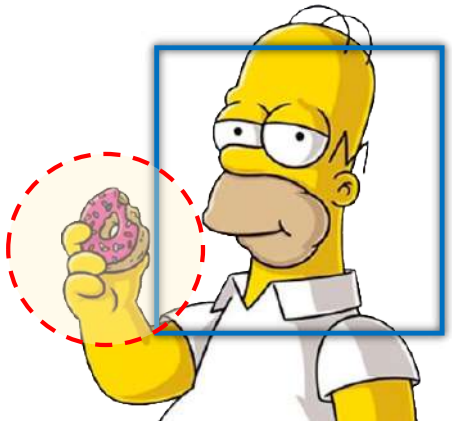


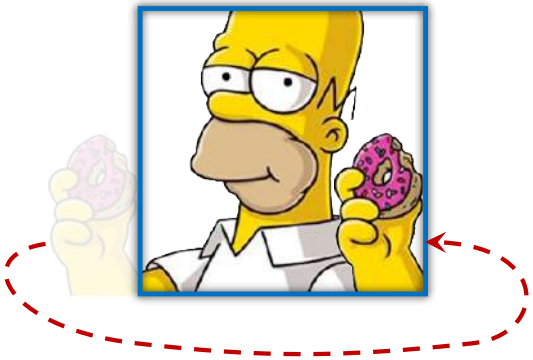
Clipping 2D

- Clipping: técnica que separa los elementos de la figura en visible o no-visible
- Los algoritmo de clipping dependen de la forma de la ventana
- Para efectos del curso, se estudiará clipping en ventanas rectangulares

Clipping 2D

- Qué puede pasar sin el uso de clipping ?





- La sección de la imagen que cae fuera del área puede aparecer por el lado opuesto.

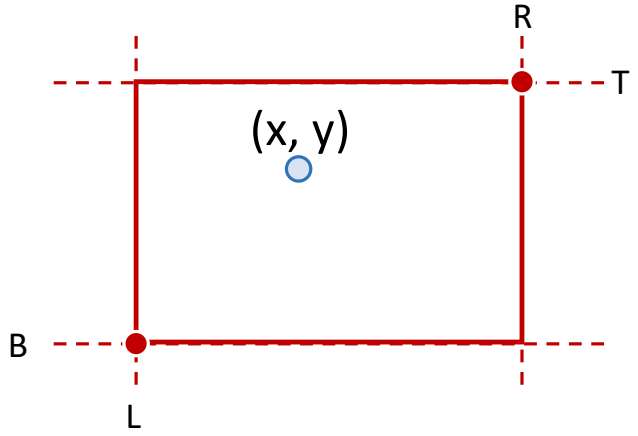
Clipping 2D

- Clipping 2D: Sobre un punto

- Clipping 2D: Sobre una recta
  - Cohen-sutherland
  - Liang-Barsky

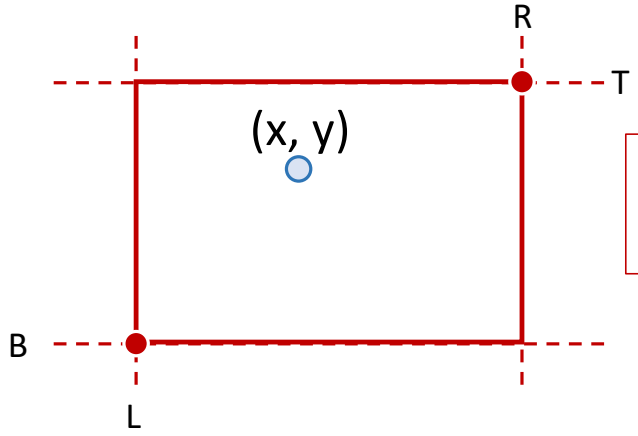
Clipping 2D: sobre un punto

- Clipping 2D
  - Sobre un punto
  - Relativamente simple
    - Determinar si está dentro de Window
    - L <= x <= R and B <= y <= T



Clipping 2D: sobre un punto

- Clipping 2D
  - Sobre un punto
  - Relativamente simple
    - Determinar si está dentro de Window
    - L <= x <= R and B <= y <= T



Pero…
Es ineficiente si separa una figura en puntos!

Clipping 2D: sobre un segmento

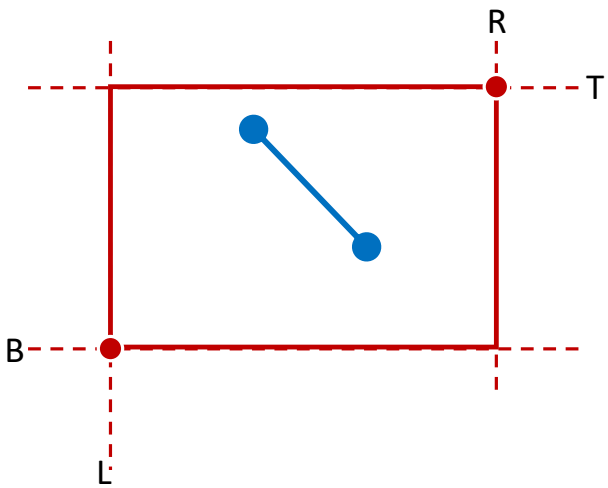
- Algoritmo
  1) Recorrer todo los segmentos y seleccionar aquellos que intersectan la ventana
  2) Recortar los segmentos del paso 1, calculando su intersección con el borde

- El paso 1 puede separar los segmento en:
  - Visibles: ambos extremos están dentro de la ventana
  - Invisibles: los puntos extremos están fuera*
  - Indeterminado: el resto

(*): Falta una condición 🤔

Clipping 2D: sobre un segmento

- (*) Condición faltante
- Es invisible si el segmento cumple:
  - x1 y x2 < L
  - x1 y x2 > R
  - y1 e y2 < B
  - y1 e y2 > T



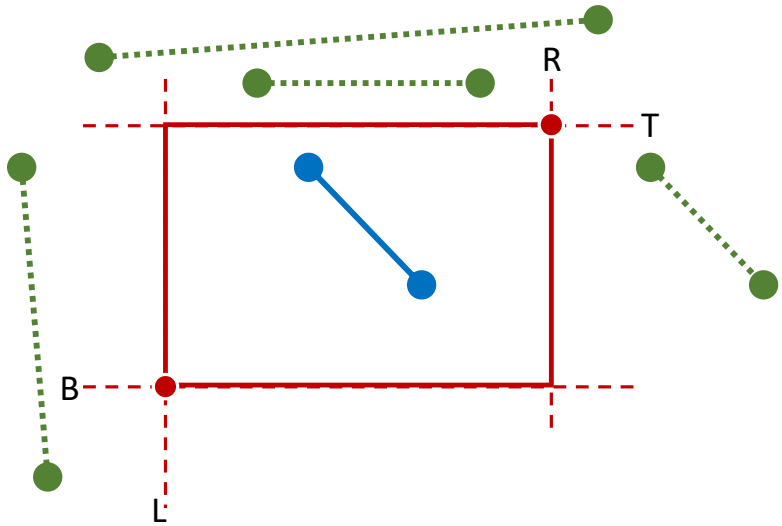
Visible :

Invisible :

Indeterminado:

Clipping 2D: sobre un segmento

- (*) Condición faltante
- Es invisible si el segmento cumple:
  - x1 y x2 < L
  - x1 y x2 > R
  - y1 e y2 < B
  - y1 e y2 > T



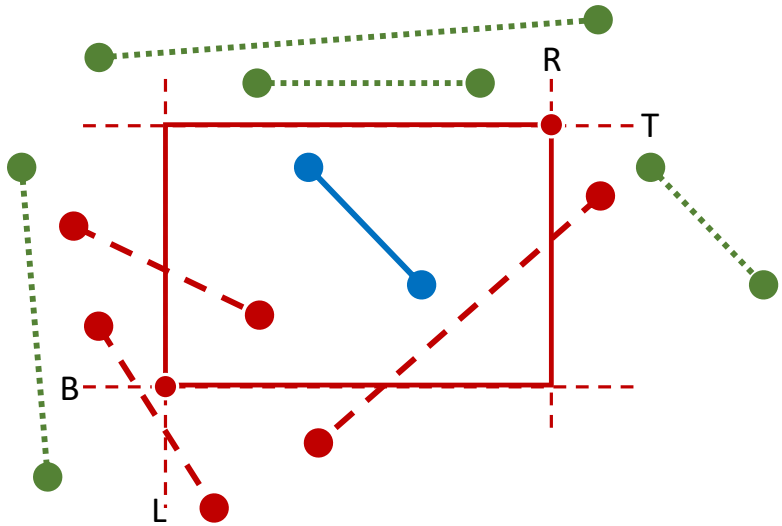
Visible :

Invisible :

Indeterminado:

Clipping 2D: sobre un segmento

- (*) Condición faltante
- Es invisible si el segmento cumple:
  - x1 y x2 < L
  - x1 y x2 > R
  - y1 e y2 < B
  - y1 e y2 > T



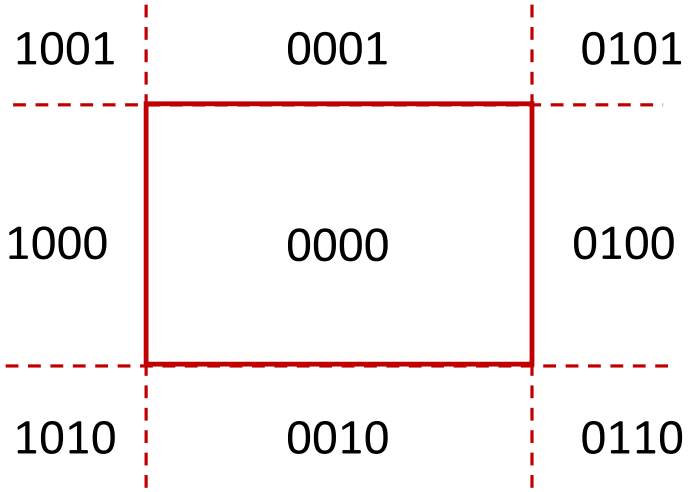
Visible :

Invisible :

Indeterminado:

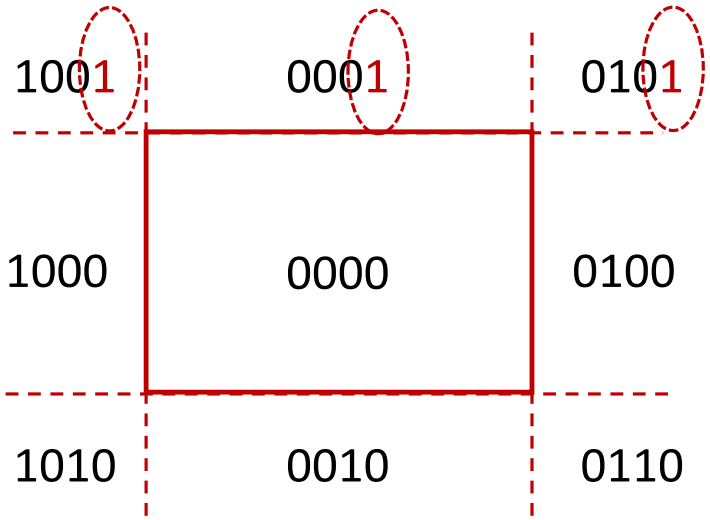
Clipping 2D: Cohen-sutherland

- Algoritmo de Cohen-sutherland
- Se asigna un código de 4 bits a las nueve regiones



Clipping 2D: Cohen-sutherland

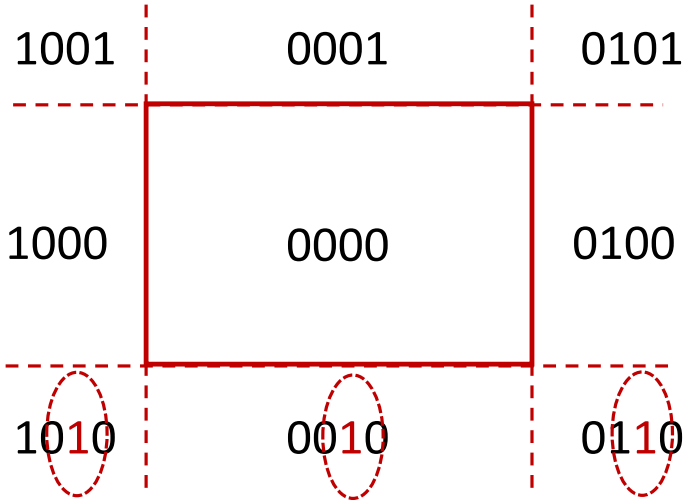
- Algoritmo de Cohen-sutherland
- Se asigna un código de 4 bits a las nueve regiones



- Bit 1: 1 punto extremo sobre la *window*
- Bit 2: 1 punto extremo bajo la *window*
- Bit 3: 1 punto extremo a la derecha de la *window*
- Bit 4: 1 punto extremo a la izquierda de la *window*

Clipping 2D: Cohen-sutherland

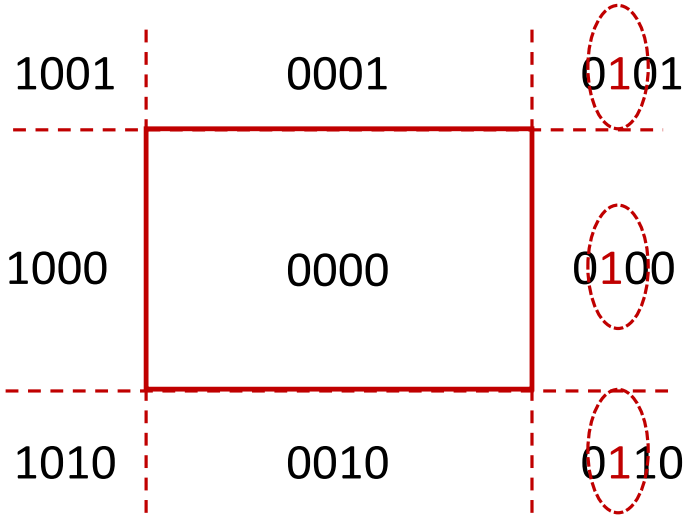
- Algoritmo de Cohen-sutherland
- Se asigna un código de 4 bits a las nueve regiones



- Bit 1: 1 punto extremo sobre la *window*
- Bit 2: 1 punto extremo bajo la *window*
- Bit 3: 1 punto extremo a la derecha de la *window*
- Bit 4: 1 punto extremo a la izquierda de la *window*

Clipping 2D: Cohen-sutherland

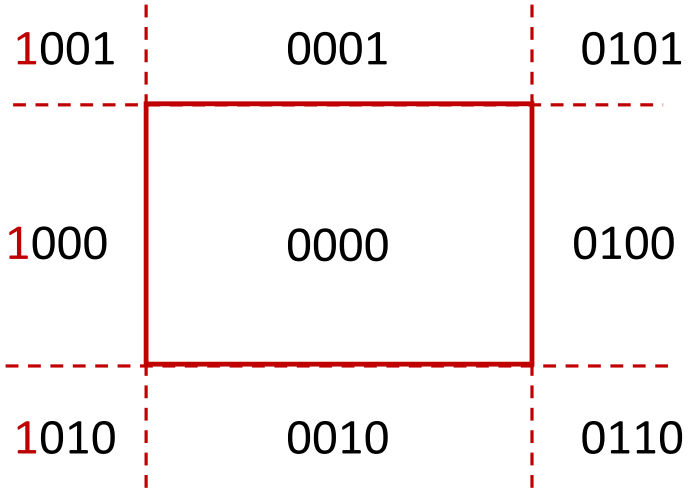
- Algoritmo de Cohen-sutherland
- Se asigna un código de 4 bits a las nueve regiones



- Bit 1: 1 punto extremo sobre la *window*
- Bit 2: 1 punto extremo bajo la *window*
- Bit 3: 1 punto extremo a la derecha de la *window*
- Bit 4: 1 punto extremo a la izquierda de la *window*

Clipping 2D: Cohen-sutherland

- Algoritmo de Cohen-sutherland
- Se asigna un código de 4 bits a las nueve regiones



- Bit 1: 1 punto extremo sobre la *window*
- Bit 2: 1 punto extremo bajo la *window*
- Bit 3: 1 punto extremo a la derecha de la *window*
- Bit 4: 1 punto extremo a la izquierda de la *window*

Una pausa...

- En computación hay dos tipo de *"and"*

1. *"and"* lógico

```
if a <2 and b != 5 :
    print ("hola todos")
```

2. "&" binario (entre bit's)

0 & 0 → 0

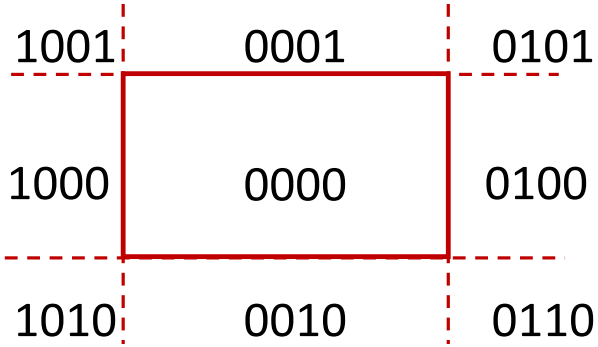
1 & 0 → 0

1 & 1 → 1

1010 & 1001 → 1000

Clipping 2D: Cohen-sutherland

- pseudo algoritmo



si codigoPunto1 == 0 and codigoPunto2 == 0 :

    segmento visible

si (codigoPunto1 & codigoPunto2) != 0 :

    segmento no-visible

sino:

    Indeterminado 😟

Clipping 2D: Cohen-sutherland

si codigoPunto1 == 0 and codigoPunto2 == 0 :

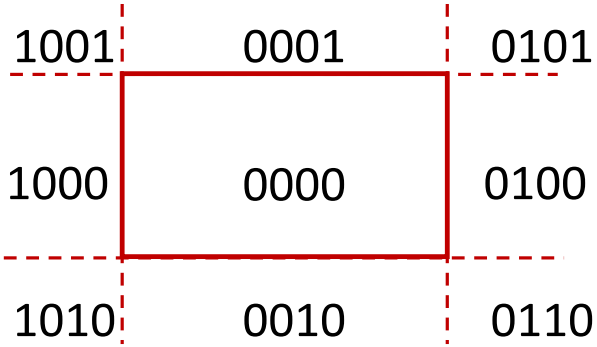
segmento visible

si (codigoPunto1 & codigoPunto2) != 0 :

segmento no-visible

sino:

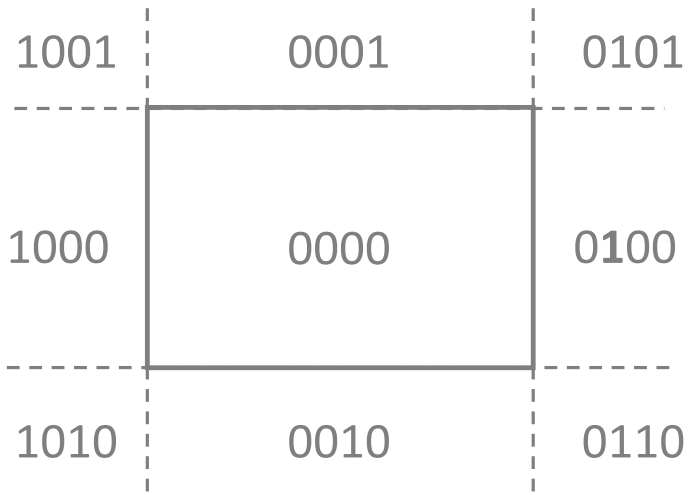
Indeterminado



- Hay que calcular los punto en la intersección con Window
  - ¿Con todos los bordes?
  - ¿Cuántos lados de Window pueden ser intersectados por un segmento?

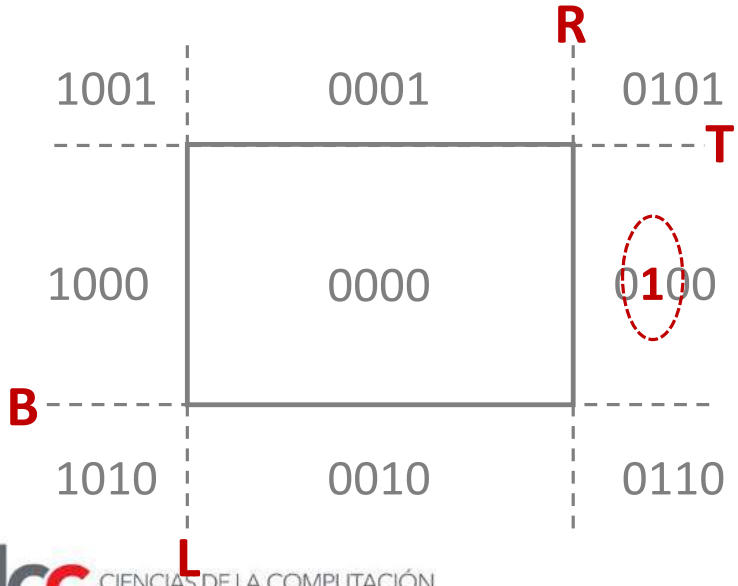
Clipping 2D: Cohen-sutherland

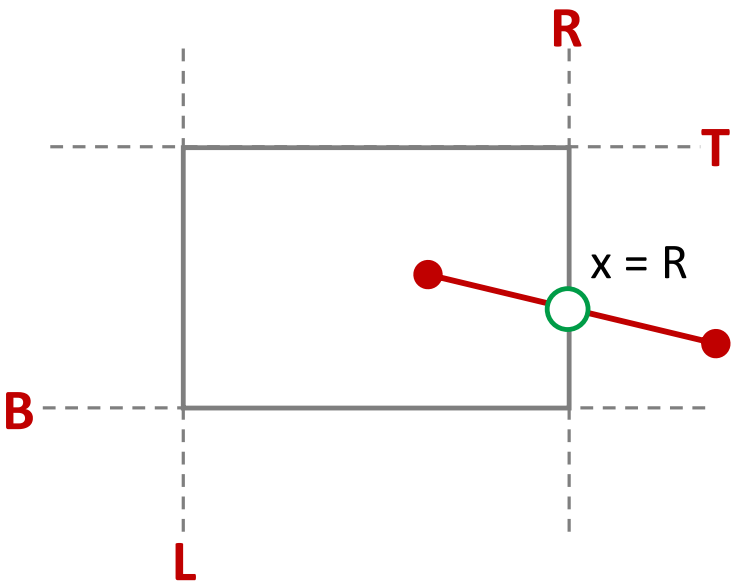
- Análisis de bits en la intersección
  - Bit 1 = 1 → intersección con y = T
  - Bit 2 = 1 → intersección con y = B
  - Bit 3 = 1 → intersección con x = R
  - Bit 4 = 1 → intersección con x = L



Clipping 2D: Cohen-sutherland

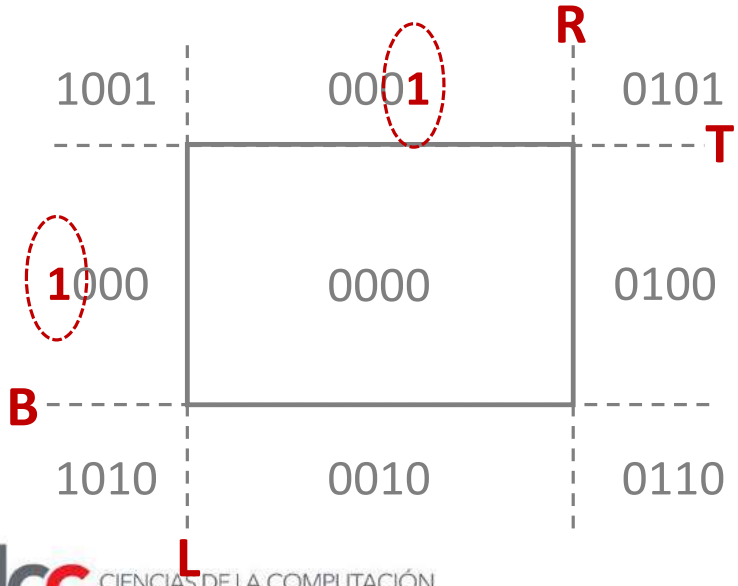
- Análisis de bits en la intersección
  - Bit 1 = 1 → intersección con y = T
  - Bit 2 = 1 → intersección con y = B
  - Bit 3 = 1 → intersección con x = R
  - Bit 4 = 1 → intersección con x = L

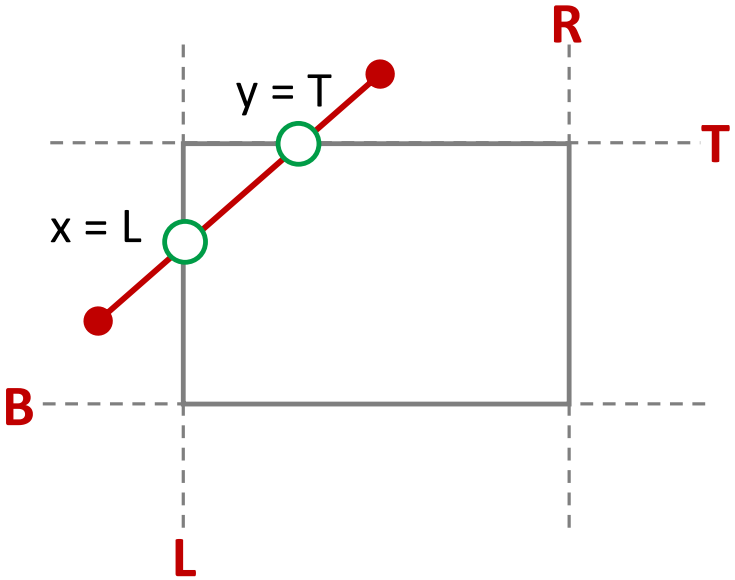




Clipping 2D: Cohen-sutherland

- Análisis de bits en la intersección
  - Bit 1 = 1 → intersección con y = T
  - Bit 2 = 1 → intersección con y = B
  - Bit 3 = 1 → intersección con x = R
  - Bit 4 = 1 → intersección con x = L



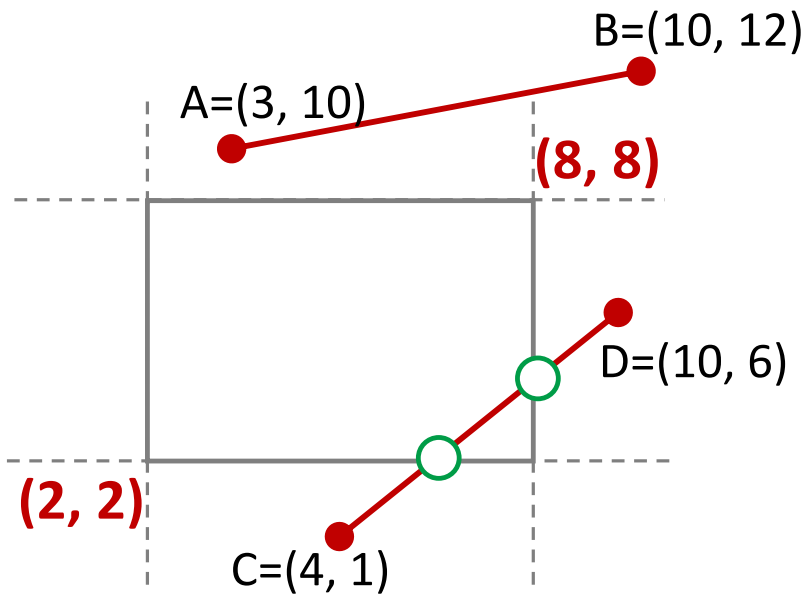


Ejemplo

- Ecuación paramétrica

$$\left.\begin{aligned} x &= x1 + \mathbf{t}\,(x2 - x1) \\ y &= y1 + \mathbf{t}\,(y2 - y1) \end{aligned}\right\}$$

Cuando 0 <= **t** <= 1, el punto está dentro del segmento



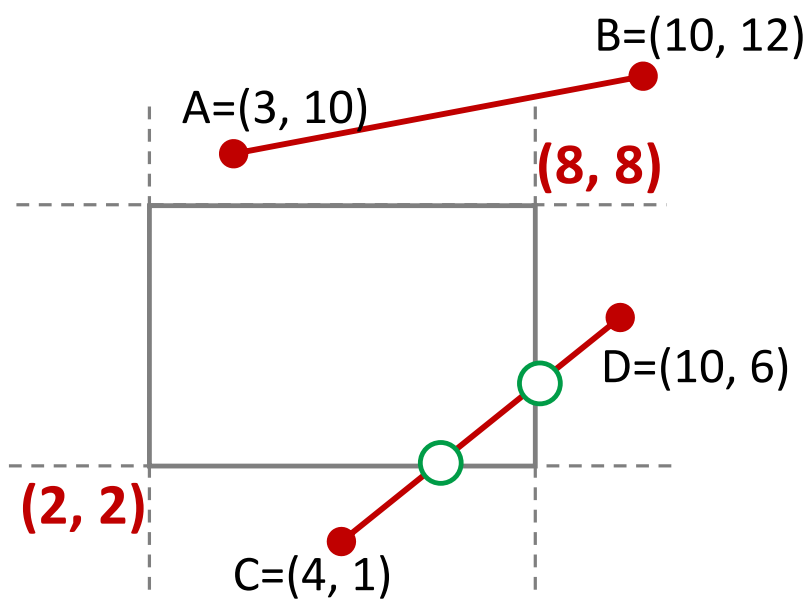
Ejemplo

- Ecuación paramétrica

$$\left.\begin{aligned} x &= x1 + \mathbf{t}\,(x2 - x1) \\ y &= y1 + \mathbf{t}\,(y2 - y1) \end{aligned}\right\}$$

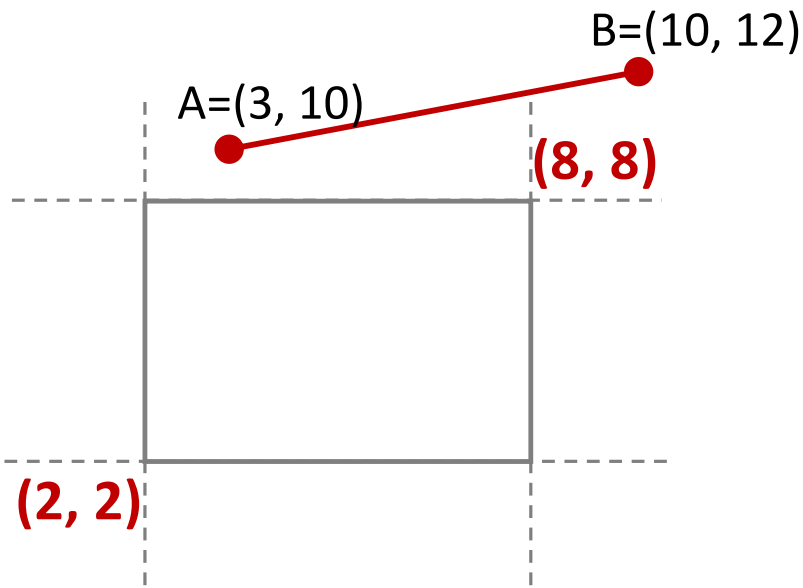
Cuando 0 <= **t** <= 1, el punto está dentro del segmento

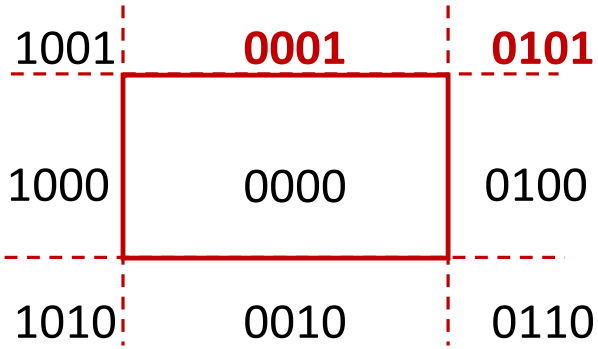
- Problema:
  - Determinar la visibilidad de los segmentos de recta AB y CD sobre un *window* limitado por las coordenadas (2, 2) y (8, 8).



Ejemplo

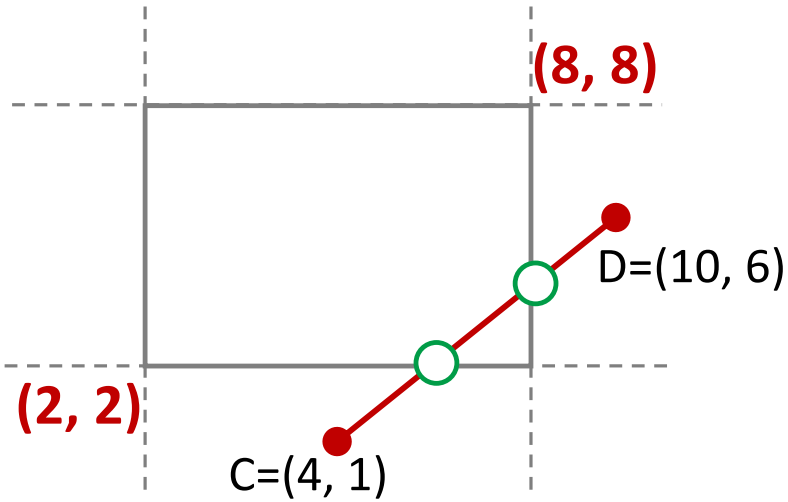
- Solución para el segmento AB
  - Código del punto A → 0001
  - Código del punto B → 0101
  - Dado que (0001 & 0101) != 0 entonces AB no es visible

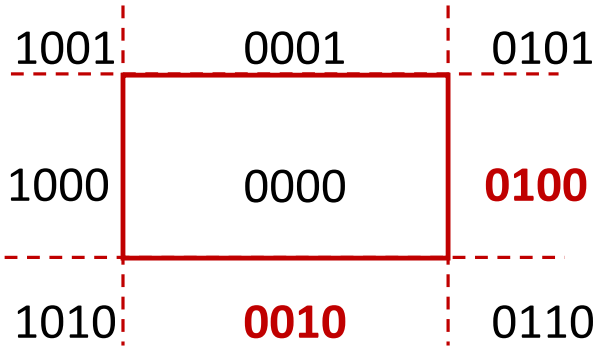




Ejemplo

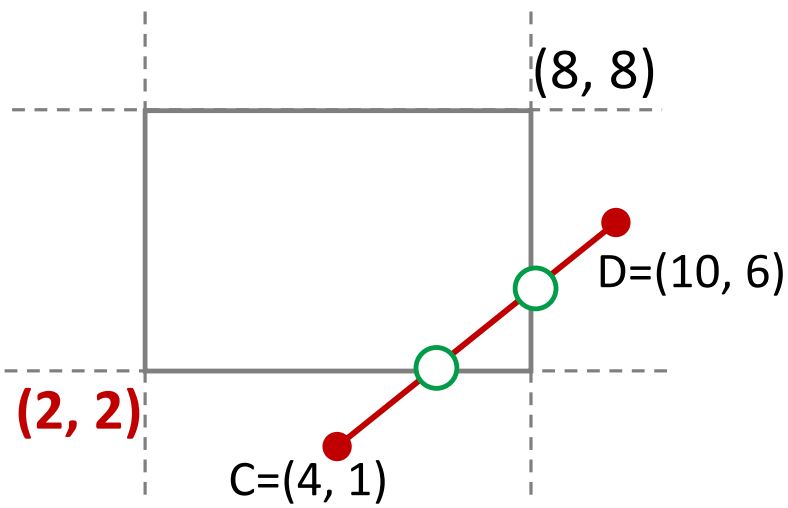
- Solución para el segmento AB
  - Código del punto C → 0010
  - Código del punto D → 0100
  - Dado que (0010 & 0100) == 0 entonces no se sabe 🤔

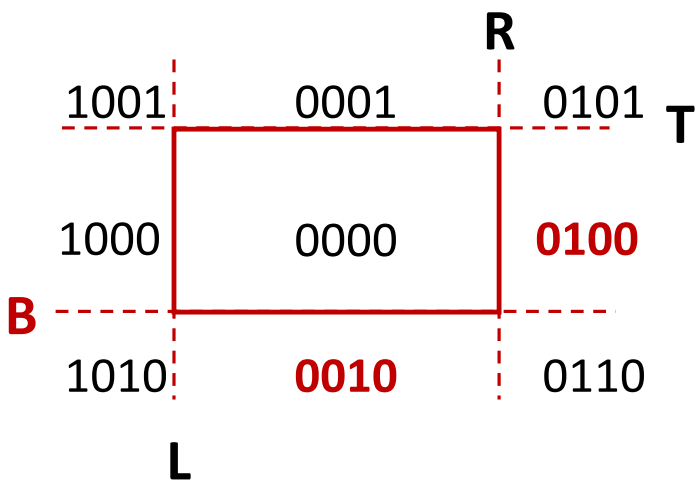




Ejemplo

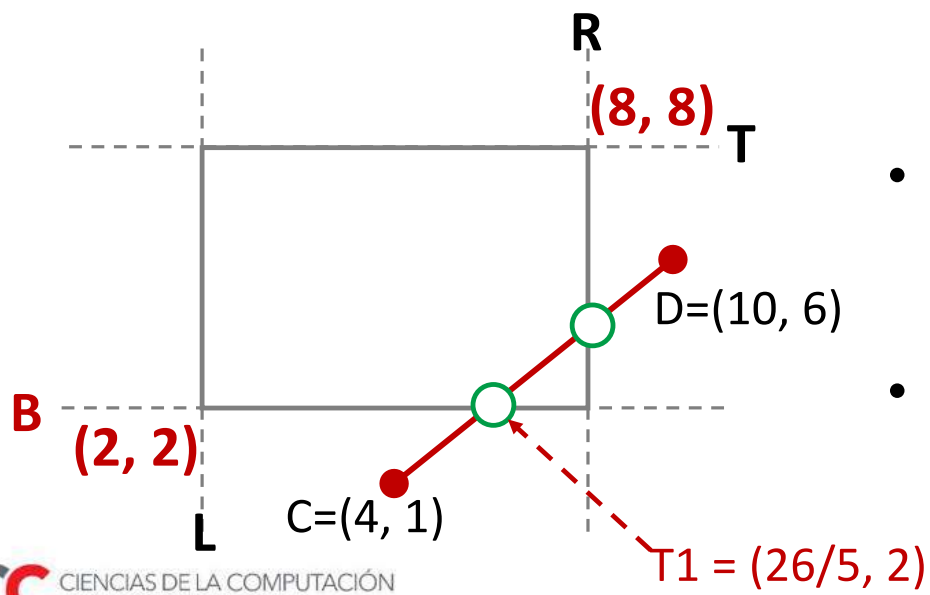
- Solución para el segmento CD
- x = 4 + t (10 – 4) = 4 + 6 t
- y = 1 + t (6 – 1) = 1 + 5 t
- C = (4, 1) → código = 0010 → 2do bits intersecta en B = 2





Ejemplo

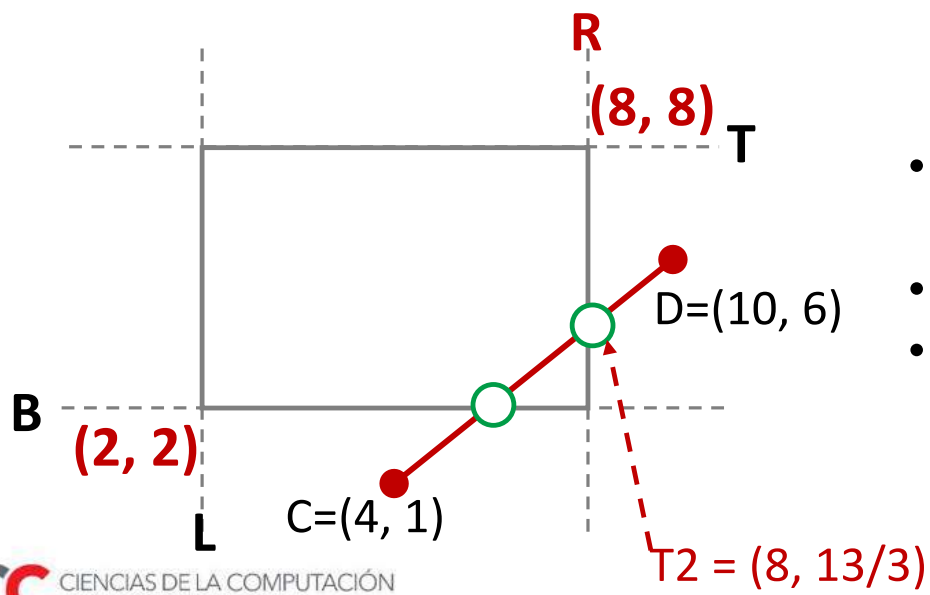
- Solución para el segmento CD
- x = 4 + t (10 – 4) = 4 + 6 t
- y = 1 + t (6 – 1) = 1 + 5 t
- C = (4, 1) → código = 0010 → 2do bits intersecta en B = 2



- Reemplazando: y = 2
  - 2 = 1 + 5 t → t = 1/5
  - x = 4 + 6 (1/5) = 26/5
- Punto de intersección: T1 = (26/5, 2)

Ejemplo

- Solución para el segmento CD
- D = (10, 6) → código = 0100 → 3er bits intersecta en R = 8
- x = 4 + 6 t
- y = 1 + 5 t

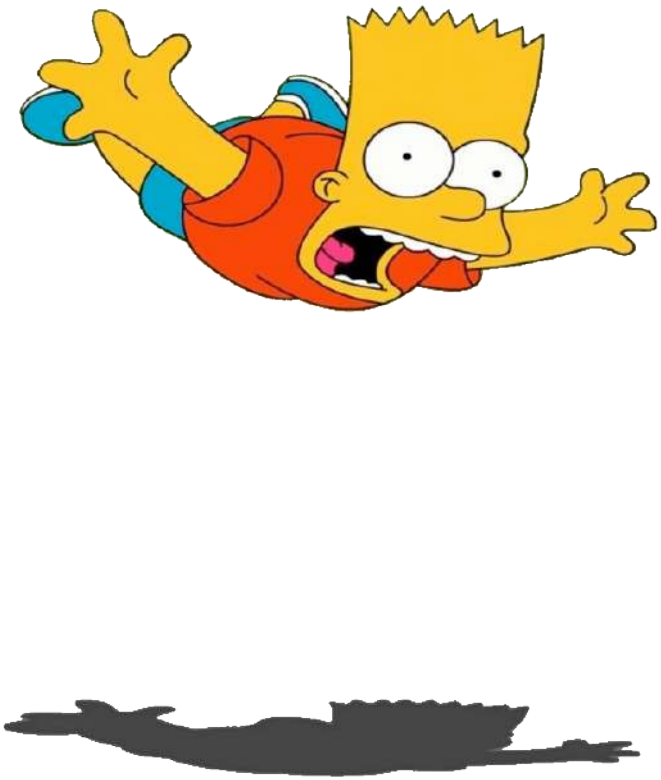


- Reemplazando: x = 8
  - 8 = 4 + 6 t → t = 2/3
  - y = 1 + 5 (2/3) = 13/3
- Punto de intersección: T2 = (8, 13/3)

- T1 y T2 tiene código 0000
- Luego:
  - segmento T1T2 está dentro de *window*

Clipping 2D: Liang-Barsky

- Un segundo algoritmo de Clipping
- Liang-Barsky



Clipping 2D: Liang-Barsky

- Recordemos que la zona de clipping está limitada por:
  - $L <= x <= R$
  - $B <= y <= T$

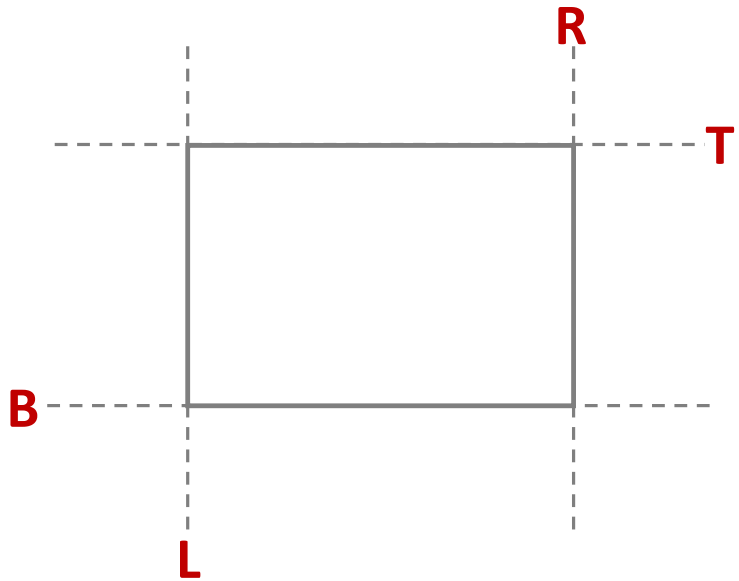
- Además:
  - $x = x1 + t\,(x2 - x1)$ donde: $0 <= t <= 1$
  - $z = y1 + t\,(y2 - y1)$

- Definiendo:
  - $dx = (x2 - x1)$
  - $dy = (y2 - y1)$

Clipping 2D: Liang-Barsky

- Definiendo:
  - dx = (x2 – x1) ; dy = (y2 – y1)
  - d1 = –dx ; d3 = –dy
  - d2 = dx ; d4 = dy

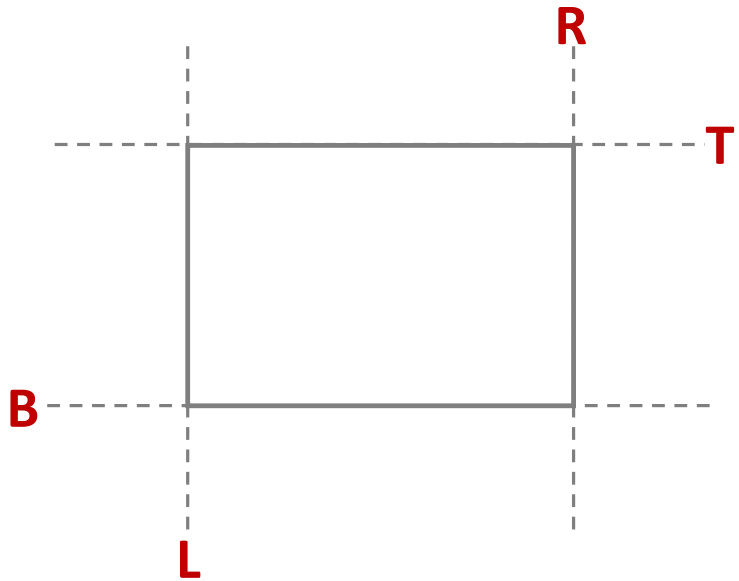
- Además:
  - q1 = x1 – L ; q3 = y1 – B
  - q2 = R – X1 ; q4 = T – y1



Clipping 2D: Liang-Barsky

- Definiendo:
  - dx = (x2 – x1) ; dy = (y2 – y1)
  - d1 = –dx ; d3 = –dy
  - d2 = dx ; d4 = dy

- Además:
  - q1 = x1 – L ; q3 = y1 – B
  - q2 = R – X1 ; q4 = T – y1



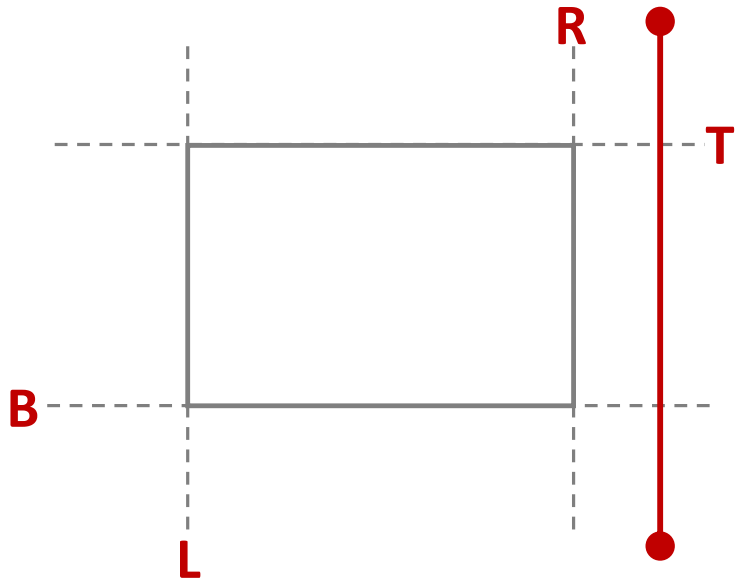
- Primer resultado:
  - Si d1 = d2 = 0 → segmento paralelo el eje y
  - Si d3 = d4 = 0 → segmento paralelo el eje x

Clipping 2D: Liang-Barsky

- Definiendo:
  - dx = (x2 – x1) ; dy = (y2 – y1)
  - d1 = –dx ; d3 = –dy
  - d2 = dx ; d4 = dy

- Además:
  - q1 = x1 – L ; q3 = y1 – B
  - q2 = R – X1 ; q4 = T – y1

- Mejorando el resultado:
  - Si d1 = 0 && q1 < 0 → no visible
  - Si d2 = 0 && q2 < 0 → no visible
  - Si d3 = 0 && q3 < 0 → no visible
  - Si d4 = 0 && q4 < 0 → no visible



d2 = 0 && q2 < 0
→ No visible

Clipping 2D: Liang-Barsky

- Caso General
- Recordando que:
  - $x = x1 + t\,(x2 - x1)$; donde $t \in [0, 1]$
- Reemplazando por el punto del borde $x = L$:
  - $L = x1 + t\,dx$; *reordenando*
  - $t = (L - x1) / dx$; *pero* $(x1 - L) = q1$ y $d1 = -dx$
  - $t = q1 / d1$

- Definiendo:
  - $dx = (x2 - x1)$; $dy = (y2 - y1)$
  - $d1 = -dx$; $d3 = -dy$
  - $d2 = dx$; $d4 = dy$
- Además:
  - $q1 = x1 - L$; $q3 = y1 - B$
  - $q2 = R - X1$; $q4 = T - y1$

Clipping 2D: Liang-Barsky

- Caso General
- Recordando que:
  - $x = x1 + t\,(x2 - x1)$; donde $t \in [0, 1]$
- Reemplazando por el punto del borde $x = L$:
  - $L = x1 + t\,dx$; *reordenando*
  - $t = (L - x1) / dx$; *pero* $(x1 - L) = q1$ y $d1 = -dx$
  - $t = q1 / d1$
- Reemplazando por el punto del borde $x = R$:
  - $R = x1 + t\,dx$; *reordenando*
  - $t = (R - x1) / dx$; *pero* $(R - x1) = q2$ y $d2 = dx$
  - $t = q2 / d2$

- Definiendo:
  - $dx = (x2 - x1)$; $dy = (y2 - y1)$
  - $d1 = -dx$; $d3 = -dy$
  - $d2 = dx$; $d4 = dy$
- Además:
  - $q1 = x1 - L$; $q3 = y1 - B$
  - $q2 = R - X1$; $q4 = T - y1$

Clipping 2D: Liang-Barsky

- Recordando que:
  - $y = y1 + t\,(y2 - y1)$; *donde* $t \in [0, 1]$
- Reemplazando por el punto del borde $y = B$:
  - $B = y1 + t\,dy$; *reordenando*
  - $t = (B - y1) / dy$; *pero* $(y1 - B) = q3$ y $d3 = -dy$
  - $t = q3 / d3$
- reemplazando por el punto del borde $y = T$:
  - $T = y1 + t\,dy$; *reordenando*
    $t = (T - y1) / dy$; *pero* $(T - y1) = q4$ y $d4 = dy$
  - $t = q4 / d4$

Clipping 2D: Liang-Barsky

- Resumiendo:
  - La intersección con los limites de la zona de clipping ocurre en $t = qi / di$ con (i=1,..,4)

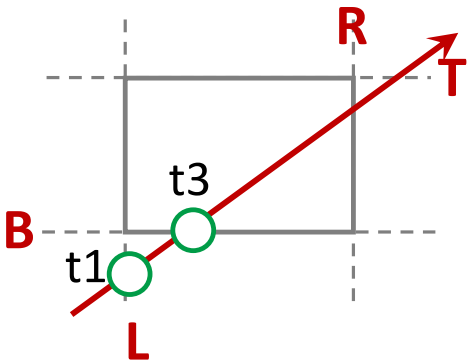
- Falta determinar los dos valores de t que definen la intersección del segmento con la zona de clipping

- Llamemos a estos límites ta y tb

Clipping 2D: Liang-Barsky

- Analizando los casos para di < 0 :

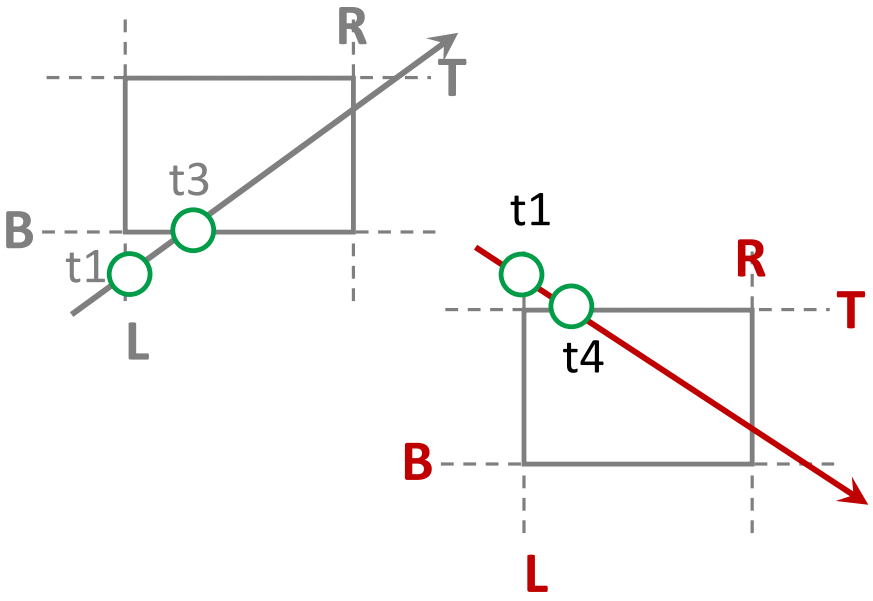
d1 < 0	d1 < 0	d2 < 0	d2 < 0
d3 < 0	d4 < 0	d4 < 0	d3 < 0



Clipping 2D: Liang-Barsky

- Analizando los casos para di < 0 :

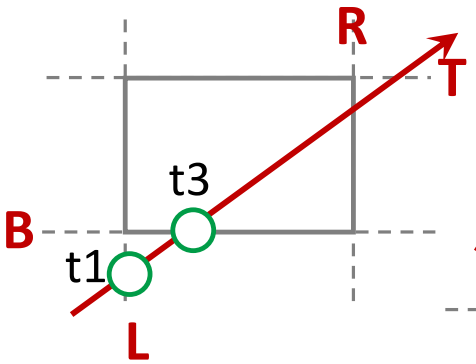
d1 < 0	d1 < 0	d2 < 0	d2 < 0
d3 < 0	d4 < 0	d4 < 0	d3 < 0



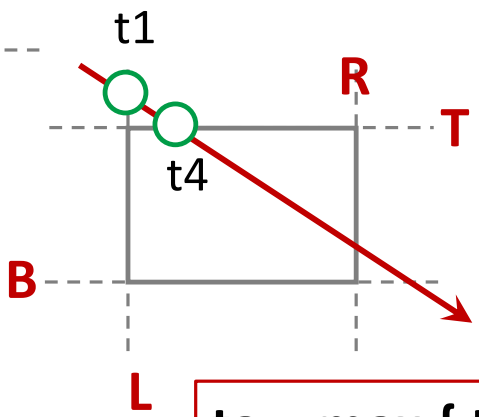
Clipping 2D: Liang-Barsky

- Analizando los casos para di < 0 :

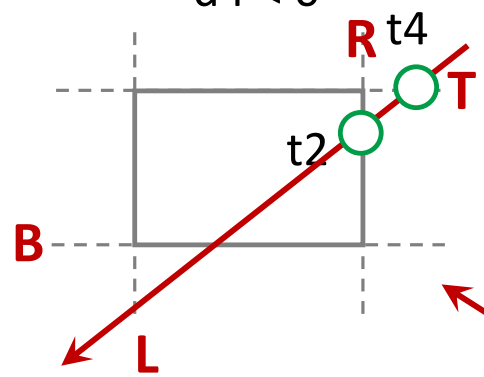
d1 < 0
d3 < 0



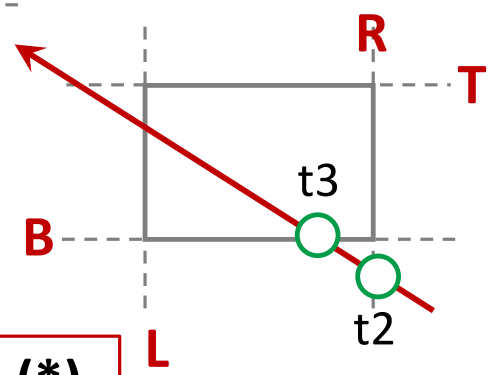
d1 < 0
d4 < 0



d2 < 0
d4 < 0



d2 < 0
d3 < 0



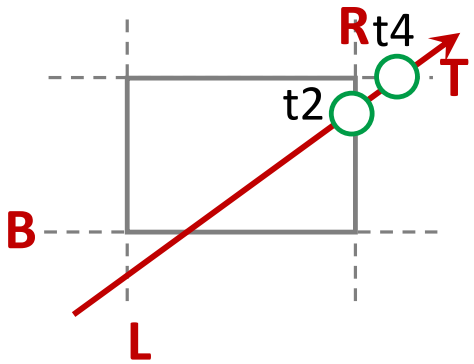
ta = max { ti = qi/di | di < 0 } (*)

(*) : Falta una condición

Clipping 2D: Liang-Barsky

- Analizando los casos para di > 0 :

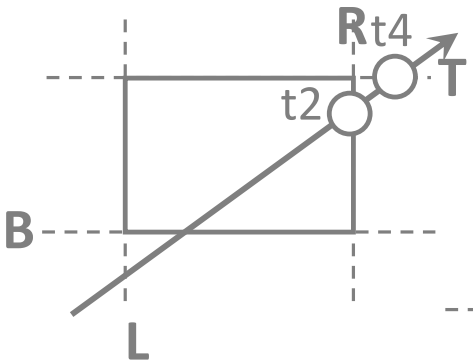
d2 > 0	d2 > 0	d1 > 0	d1 > 0
d4 > 0	d3 > 0	d3 > 0	d4 > 0

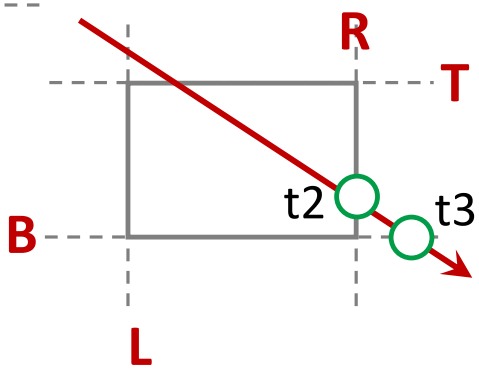


Clipping 2D: Liang-Barsky

- Analizando los casos para di > 0 :

d2 > 0	d2 > 0	d1 > 0	d1 > 0
d4 > 0	d3 > 0	d3 > 0	d4 > 0

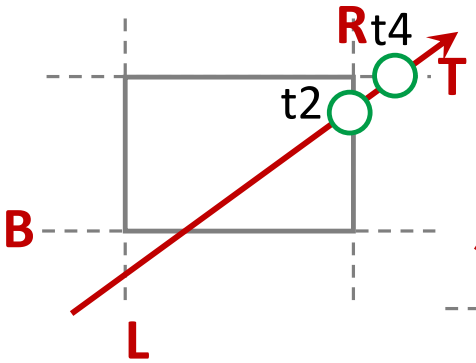




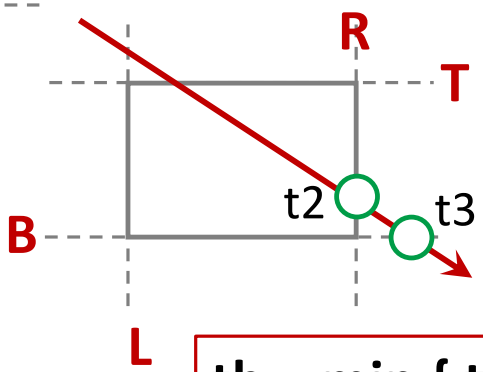
Clipping 2D: Liang-Barsky

- Analizando los casos para di > 0 :

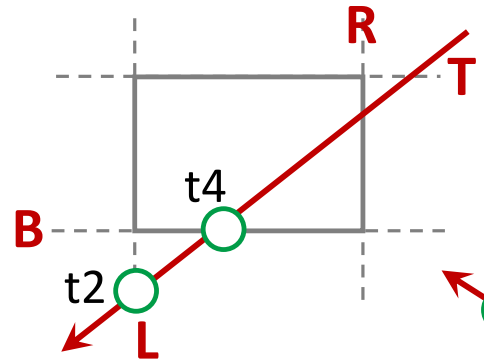
d2 > 0
d4 > 0



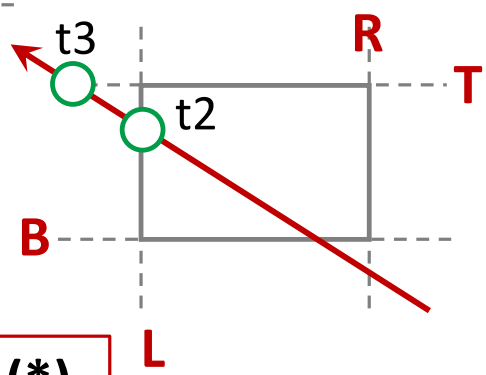
d2 > 0
d3 > 0



d1 > 0
d3 > 0



d1 > 0
d4 > 0



tb = min { ti = qi/di | di > 0 } (*)

(*) : Falta una condición

Clipping 2D: Liang-Barsky

- La condición faltante:
  - **$t \in [0, 1]$**

- Luego :
  - "ta" debe estar acotado por 0
  - i.e., el punto inicial del segmento está al interior de la ventana de clipping

  - "tb" debe estar acotado por 1
  - i.e., el punto final del segmento está al interior de la ventana de clipping

Clipping 2D: Liang-Barsky

- Pseudo-algoritmo

  - Si di < 0

    ta = max{ {qi/di | di < 0, i= 1,..,4}, 0}
  - Si di > 0

    tb = min{ {qi/di | di > 0, i= 1,...,4}, 1}
  - Luego, si ta > tb → se descarta el segmento
  - En caso contrario,

    el segmento visible está definido entre ta <= t <= tb

Clipping 2D: Liang-Barsky

- Pseudo-algoritmo

  - Si di < 0

    ta = max{ {qi/di | di < 0, i= 1,..,4}, 0}
  - Si di > 0

    tb = min{ {qi/di | di > 0, i= 1,...,4}, 1}
  - Luego, si ta > tb → se descarta el segmento
  - En caso contrario,

    el segmento visible está definido entre ta <= t <= tb

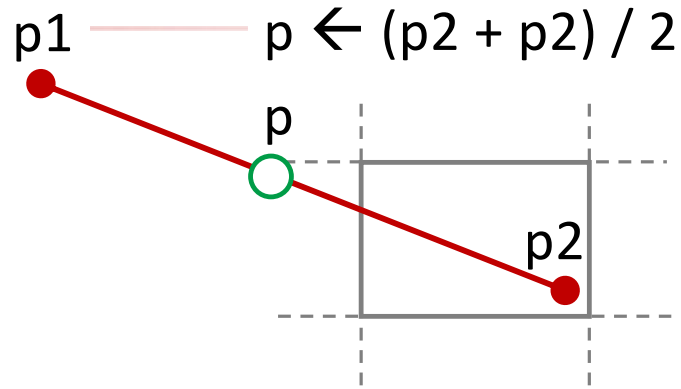
    Propuesto: programar el algoritmo 😈

Clipping: punto medio

- Un tercer algoritmo
- Punto medio

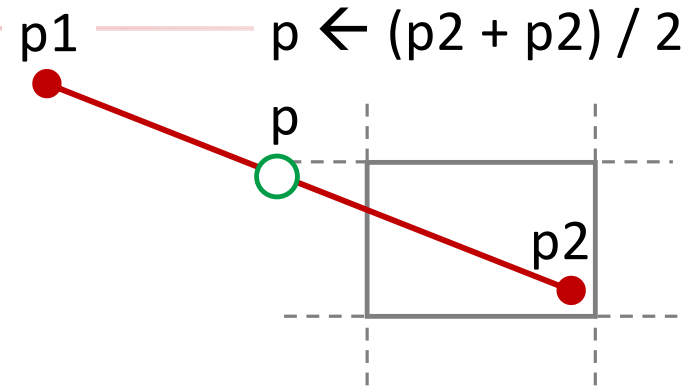
Clipping: punto medio

- Un segmento que no es completamente visible se divide en el punto medio
- Se descarta el lado no visible
- Si el otro lado aún no es completamente visible, se repite el proceso

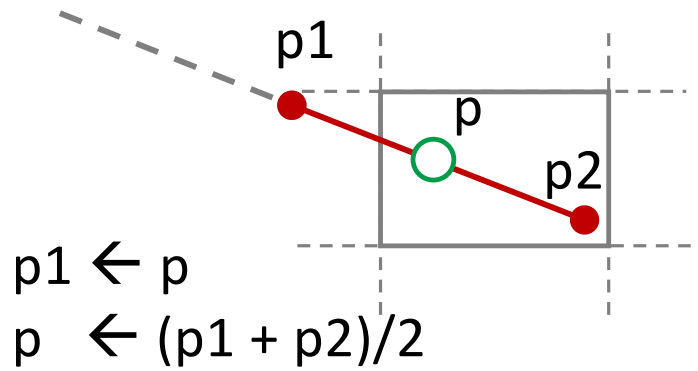


Clipping: punto medio

- Un segmento que no es completamente visible se divide en el punto medio
- Se descarta el lado no visible
- Si el otro lado aún no es completamente visible, se repite el proceso



- Propuesto:
  programar el algoritmo 😈



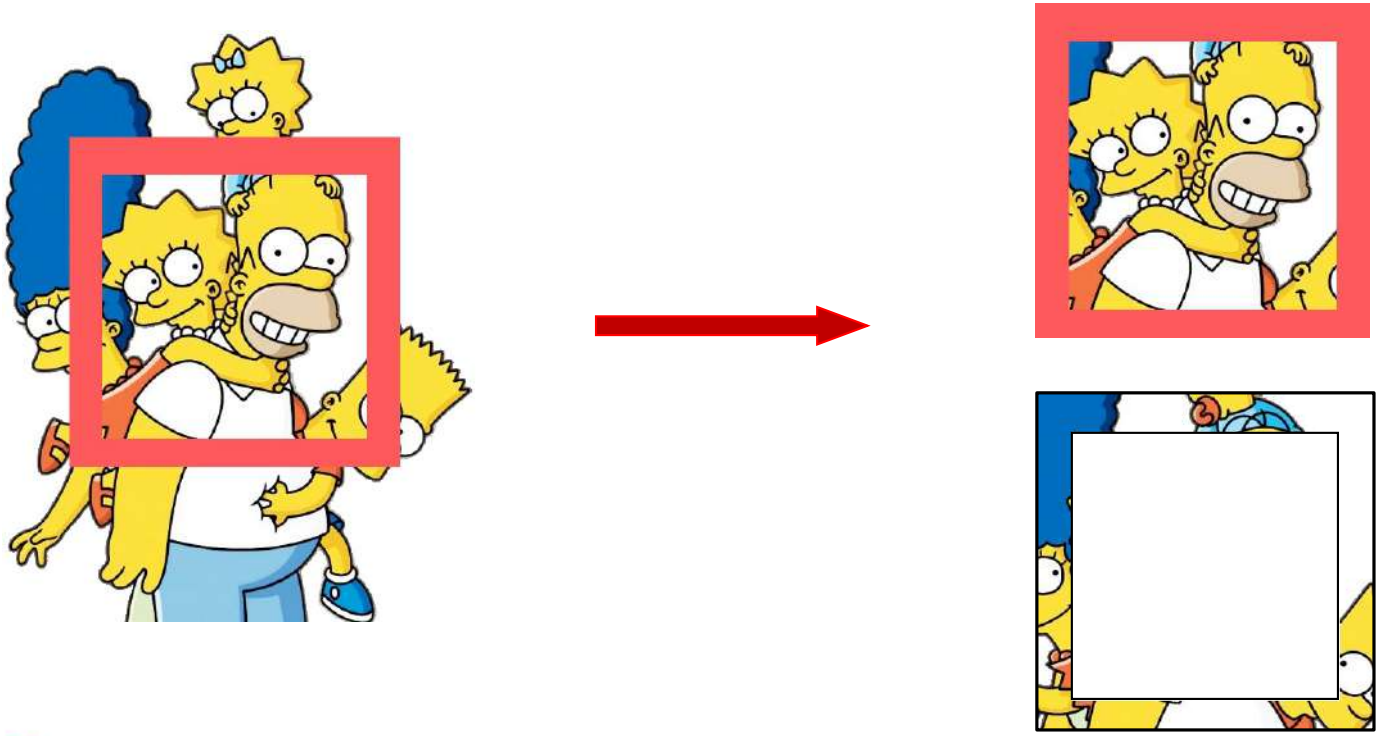
Clipping 2D

Patricio Inostroza, Dr.

dcc CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Clipping 2D

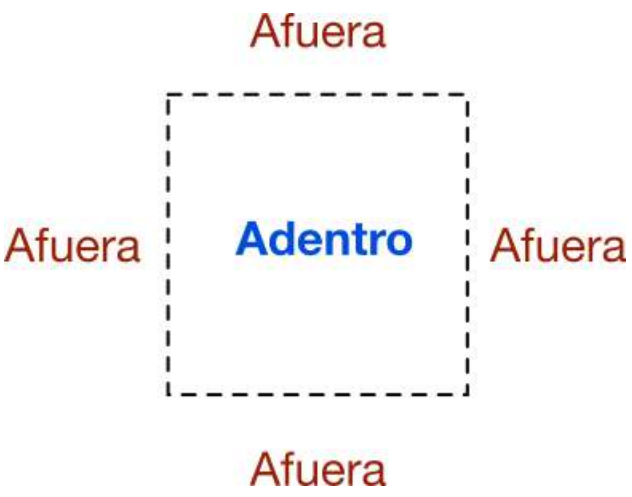
- Por lo general, sólo una parte de una figura es visualizada
- *Window*: zona seleccionada para proyección



Sutherland-Hodgman

- Cohen-sutherland, Liang-Barsky y Punto Medio
  - Detalle: Analizan cada segmento por separado
  - No aprovechan lo analizado en el paso previo

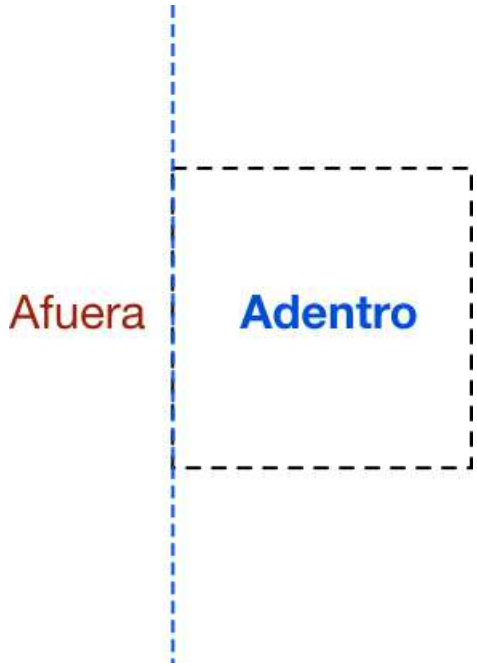
- Sutherland-Hodgman
  - Analiza el borde de la geometría
  - Recuerda "dónde está"
    - Adentro / Afuera
  - Analiza un lado a la vez



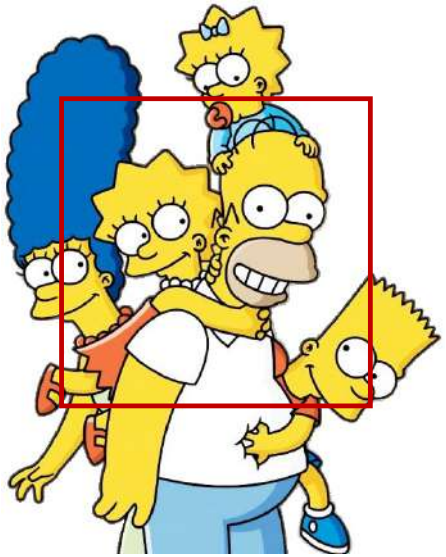
Sutherland-Hodgman

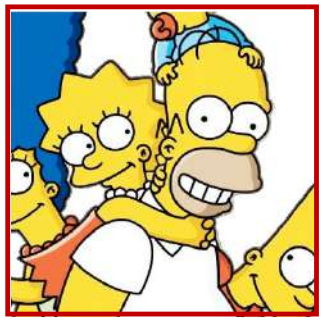
- Cohen-sutherland, Liang-Barsky y Punto Medio
  - Detalle: Analizan cada segmento por separado
  - No aprovechan lo analizado en el paso previo

- Sutherland-Hodgman
  - Analiza el borde de la geometría
  - Recuerda "dónde está"
    - Adentro / Afuera
  - Analiza un lado a la vez



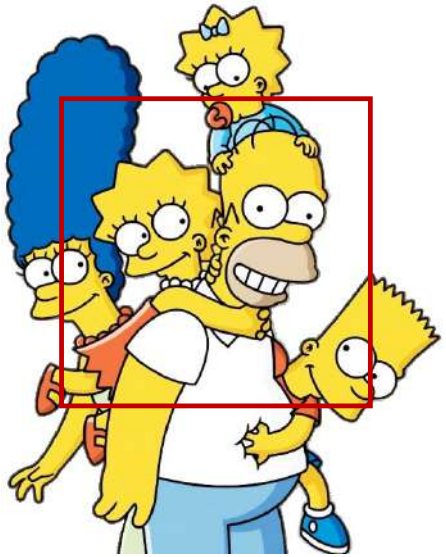
Sutherland-Hodgman

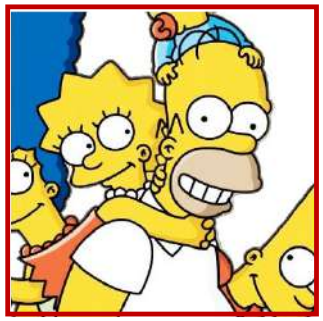




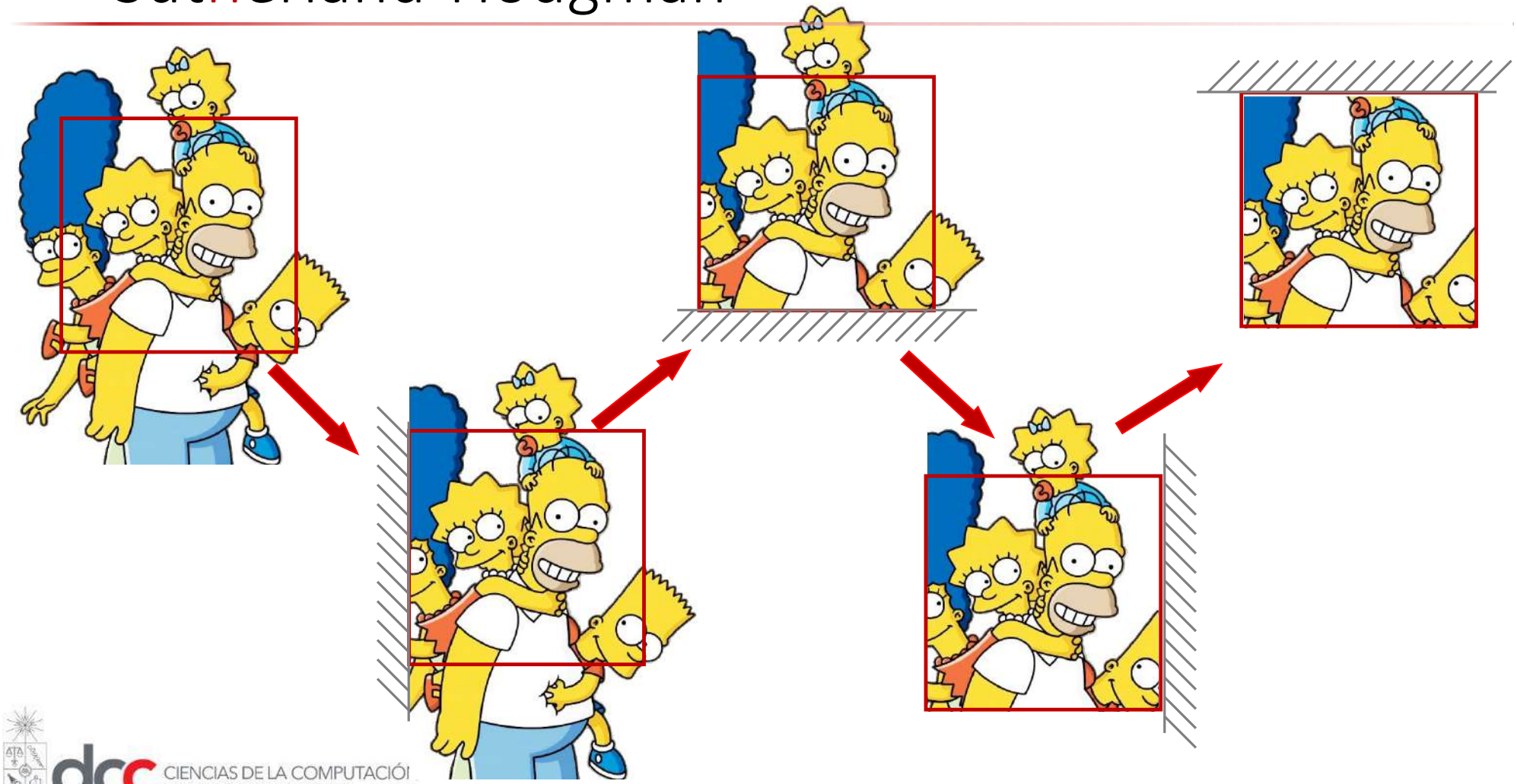
- Sutherland-Hodgman – 1974
  - Divide y conquistarás
- Simplificar el problema al recorta un polígono contra una sola arista
- Repetir el proceso para el resto de las aristas

Sutherland-Hodgman

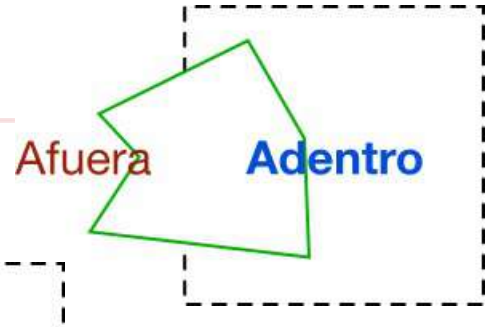


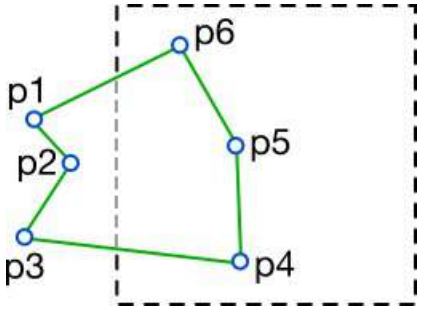


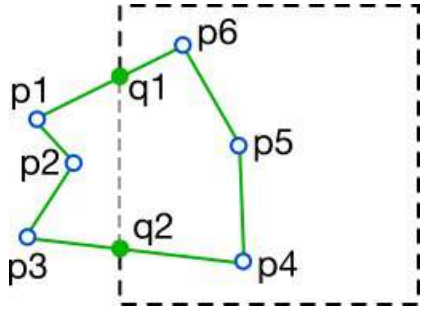
Sutherland-Hodgman

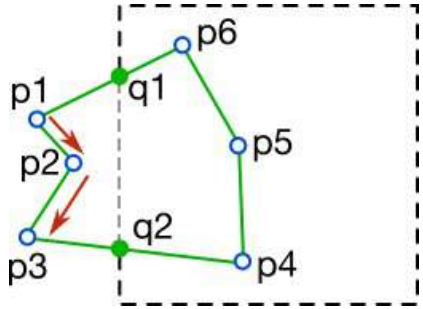


Sutherland-Hodgman







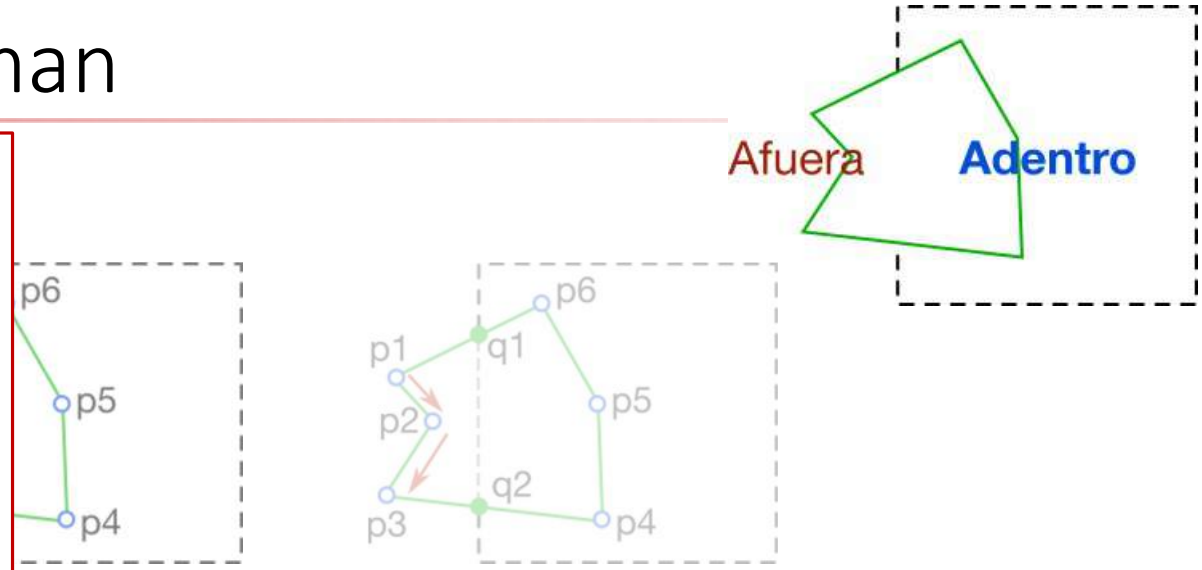


L = { }

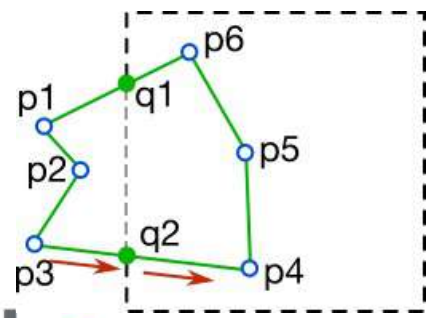
Sea **L** una lista vacía de puntos
Para cada arco formado por (pi, pj)
    Si pi y pj están afuera pase al siguiente arco
    Si pi y pj están adentro entonces
        L = L + pj
    Si pi está dentro y pj está afuera
        Calcule la intersección (pk)
        L = L + pk
    Si pi está fuera y pj está adentro
        Calcule la intersección (pk)
        L = L + pk
        L = L + pj

Sutherland-Hodgman

Sea **L** una lista vacía de puntos
Para cada arco formado por (pi, pj)
- Si pi y pj están afuera pase al siguiente arco
- Si pi y pj están adentro entonces
  - L = L + pj
- Si pi está dentro y pj está afuera
  - Calcule la intersección (pk)
  - L = L + pk
- Si pi está fuera y pj está adentro
  - Calcule la intersección (pk)
  - L = L + pk
  - L = L + pj

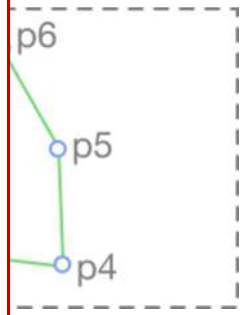


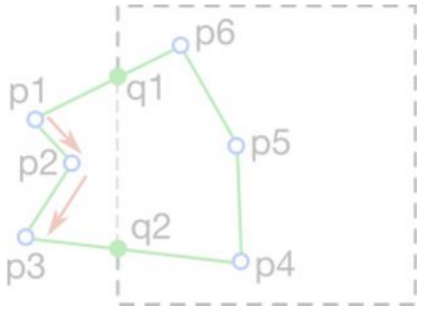
L = { q2, p4}



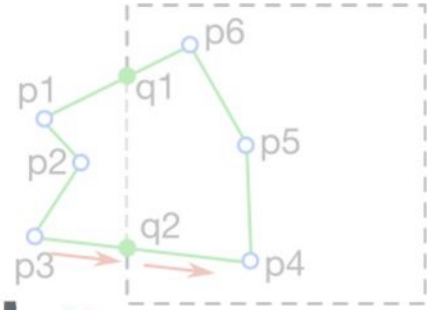
Sutherland-Hodgman

Sea **L** una lista vacía de puntos
Para cada arco formado por (pi, pj)
    Si pi y pj están afuera pase al siguiente arco
    Si pi y pj están adentro entonces
        L = L + pj
    Si pi está dentro y pj está afuera
        Calcule la intersección (pk)
        L = L + pk
    Si pi está fuera y pj está adentro
        Calcule la intersección (pk)
        L = L + pk
        L = L + pj

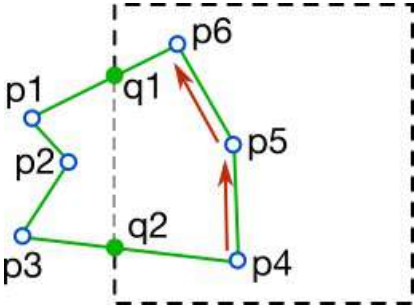




L = { q2, p4}

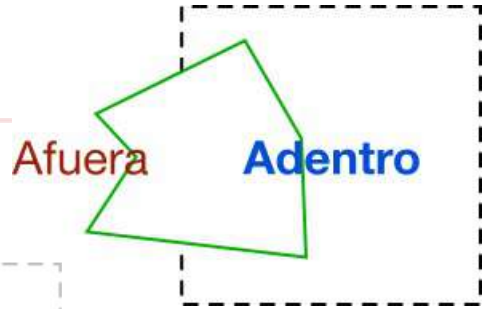


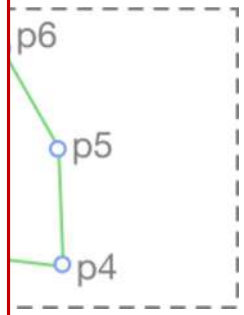
L = { q2, p4, p5, p6}

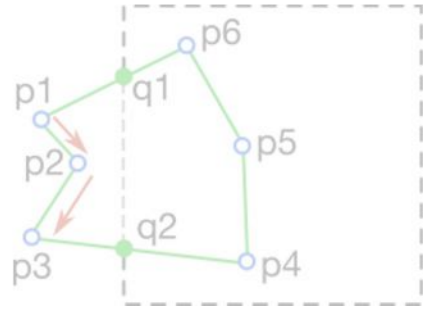


Sutherland-Hodgman

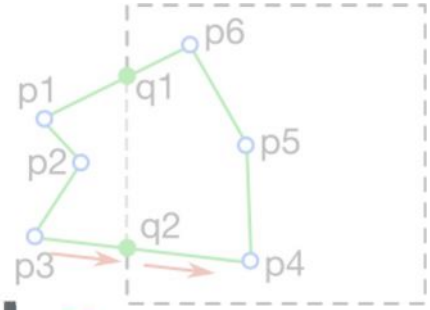
Sea **L** una lista vacía de puntos
Para cada arco formado por (pi, pj)
- Si pi y pj están afuera pase al siguiente arco
- Si pi y pj están adentro entonces
  - L = L + pj
- Si pi está dentro y pj está afuera
  - Calcule la intersección (pk)
  - L = L + pk
- Si pi está fuera y pj está adentro
  - Calcule la intersección (pk)
  - L = L + pk
  - L = L + pj



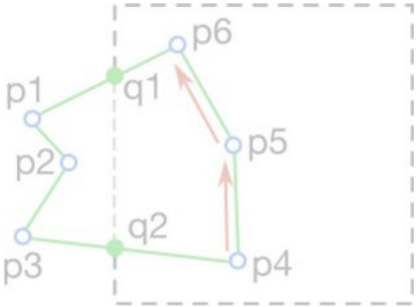




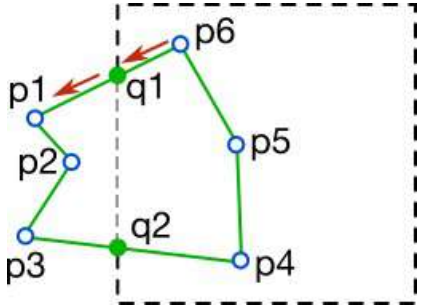
L = { q2, p4}



L = { q2, p4, p5, p6}



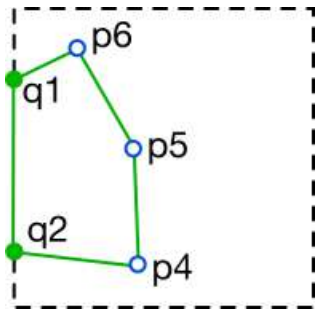
L = { q2, p4, p5, p6, q1}



Sutherland-Hodgman

Sea **L** una lista vacía de puntos
Para cada arco formado por (pi, pj)
    Si pi y pj están afuera pase al siguiente arco
    Si pi y pj están adentro entonces
        L = L + pj
    Si pi está dentro y pj está afuera
        Calcule la intersección (pk)
        L = L + pk
    Si pi está fuera y pj está adentro
        Calcule la intersección (pk)
        L = L + pk
        L = L + pj

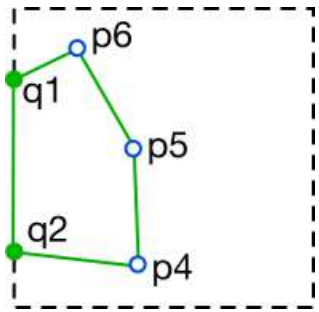
L = { q2, p4, p5, p6, q1}



Sutherland-Hodgman

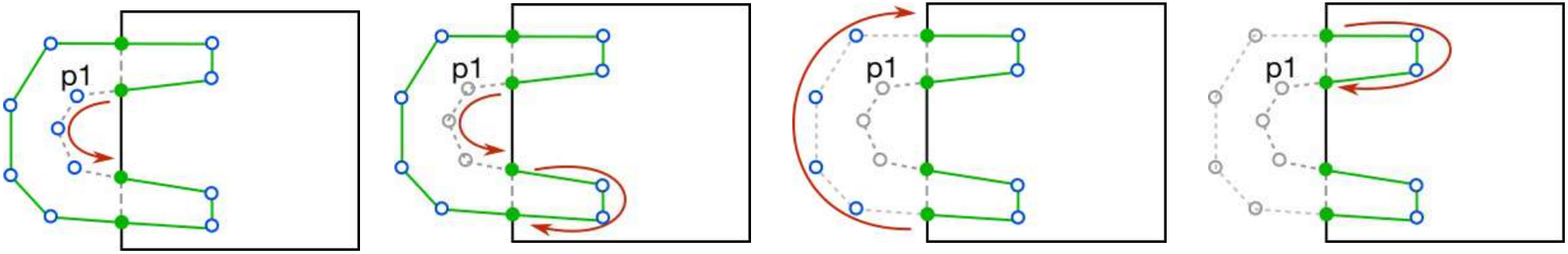
Sea **L** una lista vacía de puntos
Para cada arco formado por (pi, pj)
    Si pi y pj están afuera pase al siguiente arco
    Si pi y pj están adentro entonces
        L = L + pj
    Si pi está dentro y pj está afuera
        Calcule la intersección (pk)
        L = L + pk
    Si pi está fuera y pj está adentro
        Calcule la intersección (pk)
        L = L + pk
        L = L + pj

L = { q2, p4, p5, p6, q1}



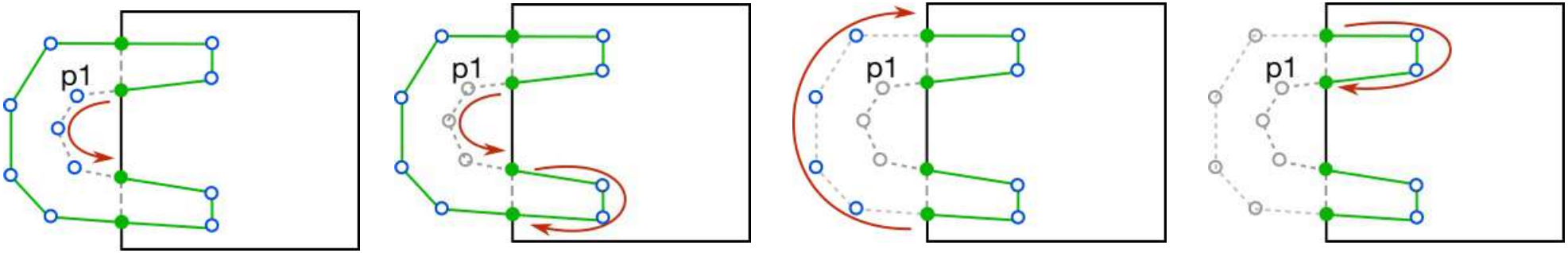
Parece que funciona 😀 … pero 🤔

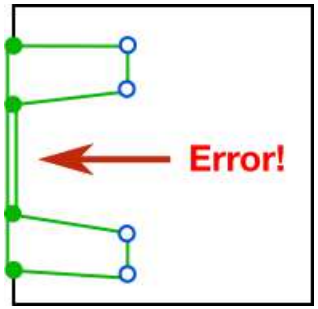
Sutherland-Hodgman



- Polígono cóncavo

Sutherland-Hodgman





- No funciona en polígonos cóncavos

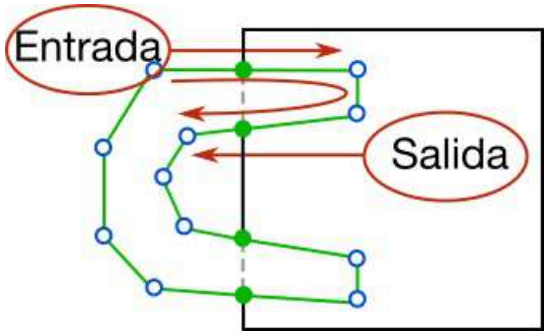
Qué hacer ?

- Separar el polígono cóncavo en polígonos convexos
  - Procesando cada uno por separado
- Verificar la lista de vértices creados
- Usar un algoritmo más general
  - Weiler-Atherton
  - Sirve para polígonos cóncavos
  - Sigue la frontera de la ventana

Weiler-Atherton

- La lista de vértices del polígono están ordenados en un sentido

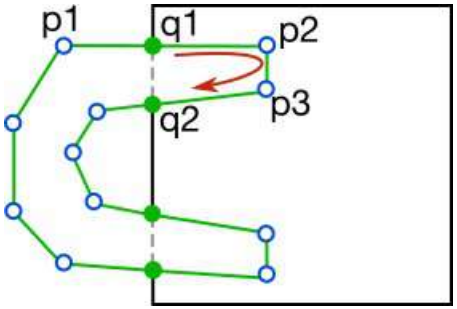
- Un punto de intersección, entre el arco y la ventana, se dirá de entrada si el arco del polígono esta ingresando a la ventana



- Un punto de intersección, entre el arco y la ventana, se dirá de salida si el arco del polígono esta saliendo de la ventana

Weiler-Atherton

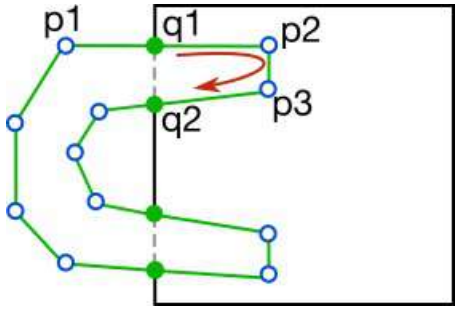
- Partir de un punto de intersección de un arco que va hacia adentro
- Recorrer el polígono
  - Incluir los puntos en la lista L
- Al llegar a un punto de intersección de un arco que va hacia afuera, seguir por el arco de la ventana
- Repetir hasta cerrar el polígono
- Repetir para el resto de los polígonos



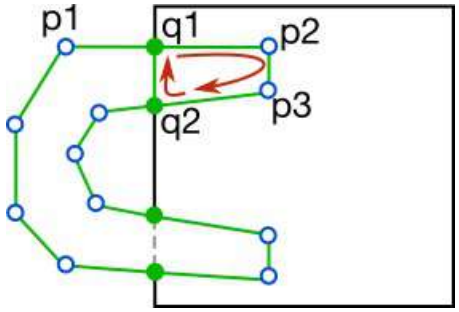
L = {{q1, p2, p3, q2}}

Weiler-Atherton

- Partir de un punto de intersección de un arco que va hacia adentro
- Recorrer el polígono
  - Incluir los puntos en la lista L
- Al llegar a un punto de intersección de un arco que va hacia afuera, seguir por el arco de la ventana
- Repetir hasta cerrar el polígono
- Repetir para el resto de los polígonos

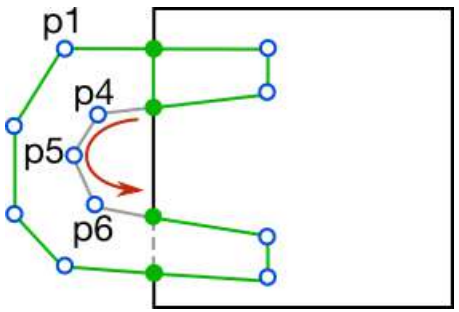


L = {{q1, p2, p3, q2}}



Weiler-Atherton

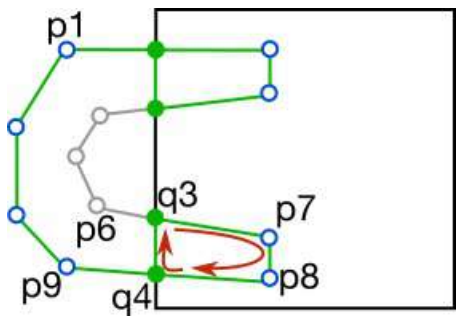
- Partir de un punto de intersección de un arco que va hacia adentro
- Recorrer el polígono
  - Incluir los puntos en la lista L
- Al llegar a un punto de intersección de un arco que va hacia afuera, seguir por el arco de la ventana
- Repetir hasta cerrar el polígono
- Repetir para el resto de los polígonos



L = {{q1, p2, p3, q2}}

Weiler-Atherton

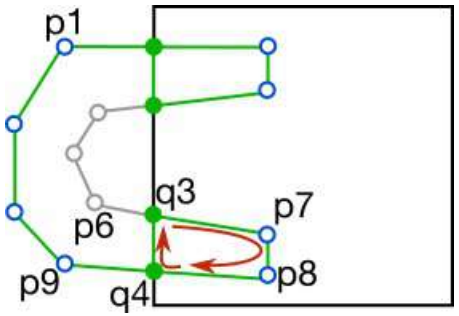
- Partir de un punto de intersección de un arco que va hacia adentro
- Recorrer el polígono
  - Incluir los puntos en la lista L
- Al llegar a un punto de intersección de un arco que va hacia afuera, seguir por el arco de la ventana
- Repetir hasta cerrar el polígono
- Repetir para el resto de los polígonos



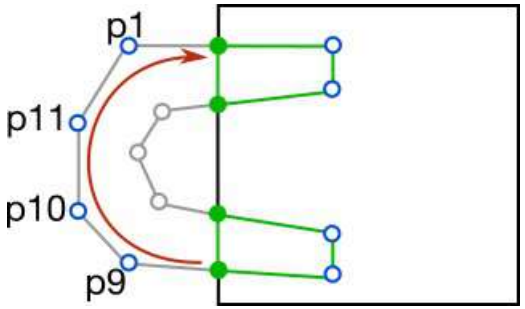
L = {{q1, p2, p3, q2}, {q3, p7, p8, q4}}

Weiler-Atherton

- Partir de un punto de intersección de un arco que va hacia adentro
- Recorrer el polígono
  - Incluir los puntos en la lista L
- Al llegar a un punto de intersección de un arco que va hacia afuera, seguir por el arco de la ventana
- Repetir hasta cerrar el polígono
- Repetir para el resto de los polígonos

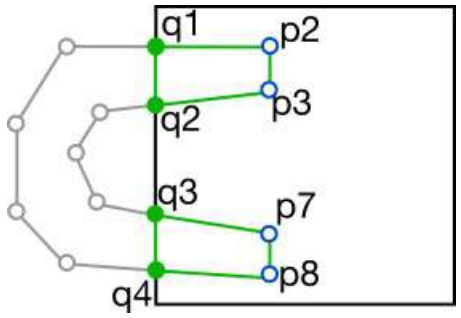


L = {{q1, p2, p3, q2}, {q3, p7, p8, q4}}



Weiler-Atherton

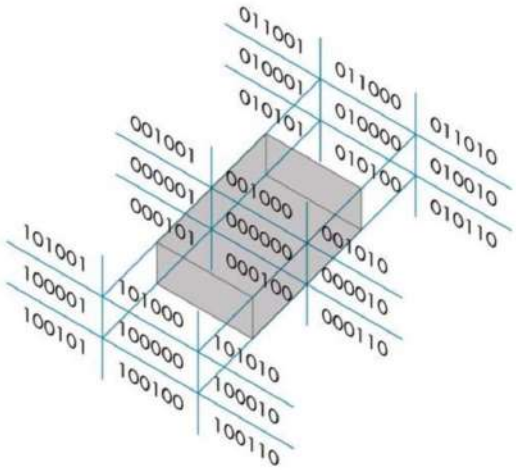
- Partir de un punto de intersección de un arco que va hacia adentro
- Recorrer el polígono
  - Incluir los puntos en la lista L
- Al llegar a un punto de intersección de un arco que va hacia afuera, seguir por el arco de la ventana
- Repetir hasta cerrar el polígono
- Repetir para el resto de los polígonos



L = {{q1, p2, p3, q2}, {q3, p7, p8, q4}}

Propuestos

- Investigue al menos el funcionamiento de otro algoritmo de clipping
- Generalice el algoritmo de Weiler-Atherton a 3D
- Generalice el algoritmo de Sutherland- Hodgman a 3D
- Generalice Cohen-Sutherland a 3D



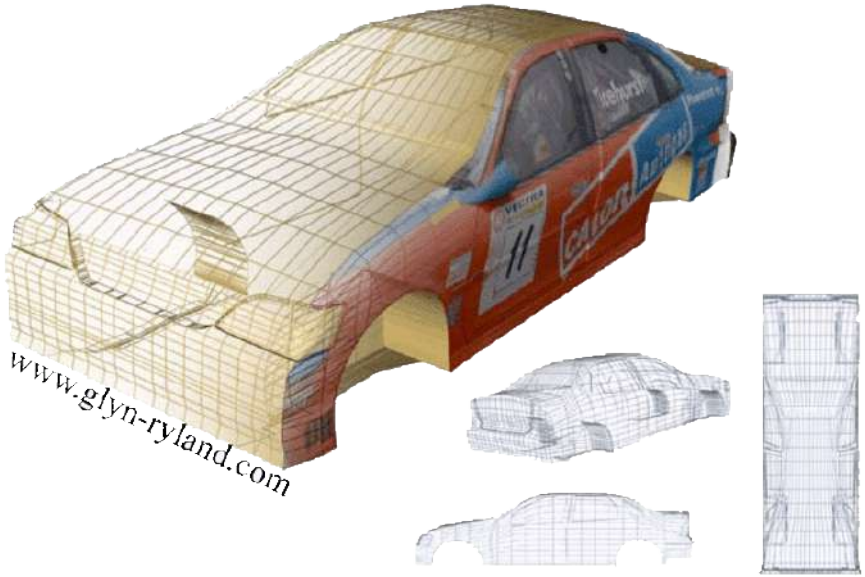
Transformaciones 3D

Patricio Inostroza, Dr.

dcc CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Geometría Tridimensional

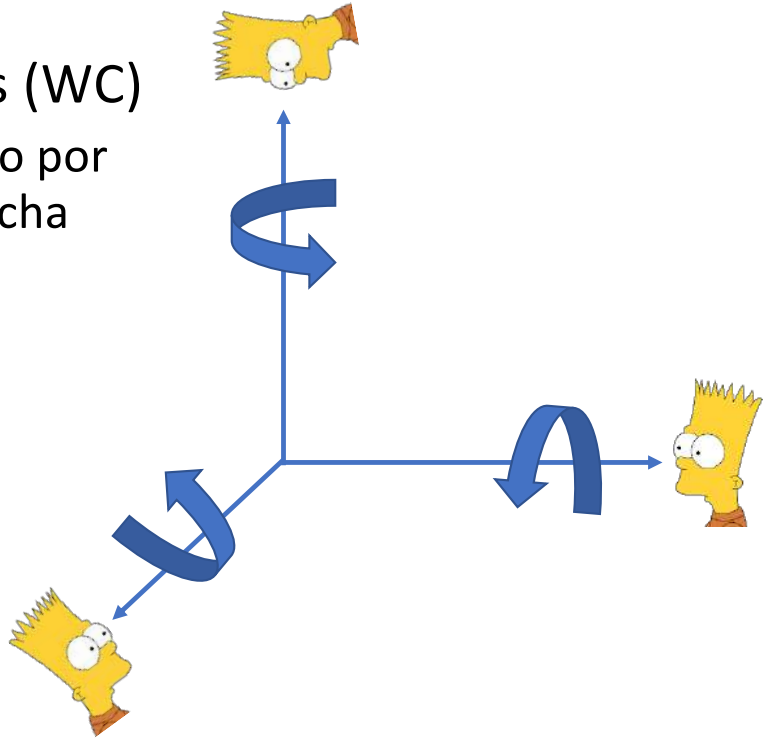
- Gran parte de los problemas de ingeniería son 3D



- Independiente de su representación 3D, hay que realizar un 'mapeo' a 2D

Geometría Tridimensional

- Sistema de coordenadas (WC)
  - El sentido queda definido por la regla de la mano derecha

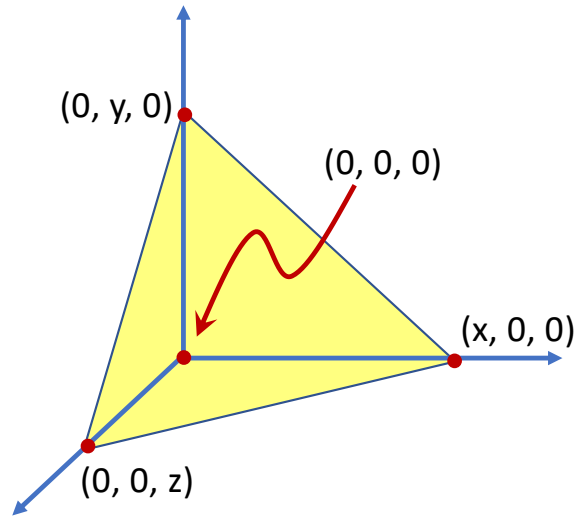


Geometría Tridimensional

- Representación de un objeto 3D
  - En coordenadas Homogéneas

$$P = \begin{pmatrix} 0 & x & 0 & 0 \\ 0 & 0 & y & 0 \\ 0 & 0 & 0 & z \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Tetraedro



Geometría Tridimensional

- Las transformaciones 3D son equivalentes a las transformaciones 2D
- Matriz de transformación en coordenadas homogéneas

Escalamiento
Rotación
Shearing
Reflexión

$$\begin{pmatrix} A & B & C & Tx \\ D & E & F & Ty \\ G & H & I & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

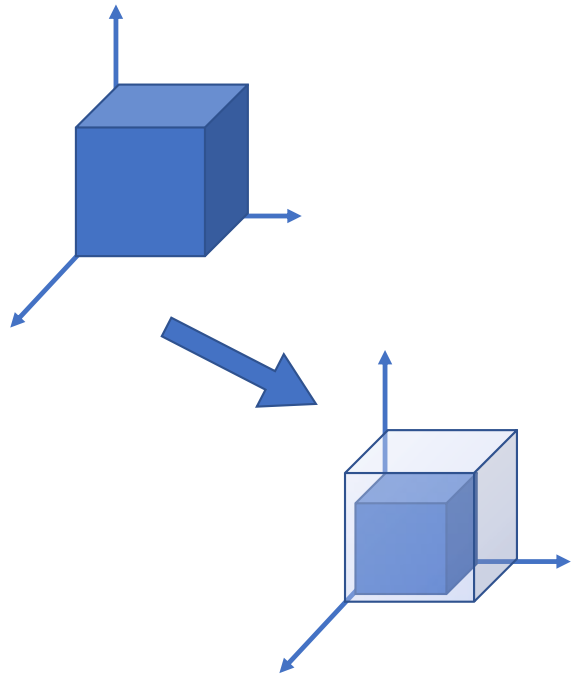
Transformaciones
De Traslación

Transformaciones
en perspectiva

Geometría 3D: Escalamiento

- Matriz homogénea de escalamiento

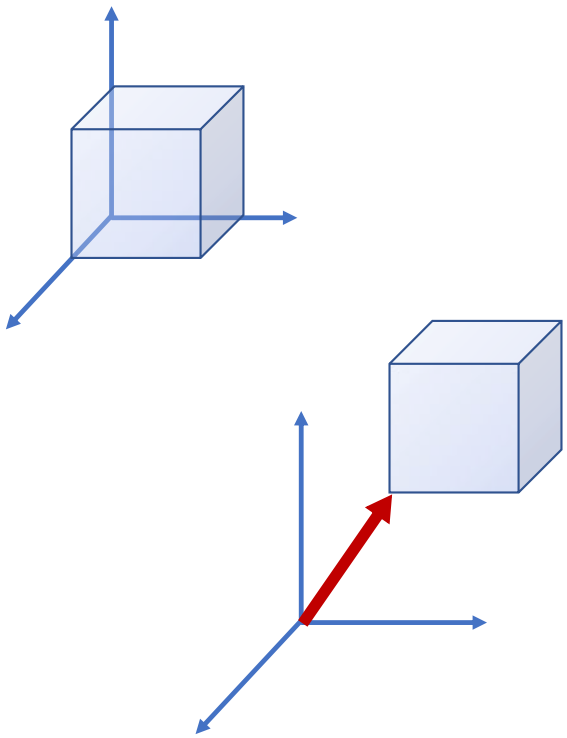
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



Geometría 3D: Traslación

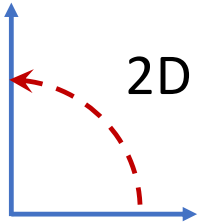
- • Matriz homogénea de traslación:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

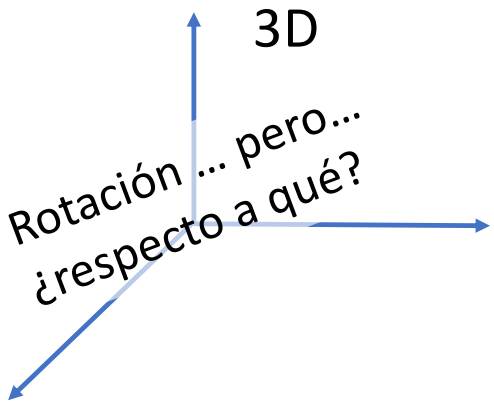


Geometría 3D: Rotación

- En 2D la rotación es "natural"
  - Es acotada



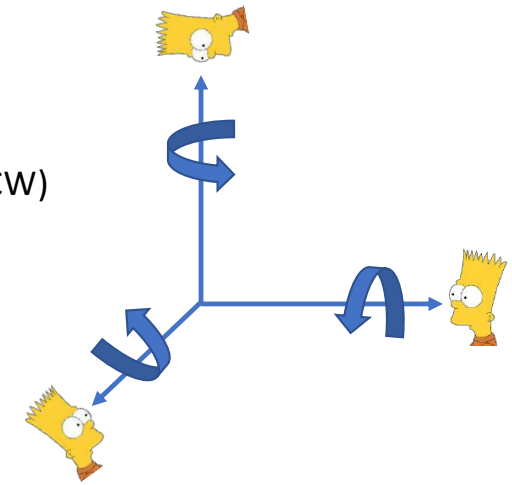
- En 3D hay más libertad



Geometría 3D: Rotación

- De mayor complejidad que en 2D
  - Se debe especificar uno de los ejes de rotación: X, Y ó Z.

Rotaciones en
sentido horario (CW)

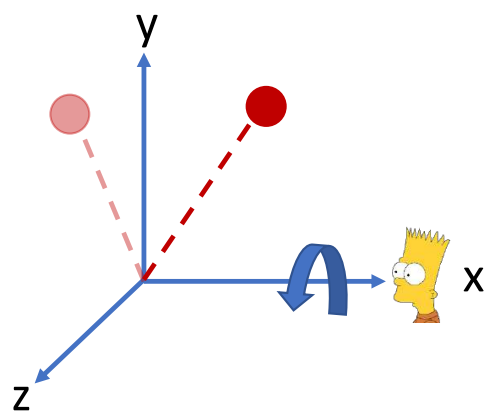


- Rotación c/r a un eje arbitrario es una descomposición de rotaciones simples c/r a los tres ejes principales

Geometría 3D: Rotación

- Rotación alrededor del eje x

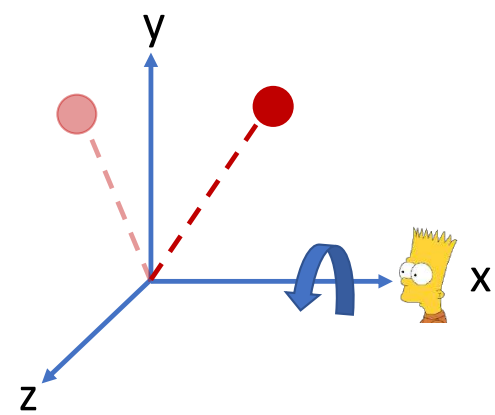
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

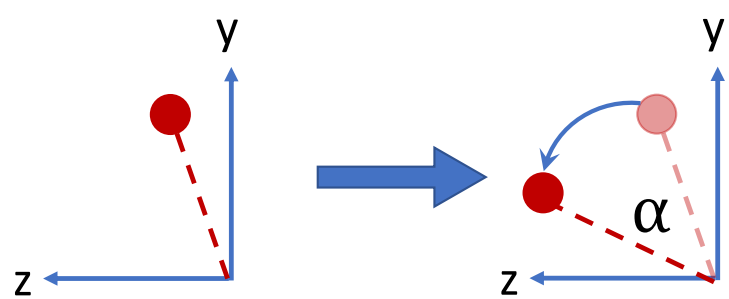


Geometría 3D: Rotación

- Rotación alrededor del eje x

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

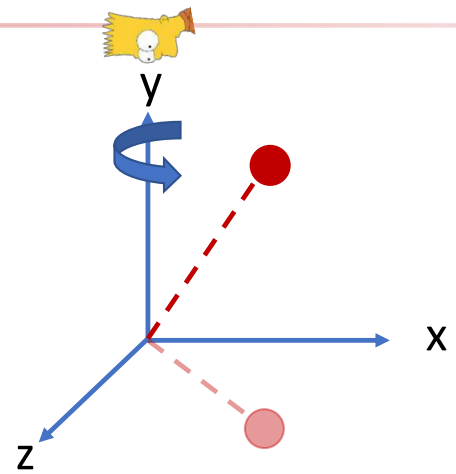




Geometría 3D: Rotación

- Rotación alrededor del eje y

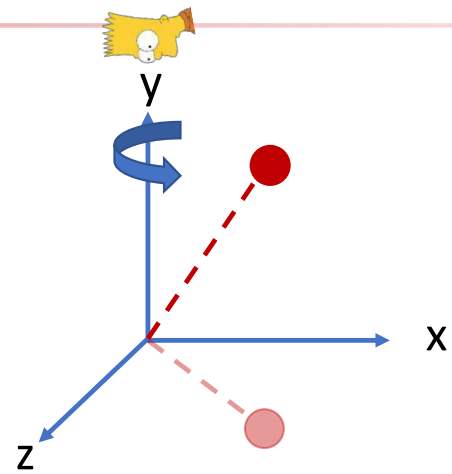
$$\begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

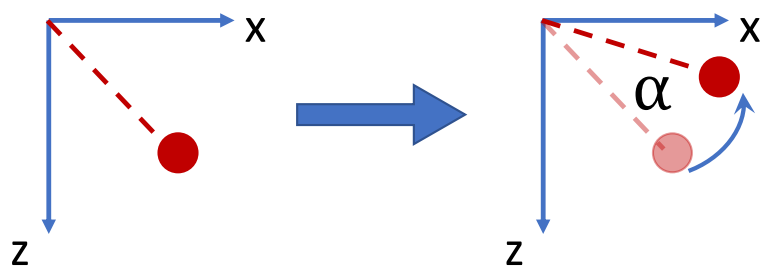


Geometría 3D: Rotación

- Rotación alrededor del eje y

$$\begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

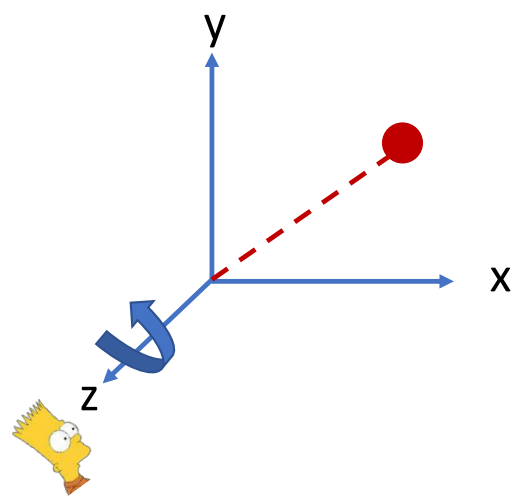


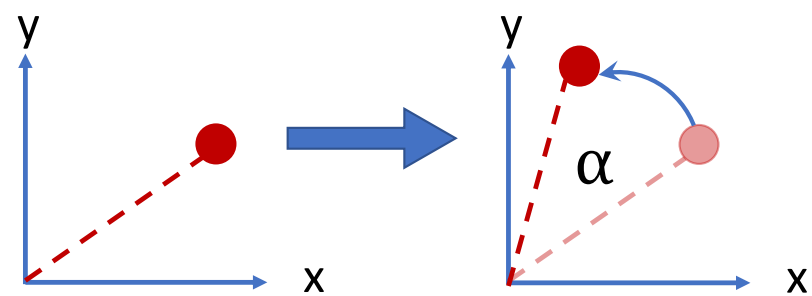


Geometría 3D: Rotación

- Rotación alrededor del eje y

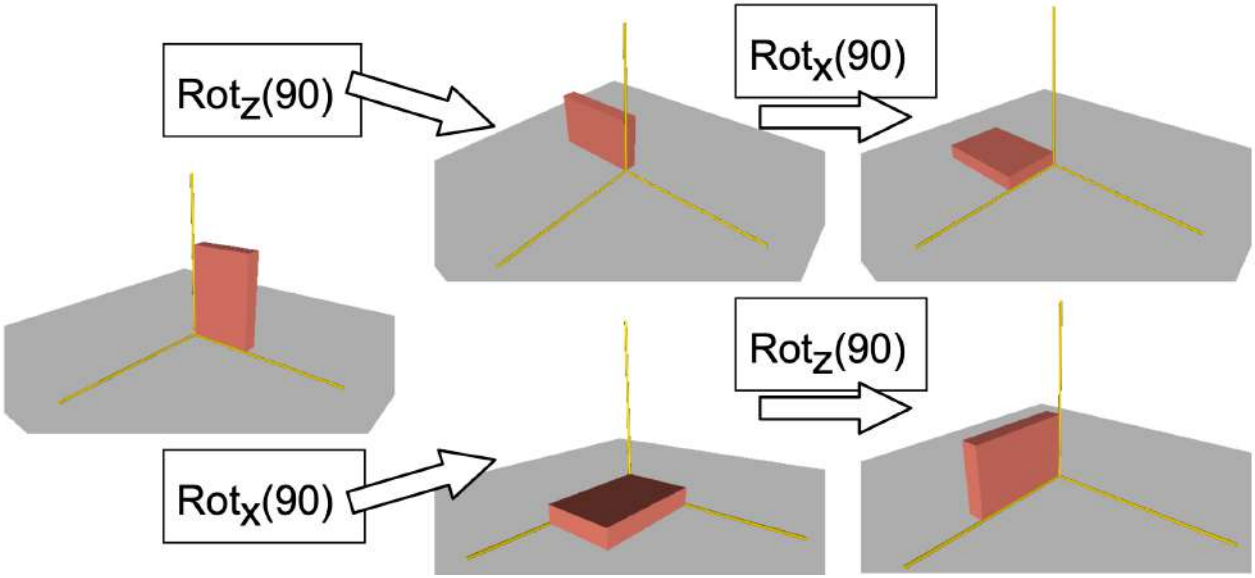
$$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$





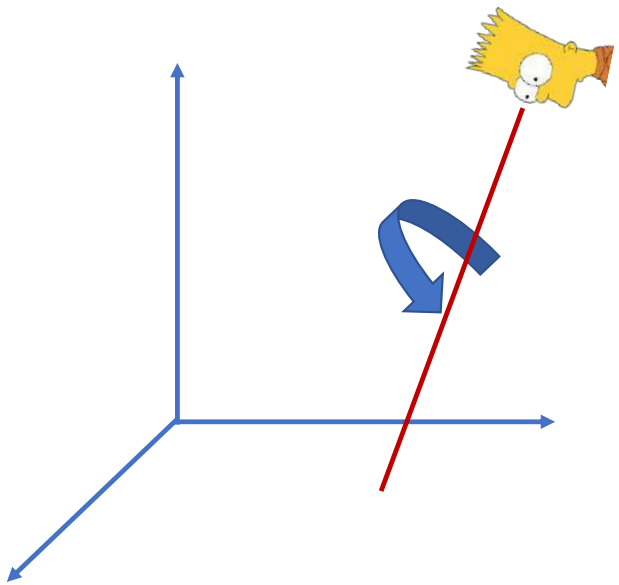
Geometría 3D: Rotación

- Las rotaciones no son conmutativas
  - El orden en que se aplican afecta el resultado
- Ejemplo:
  - Rotz(90) → Rotx(90) v/s Rotx(90) → Rotz(90)



Geometría 3D: Rotación

- Rotación alrededor de un eje arbitrario



Geometría 3D: Rotación

- Rotación alrededor de un eje arbitrario
  1. Trasladar el eje arbitrario al origen
  2. Rotar el eje alrededor de los ejes X e Y hasta alinear el eje arbitrario con Z positivo
  3. Rotar alrededor del eje Z en el ángulo deseado (Ω)
  4. Aplicar las rotaciones inversas alrededor de los ejes X e Y
  5. Aplicar la traslación inversa

M = T(-dx, -dy, -dz)

Geometría 3D: Rotación

- Rotación alrededor de un eje arbitrario
  1. Trasladar el eje arbitrario al origen
  2. Rotar el eje alrededor de los ejes X e Y hasta alinear el eje arbitrario con Z positivo
  3. Rotar alrededor del eje Z en el ángulo deseado (Ω)
  4. Aplicar las rotaciones inversas alrededor de los ejes X e Y
  5. Aplicar la traslación inversa

M = T(-dx, -dy, -dz)

Geometría 3D: Rotación

- Rotación alrededor de un eje arbitrario
  1. Trasladar el eje arbitrario al origen
  2. Rotar el eje alrededor de los ejes X e Y hasta alinear el eje arbitrario con Z positivo
  3. Rotar alrededor del eje Z en el ángulo deseado (Ω)
  4. Aplicar las rotaciones inversas alrededor de los ejes X e Y
  5. Aplicar la traslación inversa

M = T(-dx, -dy, -dz)

M = Rotx(α) * T(-dx, -dy, -dz)

Geometría 3D: Rotación

- Rotación alrededor de un eje arbitrario
  1. Trasladar el eje arbitrario al origen
  2. Rotar el eje alrededor de los ejes X e Y hasta alinear el eje arbitrario con Z positivo
  3. Rotar alrededor del eje Z en el ángulo deseado (Ω)
  4. Aplicar las rotaciones inversas alrededor de los ejes X e Y
  5. Aplicar la traslación inversa

M = T(-dx, -dy, -dz)

M = Rotx(α) * T(-dx, -dy, -dz)

M = Roty(ß) * Rotx(α) * T(-dx, -dy, -dz)

Geometría 3D: Rotación

- Rotación alrededor de un eje arbitrario
  1. Trasladar el eje arbitrario al origen
  2. Rotar el eje alrededor de los ejes X e Y hasta alinear el eje arbitrario con Z positivo
  3. Rotar alrededor del eje Z en el ángulo deseado (Ω)
  4. Aplicar las rotaciones inversas alrededor de los ejes X e Y
  5. Aplicar la traslación inversa

M = T(-dx, -dy, -dz)

M = Rotx(α) * T(-dx, -dy, -dz)

M = Roty(ß) * Rotx(α) * T(-dx, -dy, -dz)

M = Rotz(Ω) * Roty(ß) * Rotx(α) * T(-dx, -dy, -dz)

Geometría 3D: Rotación

- Rotación alrededor de un eje arbitrario
  1. Trasladar el eje arbitrario al origen
  2. Rotar el eje alrededor de los ejes X e Y hasta alinear el eje arbitrario con Z positivo
  3. Rotar alrededor del eje Z en el ángulo deseado (Ω)
  4. Aplicar las rotaciones inversas alrededor de los ejes X e Y
  5. Aplicar la traslación inversa

M = T(-dx, -dy, -dz)

M = Rotx(α) * T(-dx, -dy, -dz)

M = Roty(ß) * Rotx(α) * T(-dx, -dy, -dz)

M = Rotz(Ω) * Roty(ß) * Rotx(α) * T(-dx, -dy, -dz)

M = T(dx, dy, dz) * Rotx(–α) * Roty(–ß) * Rotz(Ω) * Roty(ß) * Rotx(α) * T(-dx, -dy, -dz)

Geometría 3D: Reflexión

- El efecto espejo es extensible a 3D
- Plano x = 0

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Plano y = 0

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Plano z = 0

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- C/r al origen →

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Geometría 3D: Shearing

- Shearing: Distorsión de una o más coordenadas debido a la acción de las otras coordenadas.

$$Sh = \begin{pmatrix} 1 & Sxy & Sxz & 0 \\ Syx & 1 & Syz & 0 \\ Szx & Szy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Sxy: cantidad de 'shear' debido a y a lo largo de x
- Syx: cantidad de 'shear' debido a x a lo largo de y

Geometría 3D: Shearing

- Ejemplo:
  - Distorsión de un punto P(x, y, z) por la siguiente matriz

$$P' = \begin{pmatrix} 1 & Sxy & Sxz & 0 \\ 0 & 1 & 0 & 0 \\ 0 & Szy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

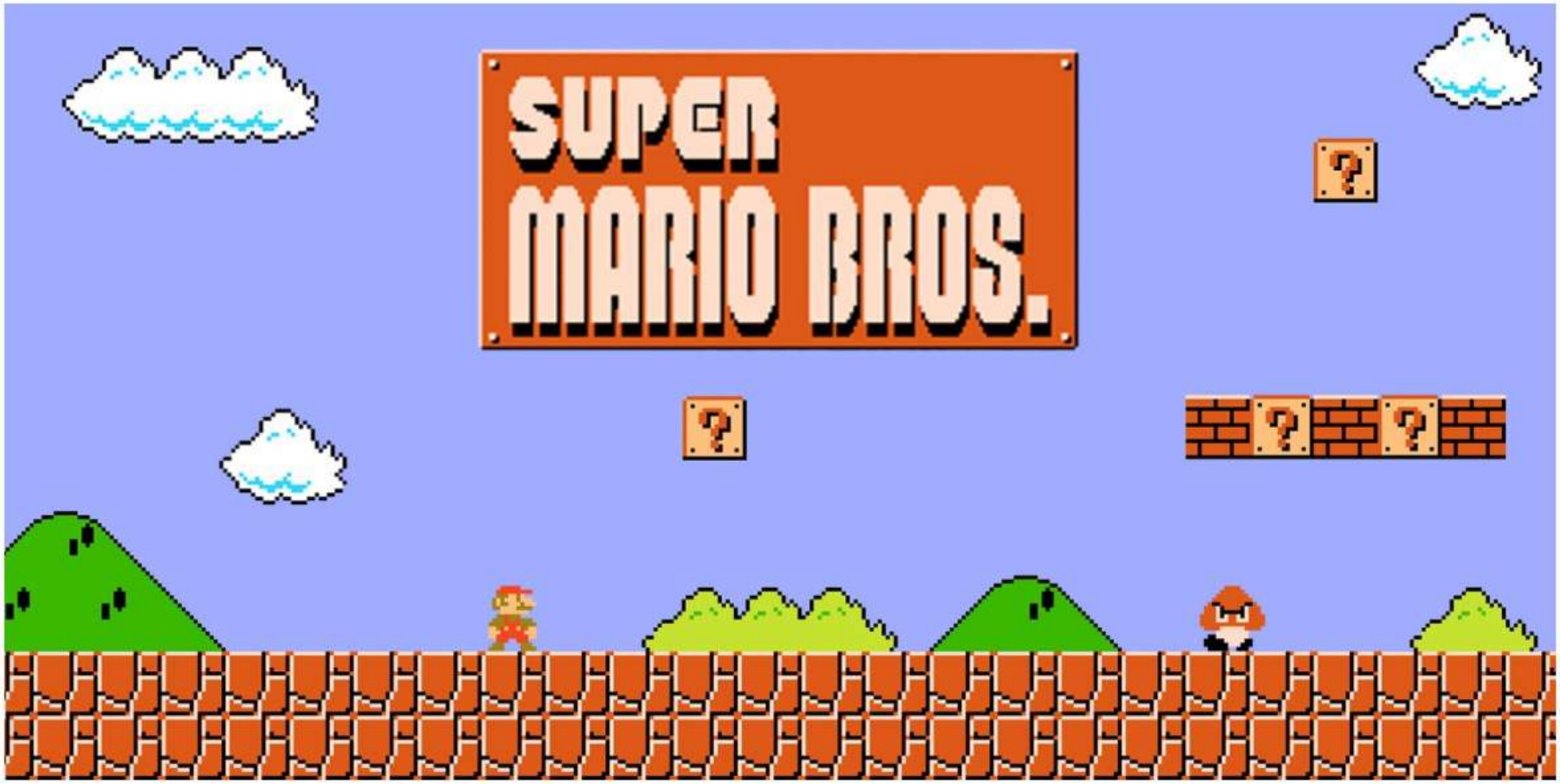
$$P' = \begin{pmatrix} x + y*Sxy + z*Sxz \\ y \\ y*Szy + z \\ 1 \end{pmatrix}$$

Modelación Jerárquica

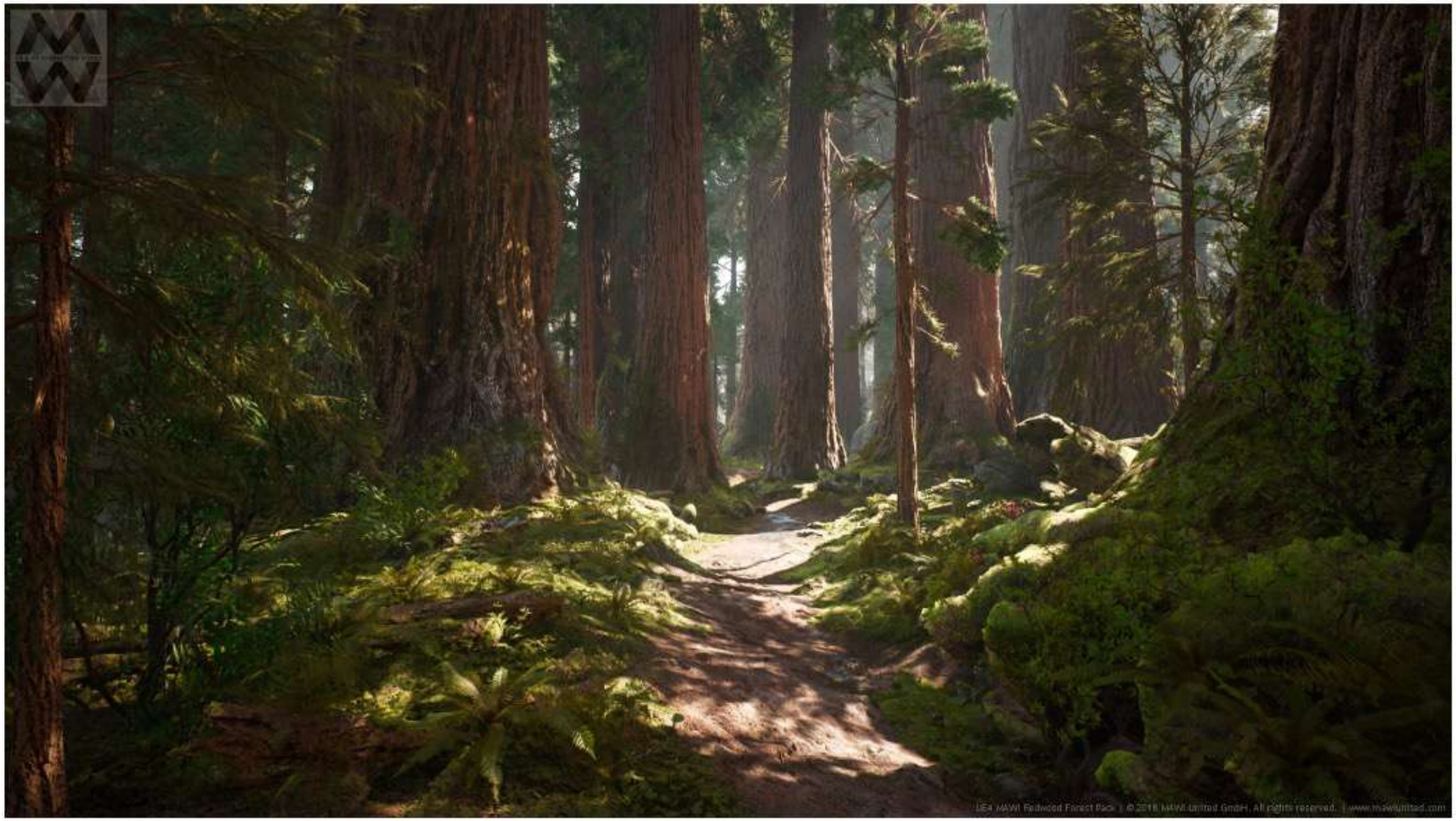
Dr. Ivan Sipiran

Cómo dibujar una escena compleja?

Cómo dibujar una escena compleja?



Cómo dibujar una escena compleja?



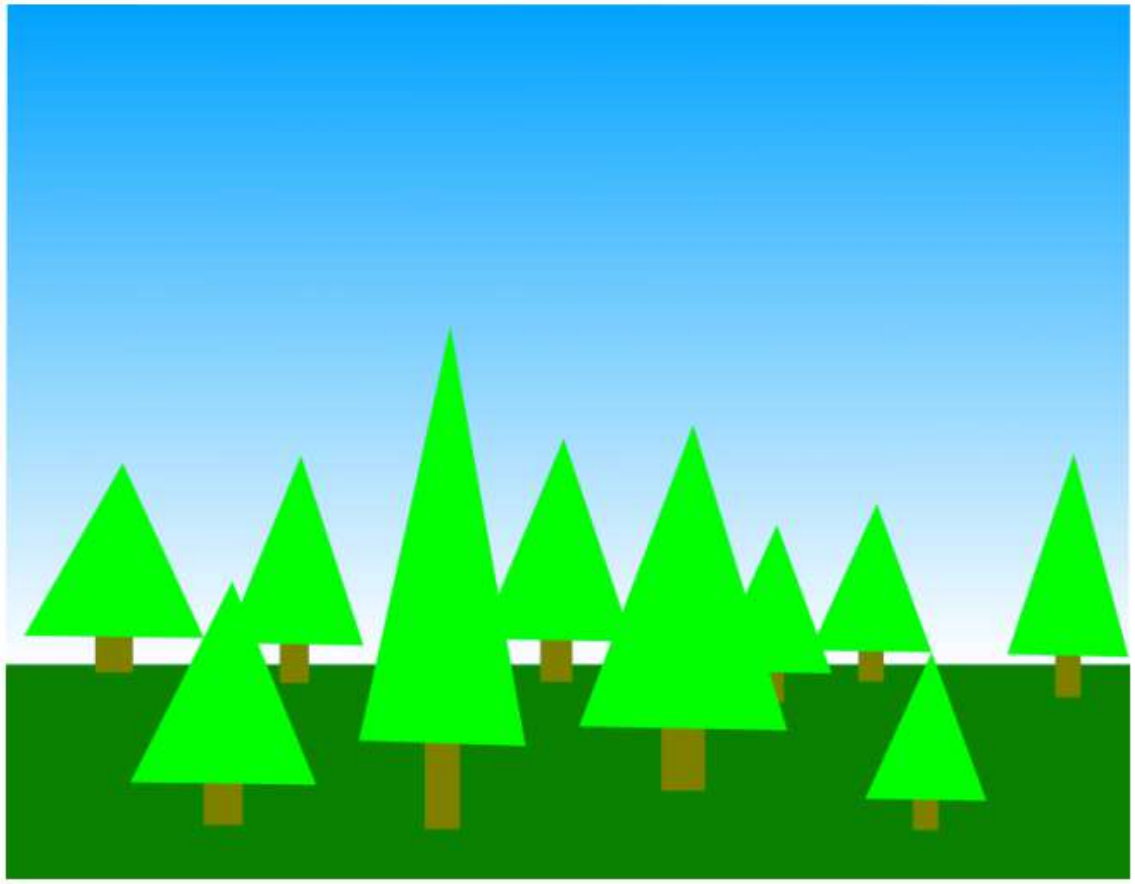
Cómo dibujar una escena compleja?

- Características
  - Objetos repetidos
  - Objetos contenidos en otros objetos

Grafo de Escena

Cómo dibujar una escena compleja?

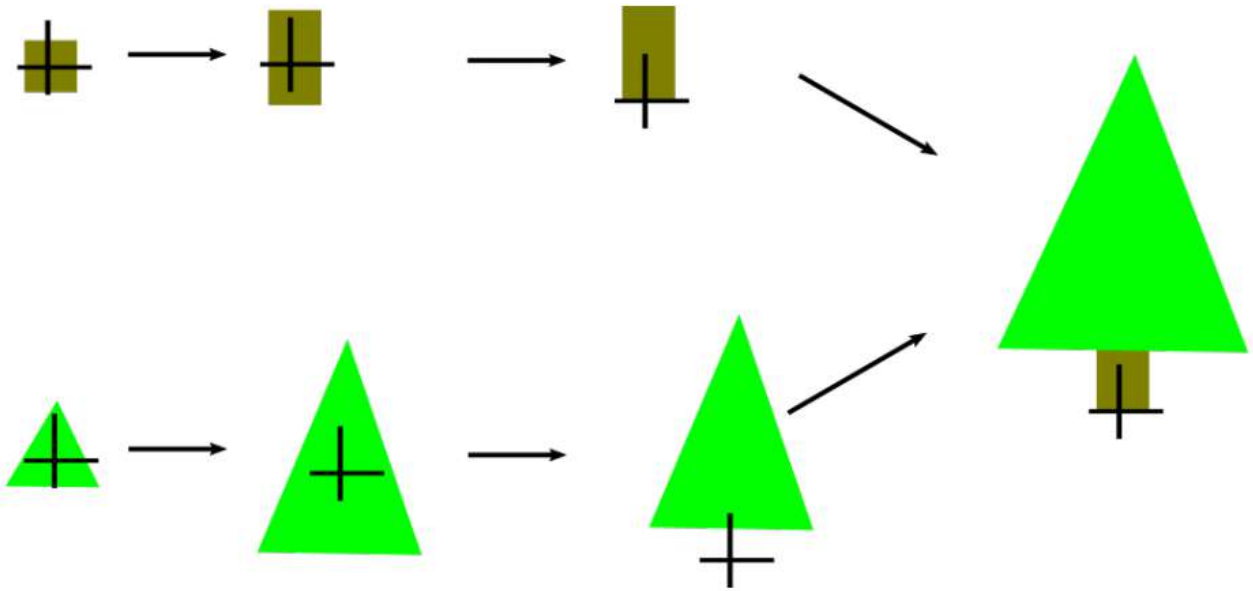
- Utilizando distintas transformaciones podemos dibujar el mismo objeto en distintos lugares de la escena



Cómo dibujar una escena compleja?

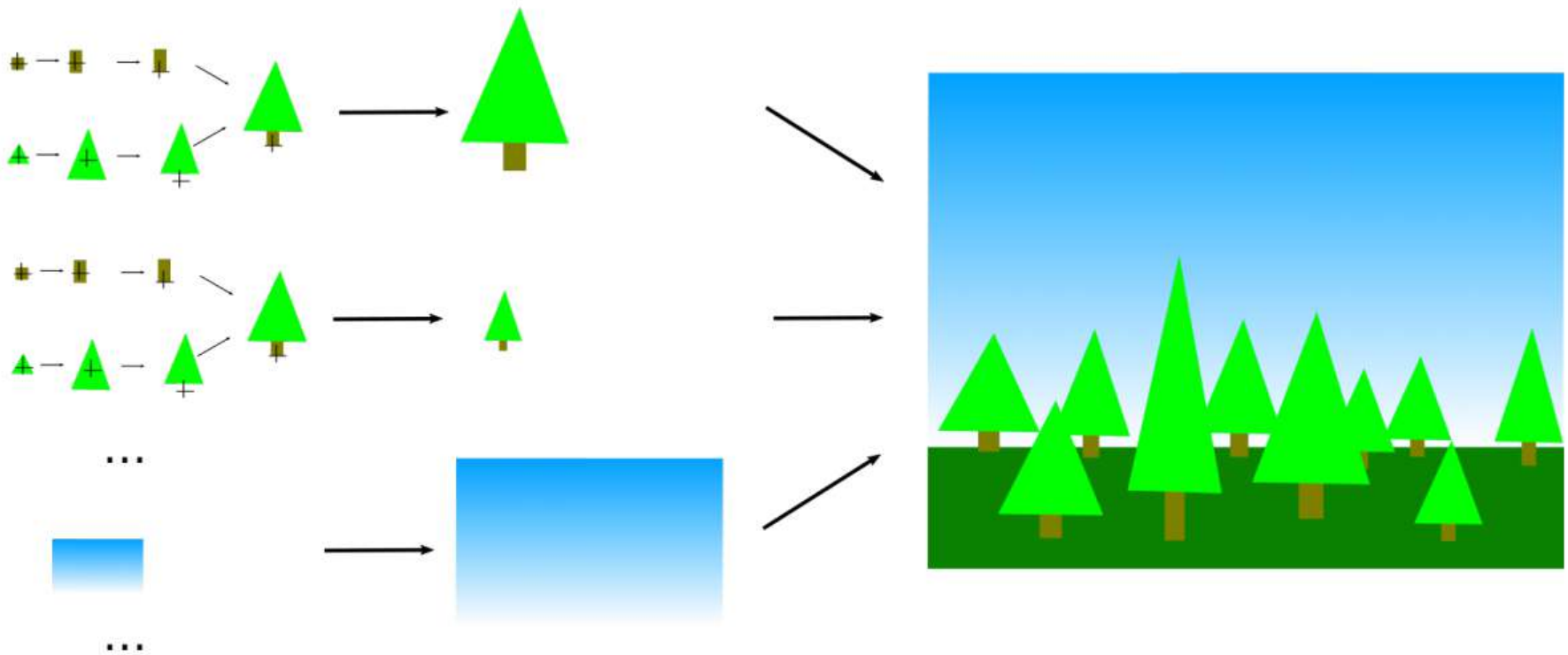
- El objeto básico es un pino compuesto de otras dos figuras básicas.
- Supongamos que tenemos una función que dibuja un cuadrado café y otra que dibuje un triángulo verde.
- Ambas figuras se dibujan centradas.
- Utilizando transformaciones podemos armar el pino.
- Se tiene una jerarquía de transformaciones.
- Esta jerarquía se puede representar como un grafo.

Grafo de Escena



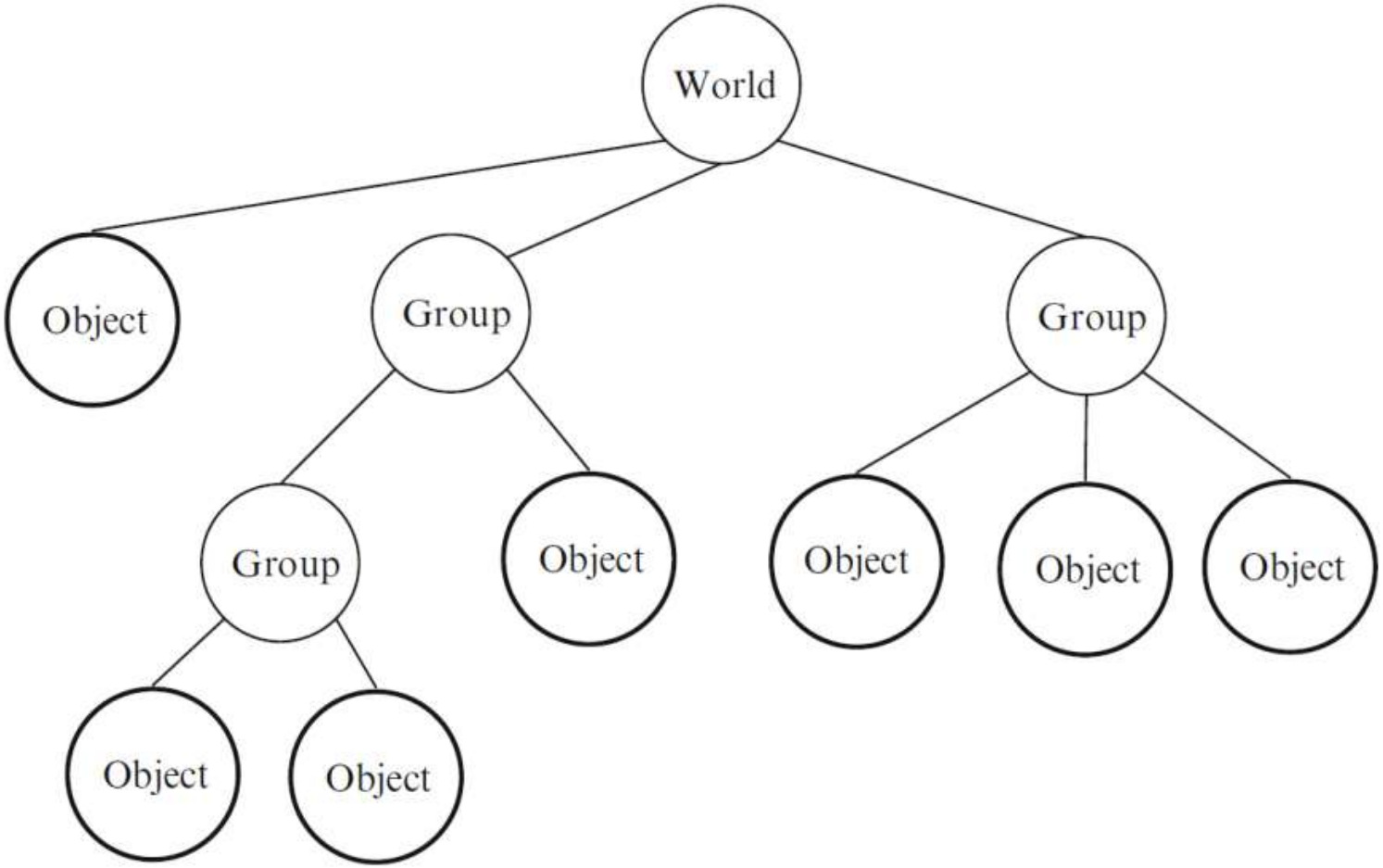
- Transformando el cuadrado y el triángulo podemos dibujar el pino
- Esta serie de operaciones es lo que se conoce como el grafo de escena.

Grafo de Escena

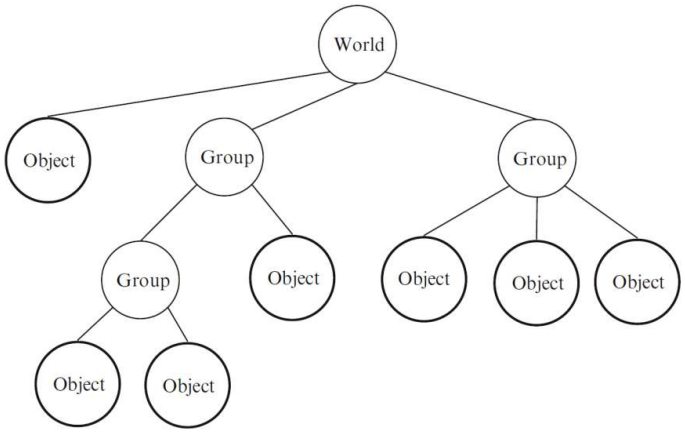


- La misma idea se expande para generar la escena completa de muchos pinos.
- Cada flecha involucra una transformación.

Grafo de Escena

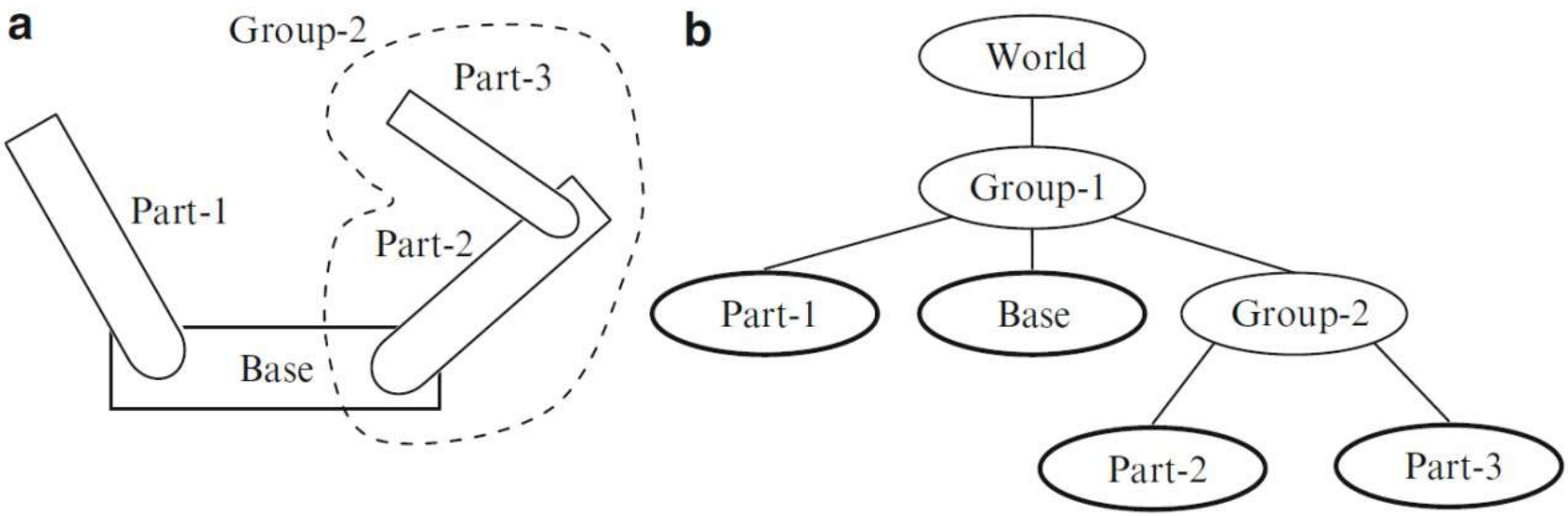


Grafo de escena



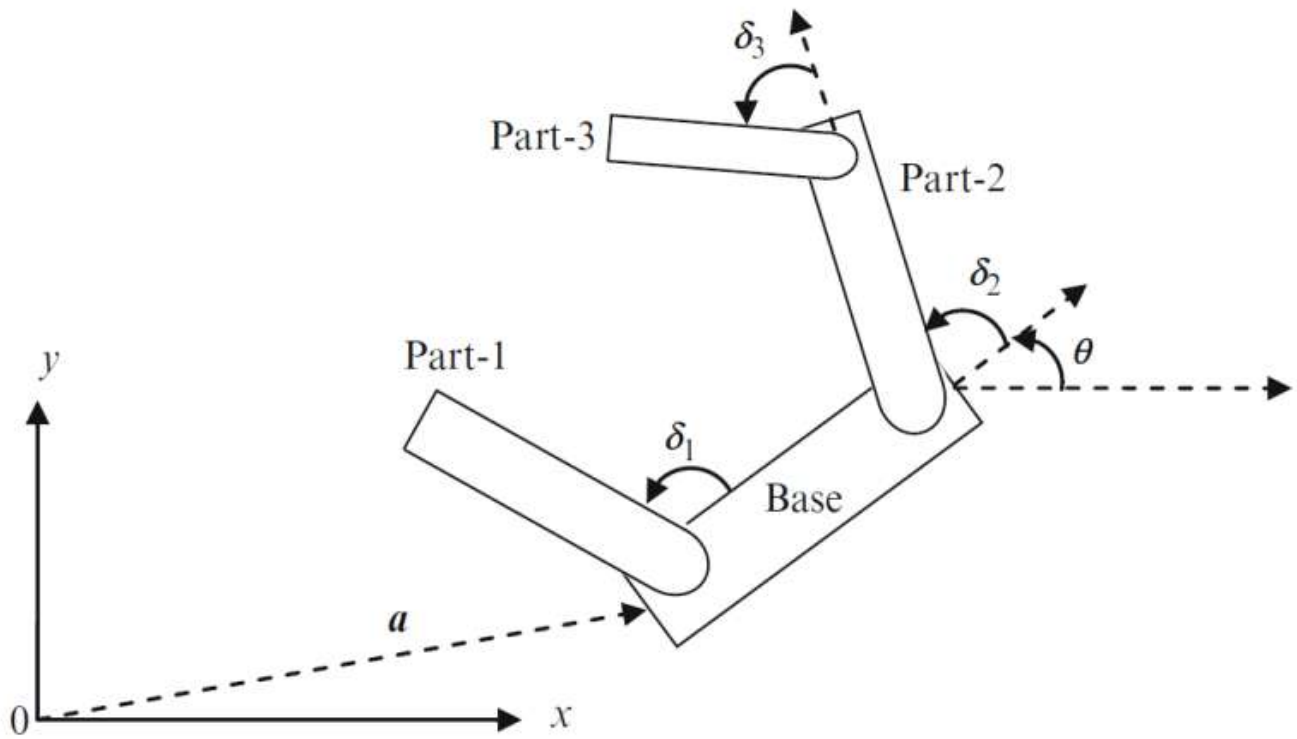
- Cada hoja del árbol es un objeto básico.
- Cada nodo interno representa un grupo de objetos.
- Cada arco representa una transformación.
- La transformación final de un objeto es la composición de todas las transformaciones desde la raíz hasta la hoja.

Grafo de escena



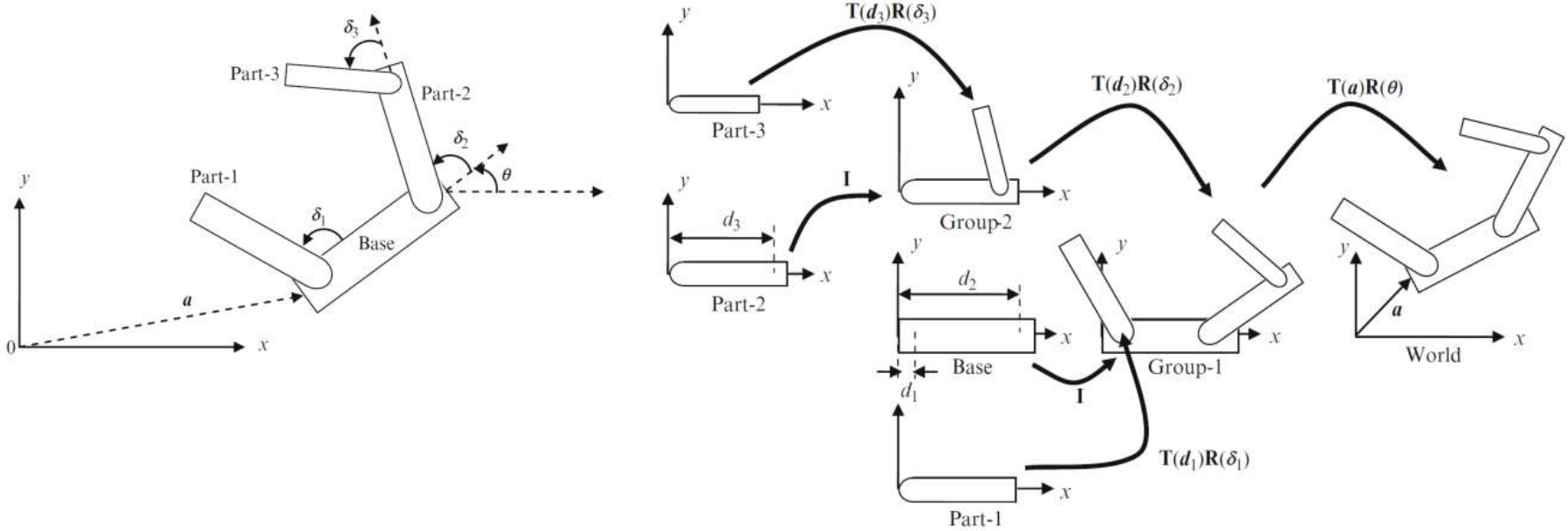
Grafo de escena

- Objeto con piezas móviles en una escena



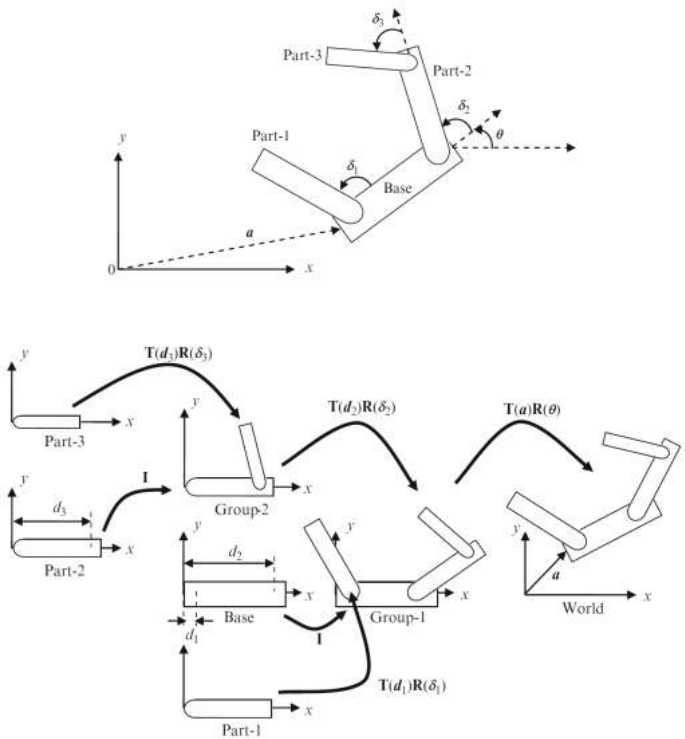
Grafo de escena

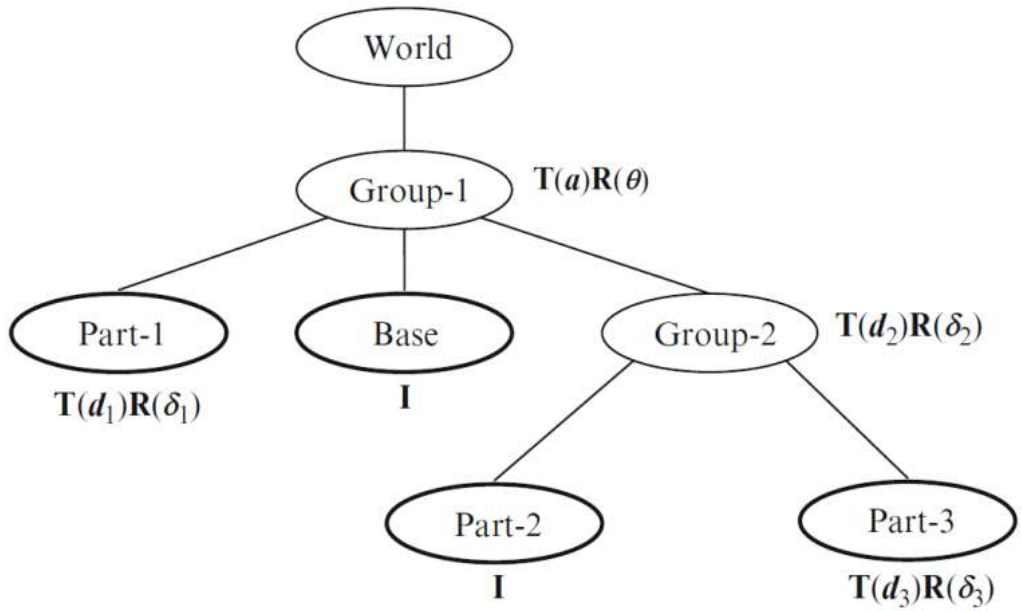
- Modelando un objeto con piezas móviles



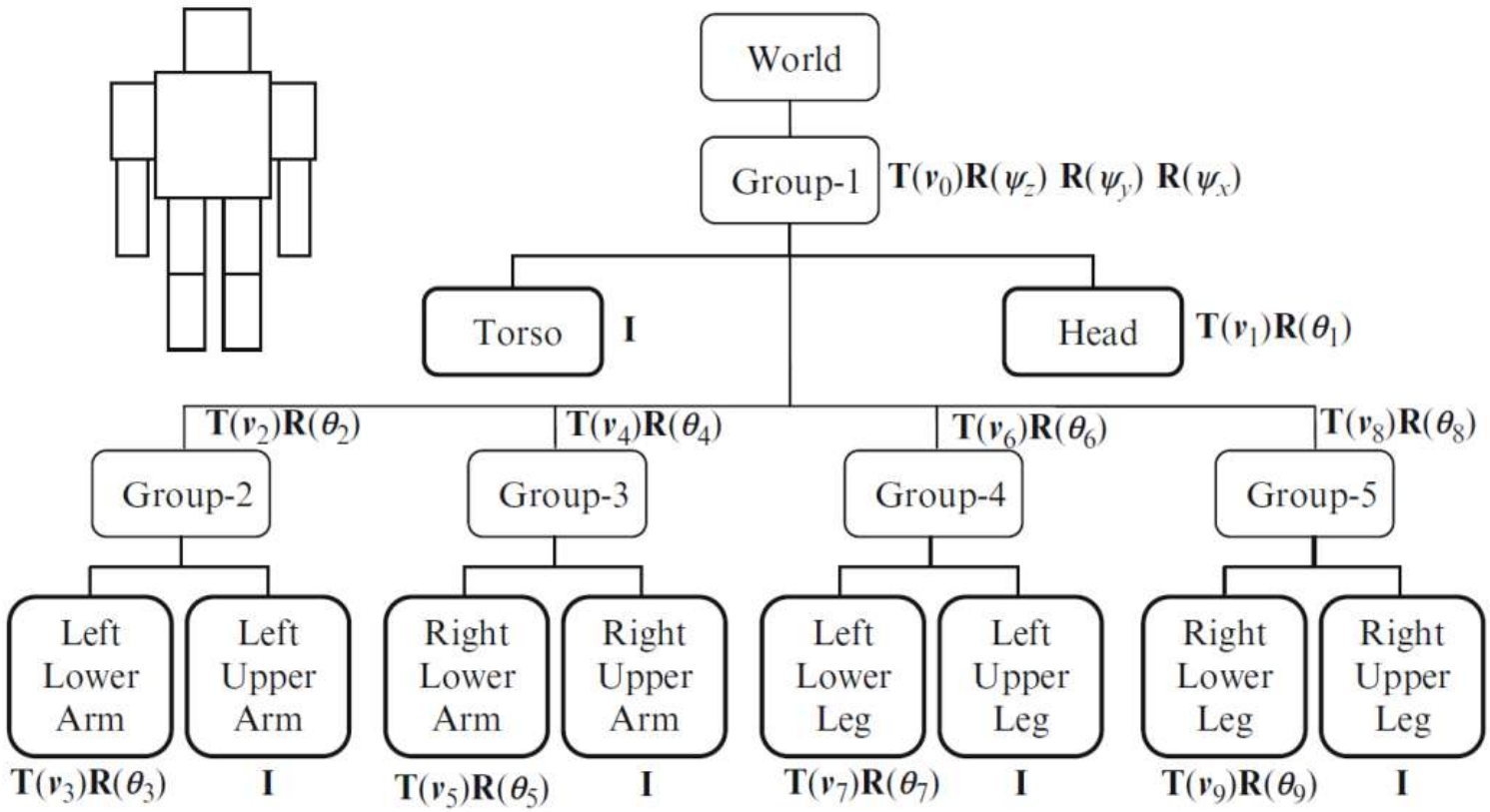
Grafo de escena

- Modelando un objeto con piezas móviles

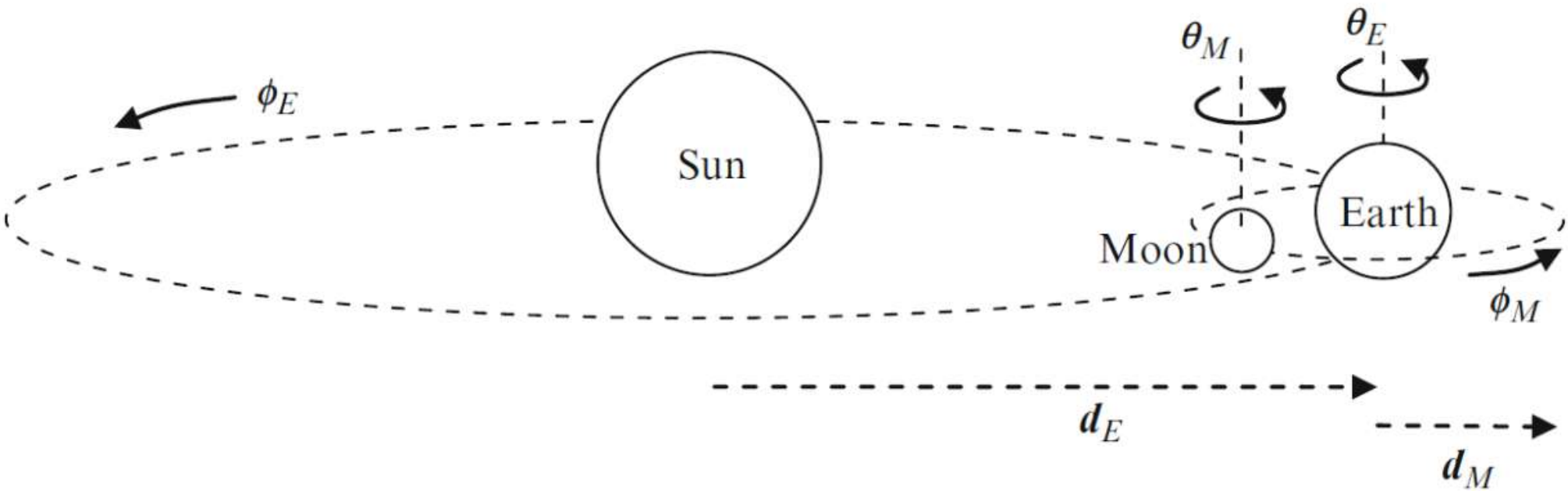




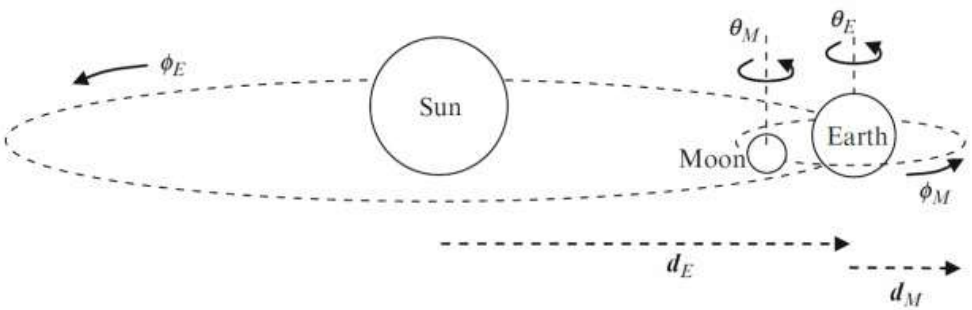
Ejemplo: figura humanoide

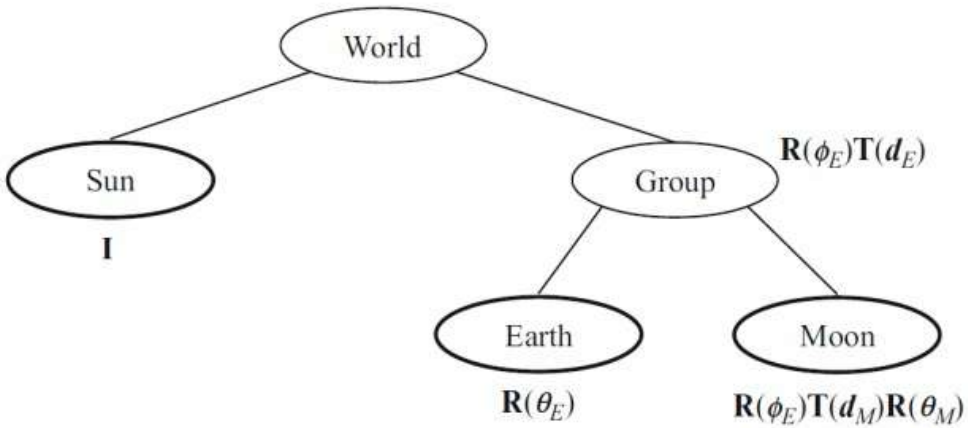


Ejemplo: Sistema planetario



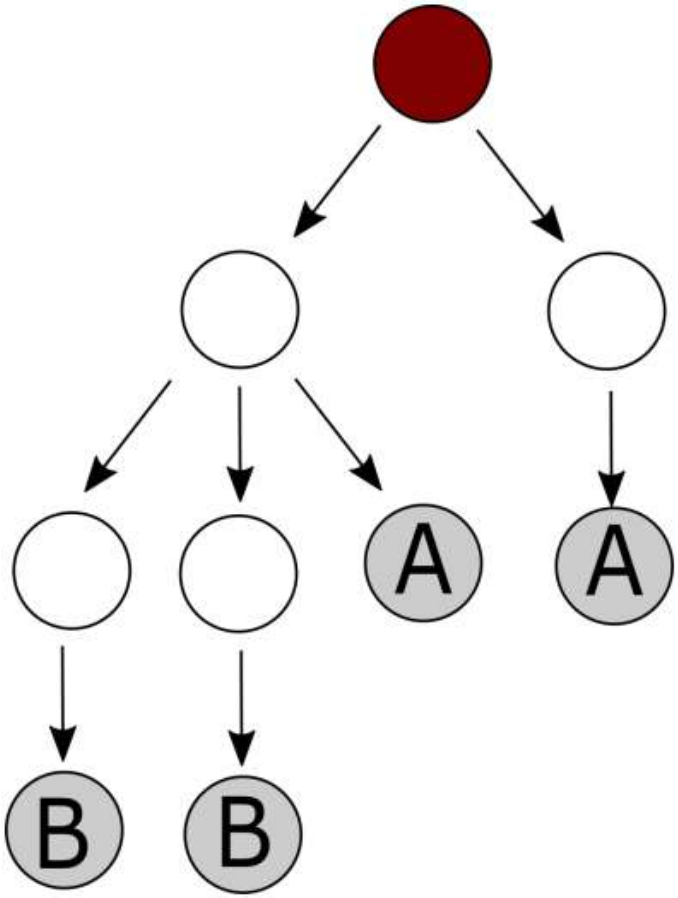
Ejemplo: Sistema planetario





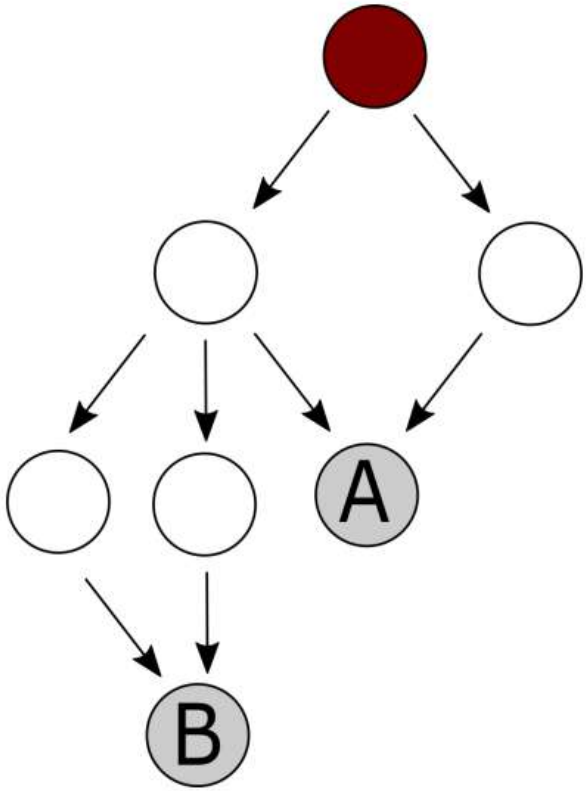
Grafo de escena – Árbol

- Estructura ampliamente usada
- Cada nodo posee uno o mas nodos hijos
- Cada nodo posee un único padre
- Transformaciones se acumulan desde el nodo raíz hasta las hojas
- Cada nodo hoja representa un objeto de la escena



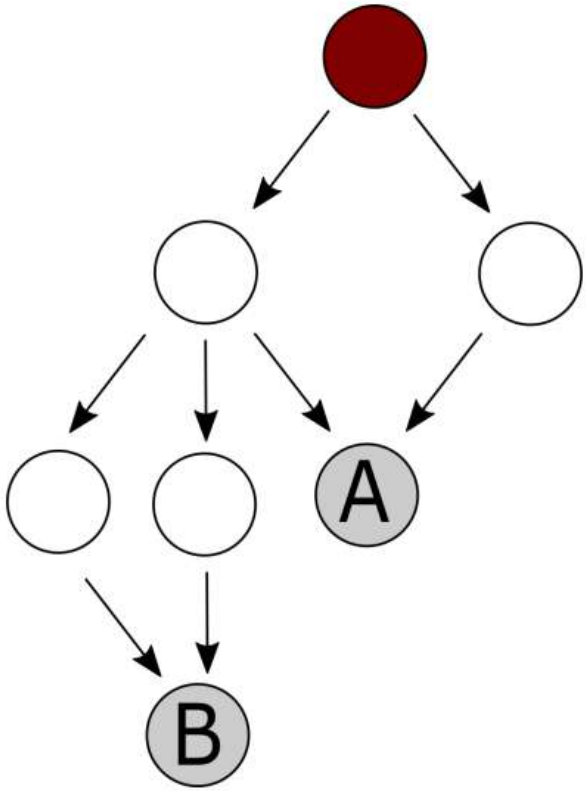
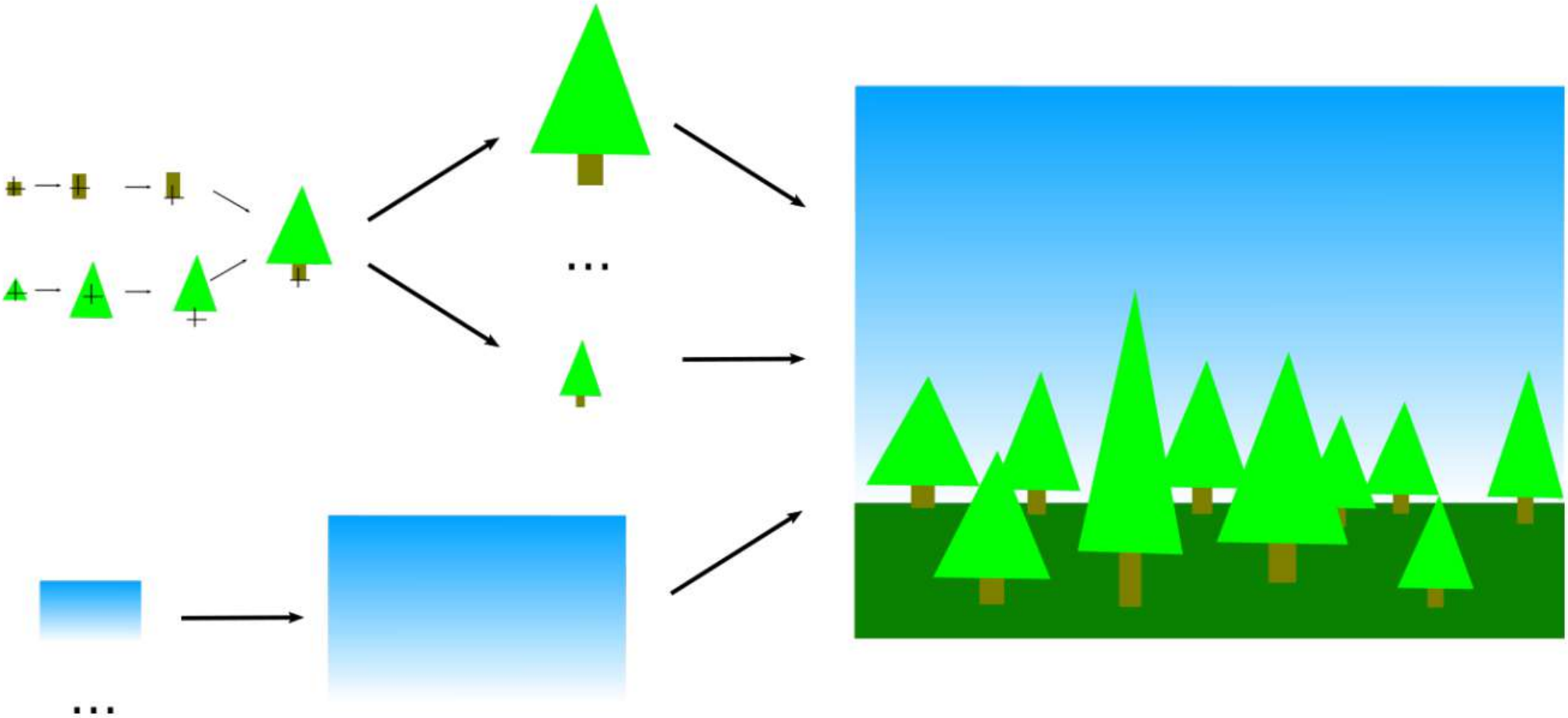
Grafo de escena- DAG

- Cada objeto puede ser instanciado múltiples veces
- Es decir, cada nodo puede tener múltiples padres
- Esto se logra modelando la escena usando un DAG.
- DAG: Directed Acyclic Graph
  - Dirigido porque arcos van en una sola dirección
  - Acíclico significa que no hay loops al recorrer el grafo



Grafo de escena - DAG

- Las transformaciones aún se acumulan desde el nodo raíz hasta las hojas (igual que con el árbol)
- Como los objetos ya se encuentran en el GPU, se aprovecha para optimizar almacenamiento.



Ejemplo DAG