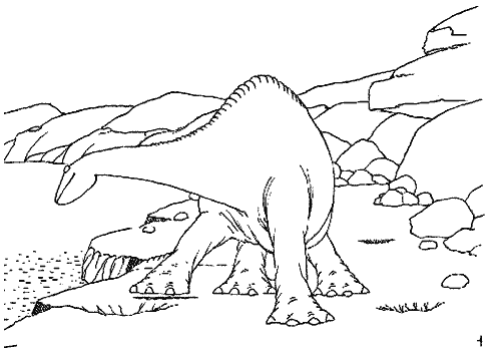
El Mundo 2D



Dpto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Introducción

Los distintos procedimientos que vimos para mostrar las primitivas de salida y sus atributos nos permiten crear una gran variedad de imágenes y gráficos.

A partir de estas primitivas podemos generar escenas 2D, cambiando, por ejemplo, orientaciones y tamaño de las distintas componentes; también podemos generar animaciones moviendo los objetos en la escena a lo largo de distintos caminos.

Estos cambios en orientación, tamaño y forma se llevan a cabo mediante *transformaciones geométricas* que alteran las descripciones de las coordenadas de los objetos.

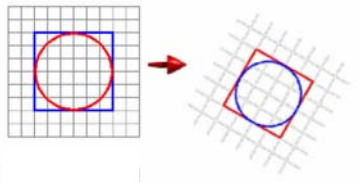
Introducción

La traslación, la rotación, el escalado y el sesgado son ejemplos de transformaciones geométricas. Estas transformaciones son lineales; también podemos usar transformaciones no lineales.

Veremos primero cómo llevar a cabo estas transformaciones y luego, a partir de esto, veremos cómo podemos crear escenas 2D a partir de éstas y de objetos primitivos.

Transformaciones

Las transformaciones geométricas nos permiten mapear puntos de una región a otra



y pueden aplicarse a:

- Primitivas de dibujado
- Coordenadas de pixel de una imagen

Transformaciones

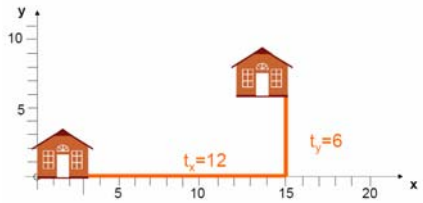
Las transformaciones geométricas nos permitirán, entre otras cosas,

- Mapear puntos de un sistema de coordenadas en otro.
- Cambiar la forma de los objetos
- Posicionar objetos en una escena
- Crear múltiples copias de objetos en la escena
- Proyectar escenas tridimensionales en la pantalla
- Crear animaciones
- …

Transformaciones Geométricas en 2D

Traslación

Esta operación se usa para mover un objeto o grupo de objetos de manera lineal a una nueva ubicación en el espacio bidimensional.



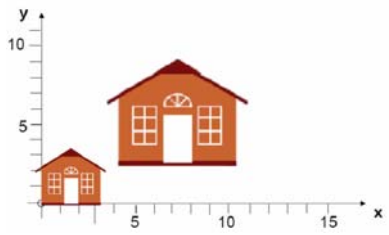
Trasladar un objeto una distancia t_x en x y una distancia t_y en y se expresa como:

$$x' = x + t_x \qquad y' = y + t_y \quad \Longrightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Transformaciones Geométricas en 2D

Escalado

Es una transformación que permite cambiar el tamaño o la proporción de un objeto o grupo de objetos. Hay escalados proporcionales y no proporcionales.



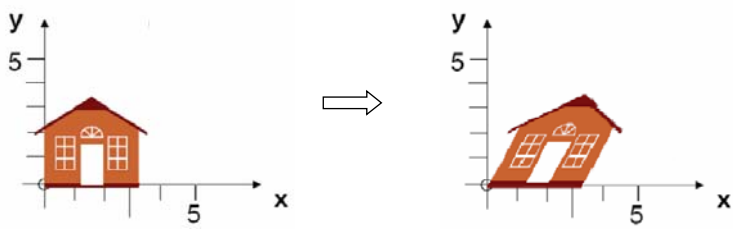
Escalar un objeto en e_x según x y en e_y según y se expresa como:

$$\begin{aligned} x' &= e_x x \\ y' &= e_y y \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} e_x & 0 \\ 0 & e_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Transformaciones Geométricas en 2D

Sesgado

Un objeto se puede sesgar tanto en sentido horizontal como en sentido vertical.



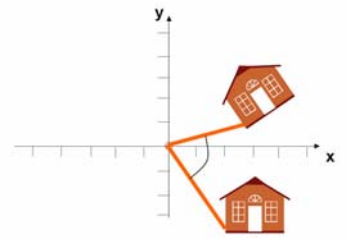
Sesgar un objeto en sentido horizontal se expresa como:

$$\begin{aligned} x' &= x + ay \\ y' &= y \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Transformaciones Geométricas en 2D

Rotación

Esta transformación geométrica se usa para mover un objeto o grupo de objetos alrededor de un punto.



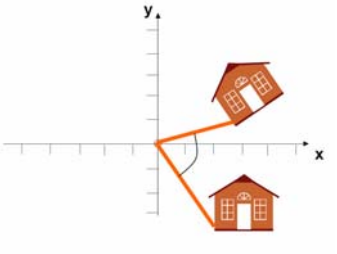
Rotar un objeto un ángulo α en sentido horario se expresa como:

$$x' = x\cos\alpha - y\sin\alpha$$
$$y' = x\sin\alpha + y\cos\alpha$$
$$\Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Transformaciones Geométricas en 2D

La matriz de rotación tiene ciertas propiedades

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



La norma de cada fila es uno: $(\sin\alpha)^2 + (\cos\alpha)^2 = 1$

Las filas son ortogonales : $\cos\alpha(-\sin\alpha) + \sin\alpha\cos\alpha = 0$

Decimos que las matrices de rotación son *ortonormales*.

Teniendo esto en cuenta vemos entonces que la inversa de una matriz ortonormal es su traspuesta.

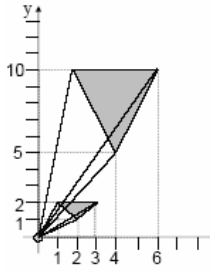
Transformaciones Geométricas en 2D

Las transformaciones de traslación y rotación se conocen como *transformaciones de cuerpo rígido*. Estas transformaciones preservan las distancias y los ángulos.

Si a las transformaciones de cuerpo rígido les adicionamos las transformaciones de reflexión y escalado uniforme, tenemos las *transformaciones de similaridad*. Éstas preservan los ángulos, las distancias entre puntos cambian en una proporción fija y se mantiene una forma similar (triángulos similares, círculos mapean a círculos, …).

Transformaciones Geométricas en 2D

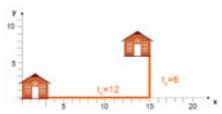
Si a las transformaciones de similaridad les adicionamos las transformaciones de desplazamiento y escalado no uniforme, tenemos las *transformaciones afines*.

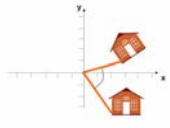


Éstas preservan las líneas paralelas.

Transformaciones Geométricas en 2D

Si tenemos las transformaciones :





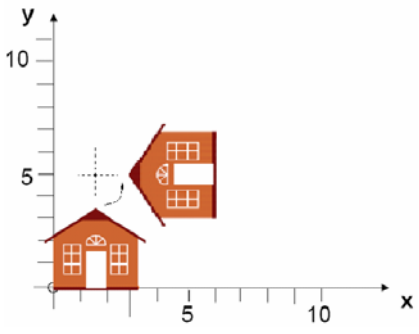
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} e_x & 0 \\ 0 & e_y \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix}$$

… podemos escribir cada punto transformado como:

$p' = p + T$ $\qquad$ $p' = E\ p$ $\qquad$ $p' = Rp$

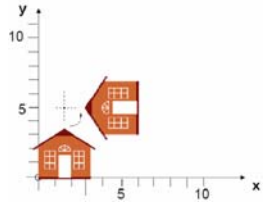
Transformaciones Geométricas en 2D

¿Qué ocurre cuando queremos realizar una determinada transformación con respecto a un punto cualesquiera?



Supongamos que queremos rotar 90 grados con respecto al punto (1.5,5)

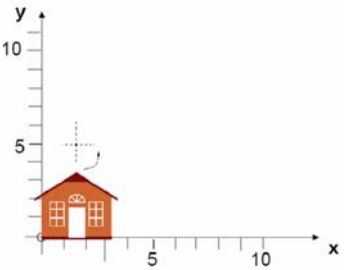
Transformaciones Geométricas en 2D



No podemos aplicar las fórmulas que vimos directamente ya que definen rotaciones con respecto al origen.

¿Cómo se ubicaría la casita si aplicamos la transformación que vimos?

¿Qué pasos debemos seguir para lograr nuestro objetivo?



Transformaciones Geométricas en 2D

Escribimos las transformaciones como:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} e_x & 0 & 0 \\ 0 & e_y & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Así podemos escribir cada punto como:

p´ = Tp p´ = E p p´ = Rp

Estamos trabajando en coordenadas homogéneas.

Transformaciones Geométricas en 2D

Al poder escribir cada punto como:

$p' = Tp$ $\quad$ $p' = E\,p$ $\quad$ $p' = Rp$

podemos aplicar sucesivas transformaciones a un punto concatenando las matrices de transformación, incluida la traslación. Para responder a la pregunta:

¿Qué transformaciones tenemos que aplicar se queremos rotar un objeto alrededor de un punto cualesquiera?

Aplicamos las transformaciones del siguiente modo:

$$p' = T_1 p \qquad p'' = R\,p' \qquad p''' = T'_1\,p''$$

$$\Rightarrow \quad p''' = T'_1\,R\,p'$$

$$\Rightarrow \quad p''' = T'_1\,R\,T_1 p$$

Las matrices aparecen en orden reverso al que son aplicadas las transformaciones.

Transformaciones Geométricas en 2D

Dadas las transformaciones:

$$p''' = T'_1\,R\,T_1\,p$$

La matriz de transformación que se aplica a cada punto es:

$$M_T = T'_1\,R\,T_1$$

Esto significa que estamos haciendo primero la translación T_1, luego la rotación R y luego la translación T'_1.

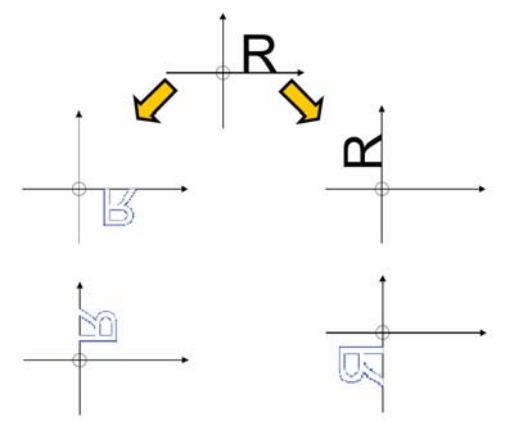
¡El orden importa! Recuerden que

$$AB \neq BA$$

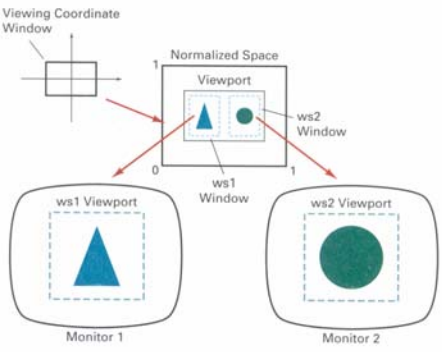
Transformaciones Geométricas en 2D

Ejemplo:

Se aplican las transformaciones de rotación de 90º y reflexión sobre x.

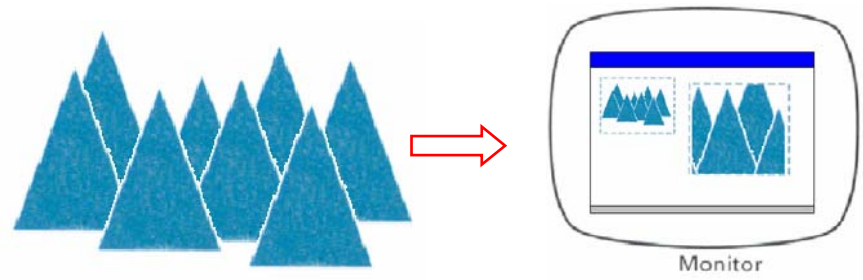


Del Mundo de los Objetos 2D a la Pantalla



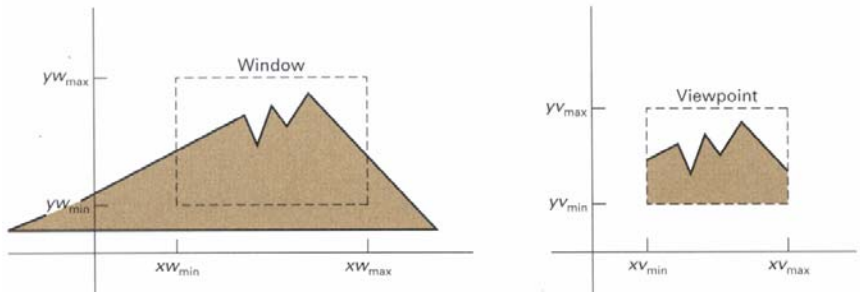
Vistas 2D de un mundo 2D

Del mundo a la pantalla



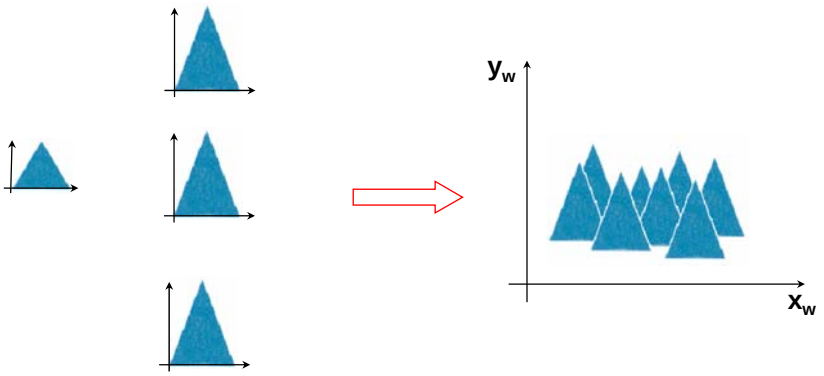
Vistas 2D de un mundo 2D

Consideraremos ahora los mecanismos formales para mostrar vistas de un mundo 2D en un dispositivo de salida.



Vistas 2D de un mundo 2D

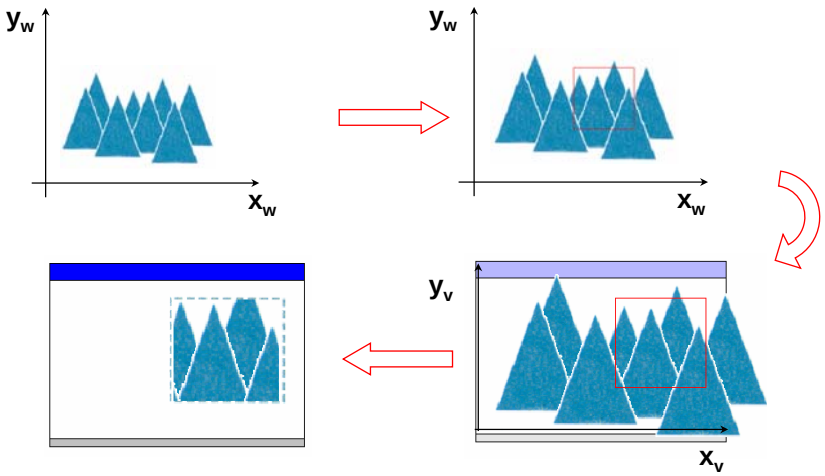
¿Cuáles son las distintas etapas que deben atravesarse?



Coord. del objeto Coord. del mundo

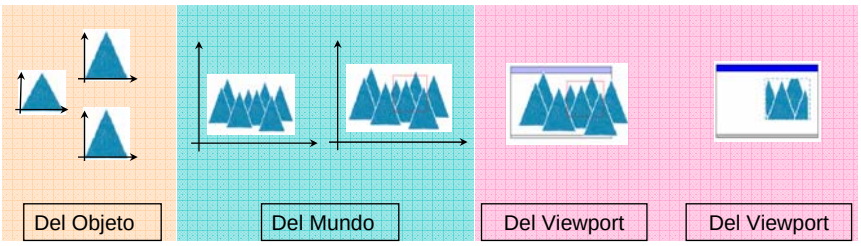
Vistas 2D de un mundo 2D

Del mundo a la pantalla



Vistas 2D de un mundo 2D

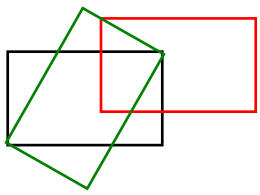
Del mundo a la pantalla



¿Cuáles son las transformaciones que deben aplicarse para ir desde las coordenadas del objeto a las de la pantalla?

Dibujamos varios rectángulos en OpenGL

Supongamos que queremos dibujar el siguiente grupo de objetos en OpenGL



glLoadIdentity()
DibujoR()
glLoadIdentity()
glTranslate(…)
DibujoR()
glLoadIdentity()
glRotate(…)
DibujoR()

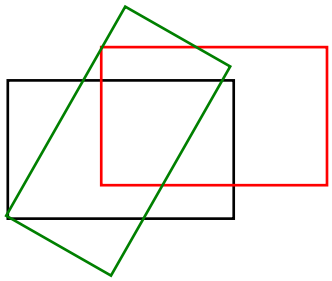
Hacerlo de este modo es muy ineficiente, porque se debe especificar la transformación de viewing y luego el resto cada vez que se dibuja un rectángulo.

Una mejor forma de hacerlo es con las pilas de matrices. La matriz de modelado y la de vista conforman una única matriz que es el tope de la pila de matrices.

Dibujamos varios rectángulos en OpenGL

¿Cómo dibujaríamos el grupo de rectángulos?

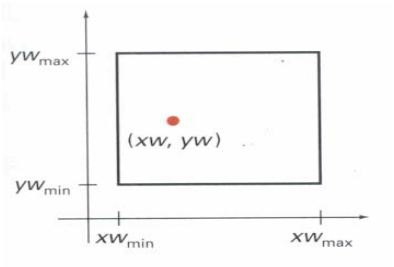
Me sitúo en el origen de un sistema de referencia … Dibujo el rectángulo negro y dejo marca para volver aquí. Me traslado a donde debo dibujar el rectángulo rojo y lo dibujo. Recupero la marca y vuelvo allí. …

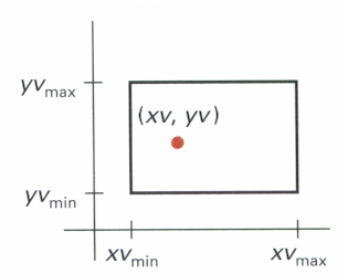


DibujarRectánulos(…)
- *DibujarRectángulo(…)*
- *glPushMatrix()*
  - *glTranslatef(…)*
  - *DibujarRectangulo()*
- *glPopMatrix()*
- *glPushMatrix()*
  - *glRotatef(…)*
  - *DibujarRectangulo(…)*
- *glPopMatrix()*
- *…*

Transformación del mundo al viewport

Del mundo al viewport





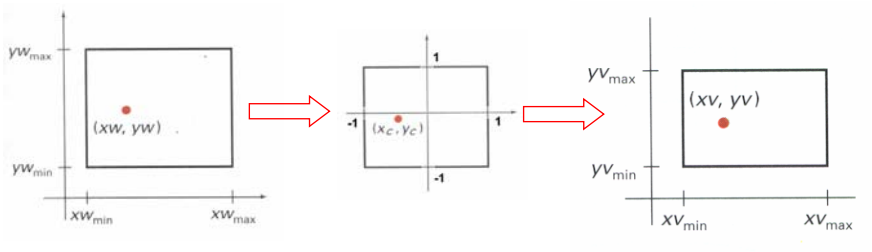
$$\frac{x_v - XV_{min}}{XV_{max} - XV_{min}} = \frac{x_w - XW_{min}}{XW_{max} - XW_{min}} \qquad \frac{y_v - YV_{min}}{YV_{max} - YV_{min}} = \frac{y_w - YW_{min}}{YW_{max} - YW_{min}}$$

Del mundo al viewport podemos despejar cada coordenada; para *x* es

$$x_v = \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} x_w + \left(XV_{min} - \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} XW_{min} \right) \qquad x_v = Ax_w + (XV_{min} - AXW_{min}) = Ax_w + B$$

Transformación del mundo al viewport

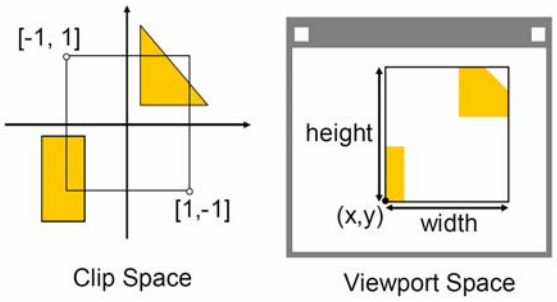
La transformación del mundo al viewport se hace habitualmente en dos etapas: del mundo al espacio de clipping y de éste al espacio de la pantalla.



Transformación del mundo al viewport

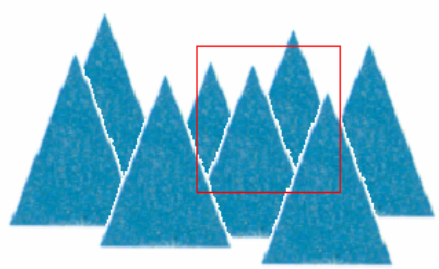
OpenGL brinda una función para establecer la transformación del Viewport

```
glViewport(x, y, width, height);
```



Clipping

Es el proceso de determinar qué primitivas y partes de primitivas caen dentro del volumen de clipping definido por el programa de aplicación.



Generalmente, cualquier procedimiento que identifica aquellas porciones de una escena que están dentro o fuera de una región especificada se denomina *algoritmo de clipping*.

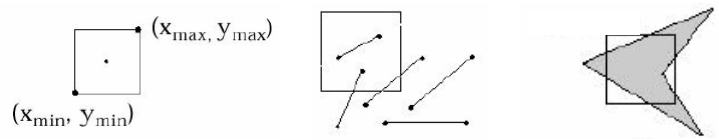
La región contra la cual se clipea un objeto se denomina ventana de clipping.

Clipping

Las primitivas que caen dentro de la ventana de clipping son aceptadas para su posterior renderización.

Las primitivas que no caen dentro de la ventana de clipping son rechazadas.

Las primitivas que están sólo parcialmente dentro de la región de vista deben ser recortadas de modo tal que cualquier parte de las mismas que esté fuera de la región sea removida.



Clipping

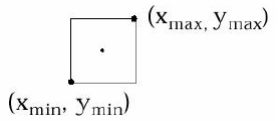
Consideraremos algoritmos para el clipping de los siguientes tipos de primitivas:

- Clipping de puntos
- Clipping de rectas
- Clipping de polígonos
- Clipping de curvas

El clipping de rectas y polígonos son componentes estándar de la mayoría de los paquetes gráficos. Los objetos curvos se aproximan con líneas rectas y polígonos planos a los que se aplican los algoritmos de clipping de rectas y polígonos.

Clipping

Si la primitiva es un punto:



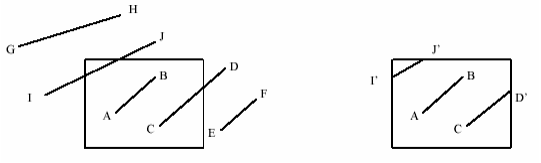
En este caso sabemos que si se verifica que:

$xmin < x < xmax \qquad ymin < y < ymax$

el punto está dentro de la ventana; si no, está fuera de la misma.

Clipping

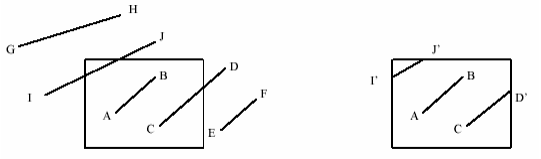
Si la primitiva es una línea:



tenemos las siguientes posibilidades:

- Líneas que están completamente dentro de la ventana
- Líneas que están completamente afuera de la ventana
- Líneas que intersectan la ventana

Clipping



El algoritmo de clipping debe:

-Identificar y dibujar las líneas que están completamente *dentro* de la ventana (AB).
- Identificar y descartar las líneas que están completamente *afuera* de la ventana (GH, EF).
- Encontrar las intersecciones de las líneas que están *parcialmente dentro* de la ventana y dibujar las secciones de las mismas que están dentro de la ventana (IJ, CD).

Clipping

Para

- Identificar y dibujar las líneas que están completamente dentro de la ventana.
- Identificar y descartar las líneas que están completamente afuera de la ventana.
- Encontrar las intersecciones de las líneas que están parcialmente dentro de la ventana y dibujar las secciones de las mismas que están dentro de la ventana.

necesitamos las representaciones matemáticas de esas rectas y de la ventana.

Por fuerza bruta podríamos resolver ecuaciones simultáneas usando la ecuación de la recta para la misma ($y = mx + b$) y los cuatro lados de la ventana.

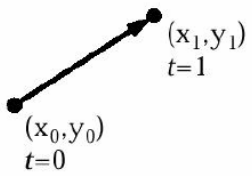
Problemas:

- La fórmula sólo maneja líneas infinitas
- No maneja líneas verticales

Clipping

Para la línea utilizaremos la ecuación paramétrica ya que es adecuada para un segmento de recta:

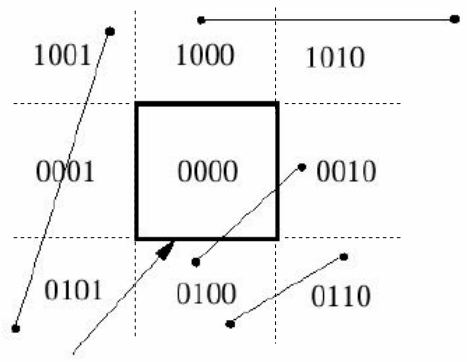
$$x = x_0 + t(x_1 - x_0) \qquad 0 < t < 1$$
$$y = y_0 + t(y_1 - y_0)$$
$$p(t) = p_0 + t(p_1 - p_0)$$



Algoritmo de Clipping de Cohen-Sutherland
(2D)

Algoritmo de Cohen-Sutherland

Divide el plano en 9 regiones:



Ventana

Algoritmo de Cohen-Sutherland

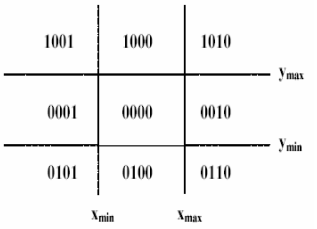
Para identificar rápidamente las líneas que están dentro y fuera de la ventana, les asigna un código de región de 4 bits a cada uno de los puntos extremos del segmento de recta

bit: 1	2	3	4
x	x	x	x

Código

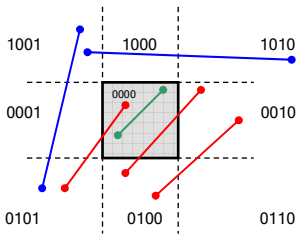
Los bits son puestos en 1 (uno) de acuerdo a:

- bit 1 → punto ***por arriba*** de la ventana
- bit 2 → punto ***por debajo*** de la ventana
- bit 3 → punto ***a la derecha*** de la ventana
- bit 4 → punto ***a la izquierda*** de la ventana



Algoritmo de Cohen-Sutherland

Si Código1=0 y Código2=0
entonces
  el segmento de línea está dentro de la ventana
sino
  si (Código1 and Código2)≠0
  entonces
    el segmento de línea está fuera de la ventana
  sino
    no sabemos



Algoritmo de Cohen-Sutherland

Entonces consideramos tres casos

→ Una línea es ***trivialmente aceptada*** si los ***códigos*** de ambos puntos extremos son ***0000.***

→ Una línea puede ser ***trivialmente rechazada*** si la operación ***AND bit a bit*** de los ***códigos*** de los puntos extremos ***NO es 0000 .***

→ De otro modo, se requiere ***más procesamiento*** ya que la línea ***no*** puede ser ni trivialmente aceptada ni trivialmente rechazada.

Algoritmo de Cohen-Sutherland

Ejemplos:

→ ***AB*** es trivialmente aceptada porque

A: 0000

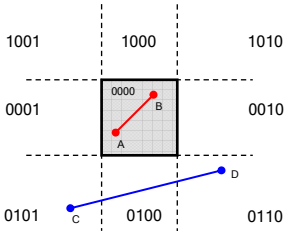
B: 0000

→ **CD** es trivialmente rechazada porque

D: 0110

C: 0101

AND = 0100



Algoritmo de Cohen-Sutherland

Ejemplos:

→ ***EF*** necesita más procesamiento: *(no puede ser trivialmente aceptada o rechazada)*

F: 1000 AND E: 0001 = 0000

→ ***GH :*** es clipeada contra el límite sup para obtener así ***H'***

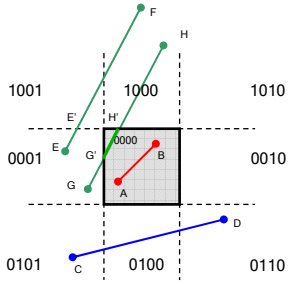
H'H es rechazada,

GH' necesita más procesamiento(clip con los bordes superior, inferior, derecho y luego izquierdo de acuerdo a como se tienen en cuenta los bordes),

clipeado contra el borde izquierdo se obtiene ***G'***

GG' es rechazado,

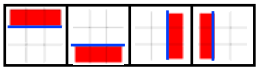
G'H' es trivialmente aceptado.



Algoritmo de Cohen-Sutherland

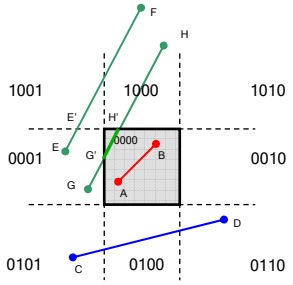
Aplicamos el algoritmo de clipping sobre un segmento de recta. Si éste no puede aceptarse o rechazarse trivialmente, debemos subdividirlo en dos segmentos; entonces aplicamos el algoritmo de clipping sobre cada uno de estos segmentos.

- Se debe convenir un orden para chequear cada uno de los lados de la ventana:
arriba - abajo - derecha - izquierda



- Para realizar la subdivisión en dos segmentos se calcula el punto de intersección; el lado contra el que se clipea fija o bien *x* o bien *y*, que puede entonces sustituirse en la ecuación de la recta

- Los códigos se pueden usar para elegir el lado de la ventana que se cruza con el segmento de recta: si los códigos difieren en el bit del lado, los puntos extremos deben cruzar ese lado.



Algoritmo de Cohen-Sutherland

Algoritmo ***Código***
DE: p (coord. x,y)
Ventana (coord. xmin,ymin,xmax,ymax)
DS: codigo

```
codigo ← centro
si x<xmin
entonces
  codigo ← codigo or izquierda
sino
   si x>xmax
   entonces
      codigo ← codigo or derecha

si y<ymin
entonces
  codigo ← codigo or abajo
sino
   si y>ymax
   entonces
      codigo ← codigo or arriba
```

Suponemos que el código es una constante:

centro ← 0000
izquierda ← 0001
derecha ← 0002
abajo ← 0004
arriba ← 0008

Algoritmo de Cohen-Sutherland

Algoritmo ***Descartar Segmento***
DE: codigo
p0(coord x0,y0), p1(coord x1,y1)
Ventana (coord xmin,ymin,xmax,ymax)
DS: p (coord x,y)

```
si (codigo and arriba)<>0
entonces
  x = x0 + (x1 – x0) * (ymax – y0)/(y1 – y0); y = ymax;
sino
   si (codigo and abajo)<>0
   entonces
      x = x0 + (x1 – x0) * (ymin – y0)/(y1 – y0); y = ymin;
   sino
       si (codigo and derecha)<>0
      entonces
          y = y0 + (y1 – y0) * (xmax – x0)/(x1 – x0); x = xmax;
      sino
        si (codigo and izquierda)<>0}
        entonces
            y = y0 + (y1 – y0) * (xmin – x0)/(x1 – x0); x = xmin;
```

Algoritmo de Cohen-Sutherland

Algoritmo ***Clipping***

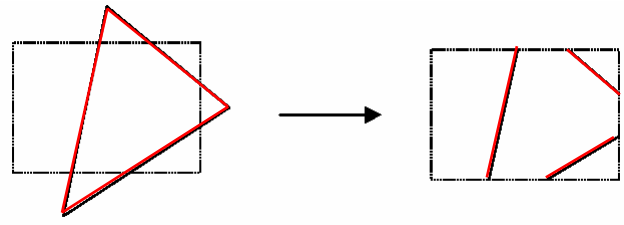
DE: p0(coord. x0,y0), p1(coord. x1,y1)
    Ventana (coordenadas xmin,ymin,xmax,ymax)
DS: p0(coordenadas x0,y0), p1(coordenadas x1,y1)
    Se Dibuja (booleano)

```
Código(p0,Ventana,codigo0); Código(p1,Ventana,codigo1);
si (codigo0 = centro) and (codigo1 = centro)
entonces
  Se Dibuja ← verdadero
sino
  si (codigo0 and codigo1 )<>centro
  entonces
     Se Dibuja ← falso
  sino
     si codigo0 <> centro
     entonces
        Descartar Segmento (codigo0,p0,p1,p)
       Clipping(p,p1,Ventana,Se Dibuja); p0 ← p
     sino
        Descartar Segmento (codigo1,p1,p0,p)
       Clipping(p0,p,Ventana,Se Dibuja); p1 ← p
```


Algoritmo de Sutherland-Hodgeman

El clipping de polígonos no puede hacerse con los algoritmos de clipping de líneas ya que no se obtiene un área cerrada.

El clipping de líneas individuales conduce a:



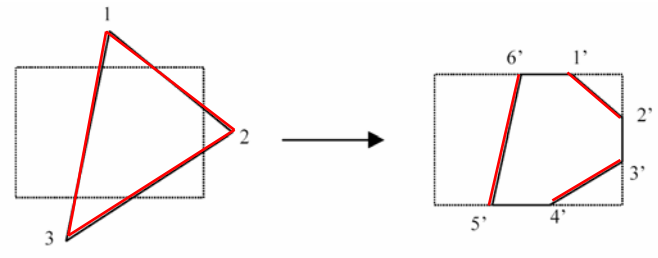
Algoritmo de Sutherland-Hodgeman

Dado el polígono que se ve en la figura:

Polígono de entrada: <1, 2, 3> donde 12, 23, 31 son lados

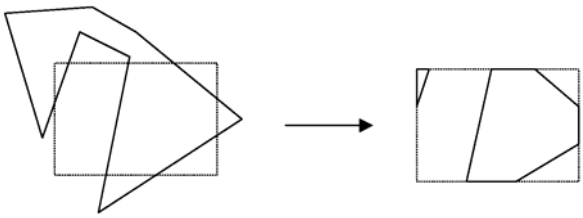
El clipping de polígonos debe hacerse de modo tal que se obtenga el polígono:

Polígono de salida: <1', 2', 3', 4', 5', 6'>



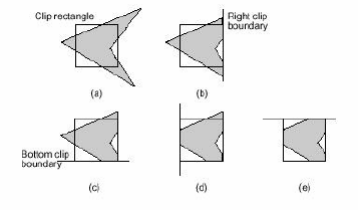
Algoritmo de Sutherland-Hodgeman

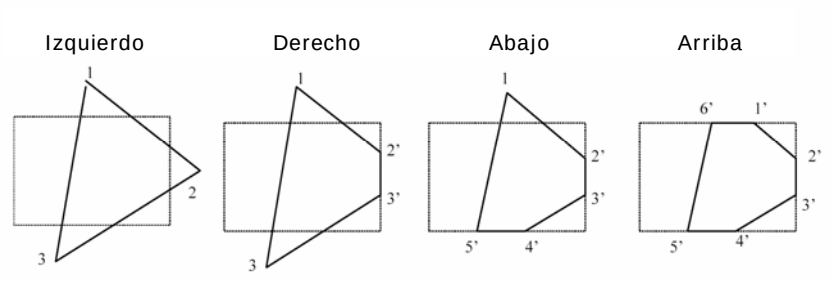
Podemos hacer lo siguiente: por cada pixel en el polígono, decidir si está dentro del volumen de vista. Esto funciona, pero es demasiado ineficiente ya que hace la conversión scan de todo el polígono aunque sólo dibuja las partes visibles. Este proceso es denominado ***scissoring*** (*clipping on the fly* durante la conversión scan).



Algoritmo de Sutherland-Hodgeman

El algoritmo de Sutherland-Hodgeman procesa cada polígono, en orden, con cada uno de los bordes.

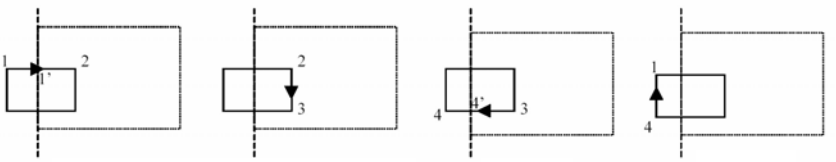




Algoritmo de Sutherland-Hodgeman

Para cada uno de los bordes de la ventana, se procesan todos los vértices del polígono; se genera un conjunto de vértices que se utilizará con el próximo borde.

En cada paso del procesamiento, pueden presentarse 4 casos posibles cuando se va de un punto extremo del segmento de línea al otro.



Afuera → Adentro
Guardar 1´,2

Adentro → Adentro
Guardar 3

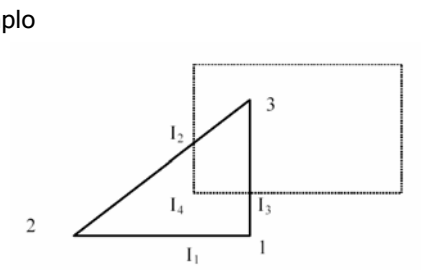
Adentro → Afuera
Guardar 4´

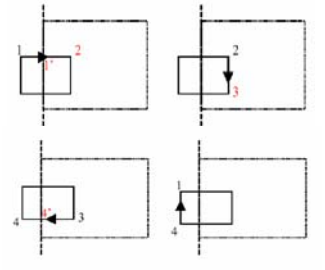
Afuera → Afuera
No guardar nada

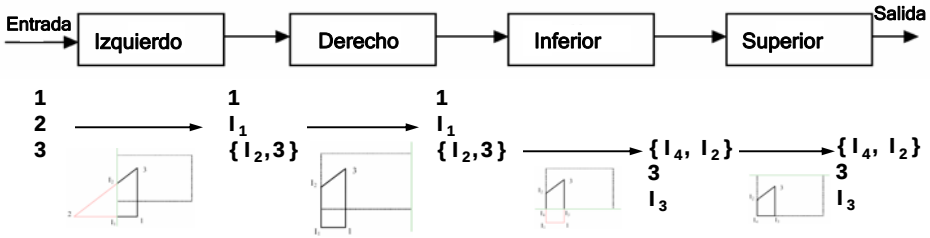
El clipping del polígono 1234 con el borde izquierdo de la ventana genera el polígono 1´234´.

Algoritmo de Sutherland-Hodgeman

Ejemplo

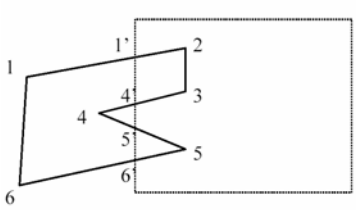


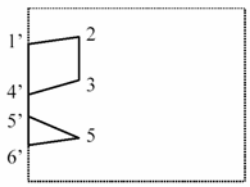




Algoritmo de Sutherland-Hodgeman

El algoritmo de Sutherland-Hodgeman clipea correctamente los polígonos convexos. Los polígonos cóncavos los clipea con líneas extrañas:



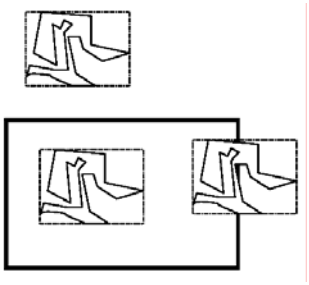


Esto ocurre porque los polígonos cóncavos pueden generar múltiples áreas pero el algoritmo sólo genera una lista de vértices de entrada.

En el ejemplo, el punto 1´ se une al punto 6´a lo largo del borde izquierdo de la ventana.

Clipping de otras primitivas

La *Bounding Box* más simple (o *extent*) es el menor rectángulo, alineado con la ventana, que contiene al polígono. Se puede evitar el clipeado detallado de polígonos complejos chequeando si la *bounding box* está completamente dentro o completamente fuera de la ventana. Si está parcialmente adentro, es necesario un *clipping* detallado.



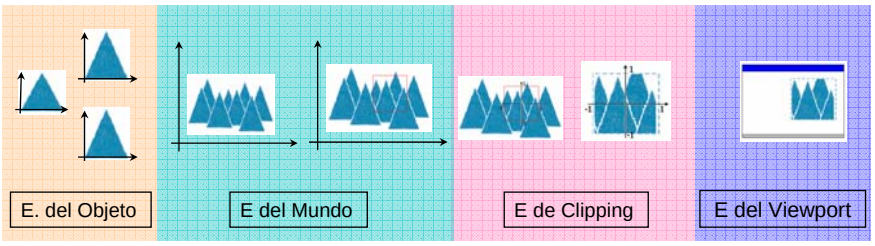
Clipping de otras primitivas

Para *curvas y superficies* es dificultoso encontrar algoritmos generales.

El problema se evita aproximando las curvas por segmentos de líneas y las superficies mediante polígonos planos y clipeando estas primitivas de acuerdo a lo visto.

Del mundo 2D a la pantalla

Del mundo a la pantalla



¿Qué tareas se llevan a cabo dentro de cada Espacio? ¿Qué transformaciones? ¿Cuáles son las transformaciones que permiten pasar de un espacio al otro?

Bibliografía

- ACM SIGGRAPH Proceedings
- Angel, Edward, *Interactive Computer Graphics. A top-down approach with OpenGL,* Addison Wesley, 1997, 1rst Edition.
- Foley, J., van Dam, A., Feiner, S. y Hughes, J., *Computer Graphics. Principles and Practice*, Addison Wesley, 1992, 2nd Edition.
- Hearn, D., Baker, M.P., *Computer Graphics, C Version,* Prentice Hall Inc., 1997, 2nd Edition.