CAPITULO 1

Transformaciones Geométricas en dos dimensiones

1.1 Traslación

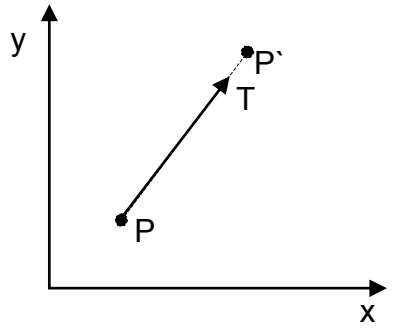
Una traslación se aplica a un objeto para reposicionarlo a lo largo de una línea recta, desde una ubicación de coordenadas a otra.

$$x' = x + t_x \qquad y' = y + t_y$$

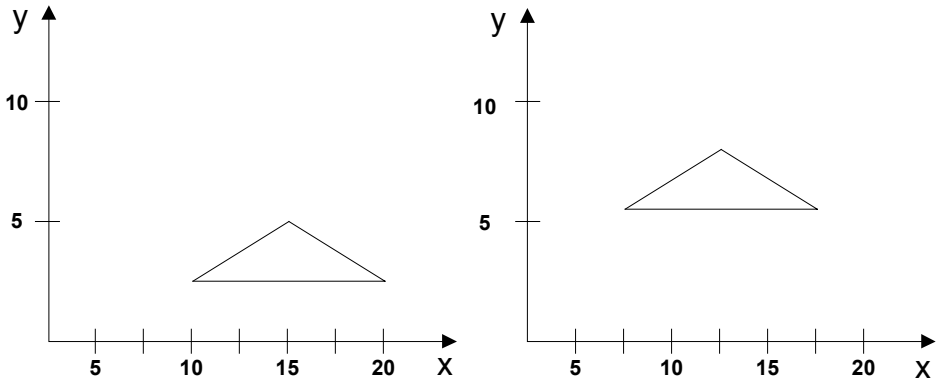
El par (t_x, t_y) se llama vector de traslación.
En forma matricial:

$$P = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad P' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$P' = P + T$$



La traslación es una transformación de cuerpo rígido que mueve al objeto sin deformarlo.



T=(-5.50, 3.75)

Ejemplo Utilizando OpenGl

En Opengl, para la traslación tanto en dos dimensiones(2D) como en tres dimensiones(3D) se utiliza la instrucción *glTranslated,* en sus versiones de parámetros tipo *float, glTranslatef,* como en su versión de parámetros tipo *double, glTranslated.* Sus prototipos son los siguientes:

void glTranslatef(GL float x, GL float y, GL float z)
void glTranslated(GL double x, GL double y, GL double z)

glTranslate mueve el sistema de coordenadas al punto especificado por x,y,z. La matriz activa es multiplicada por la matriz de traslación. Es posible utilizar glPushMatrix y glPopMatrix para guardar y recuperar el sistema de coordenadas.

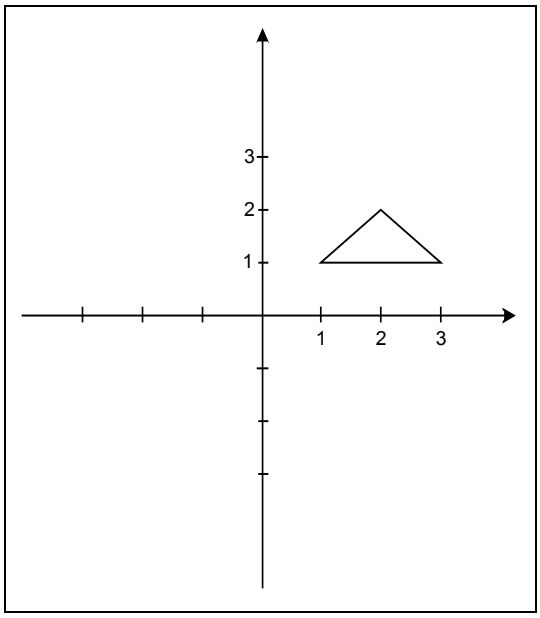


Figura 1.1: Dibujo de un triángulo en 2D

Supongamos que deseamos dibujar un triangulo en la posición indicada en la figura 1.1. Para realizar esto utilizamos el siguiente código.

```
glBegin(GL_TRIANGLES);
glVertex3f(1.0f,1.0f,0.0f);
glVertex3f(2.0f,1.0f,0.0f);
glVertex3f(1.5f,1.5f,0.0f);
glEnd();
```

Al ejecutar nuestro programa obtenemos la salida indicada en la figura 1.2. Se han agregado un par de ejes coordenados solo para referencia.

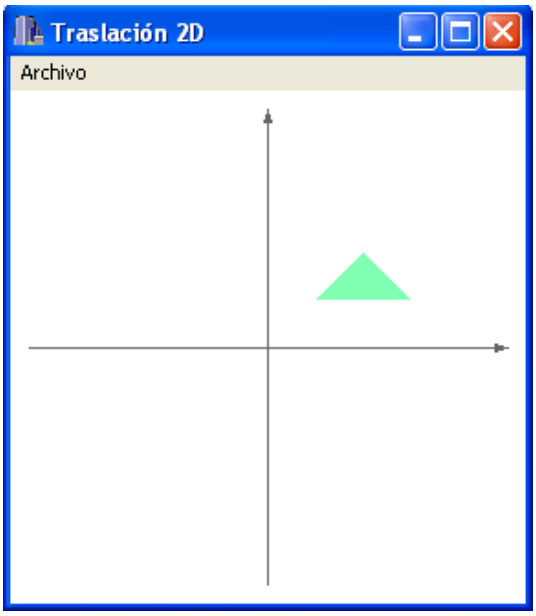


Figura 1.2 : Triángulo en la posición (1,1)

Para trasladar nuestro sistema de coordenadas a cualquier otro punto de nuestro espacio utilizamos la función glTranslate. Agreguemos una traslación a la posición (2,2) antes de nuestro dibujo, esto es:

```
glTranslatef(2.0f,2.0f,2.0f);
glBegin(GL_TRIANGLES);
glVertex3f(1.0f,1.0f,0.0f);
glVertex3f(2.0f,1.0f,0.0f);
glVertex3f(1.5f,1.5f,0.0f);
glEnd();
```

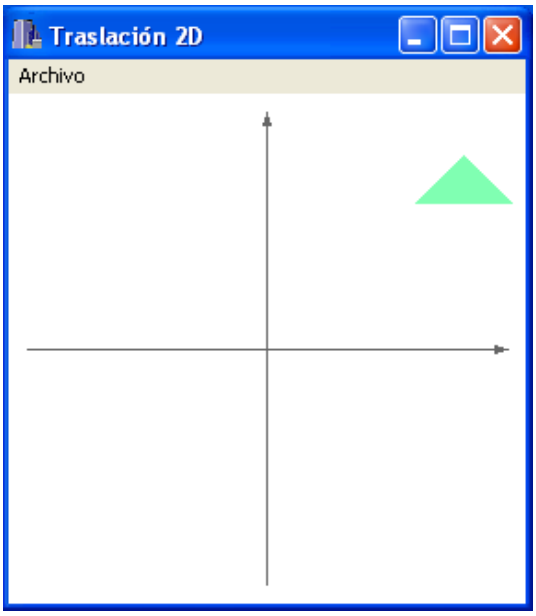


Figura 1.3: Triángulo en la posición (3,3)

La figura obtenida ahora es la mostrada en la figura 1.3, en ella notamos que nuestro triangulo se ha desplazado a la posición (3,3) esto es ya que él triángulo se dibujaba originalmente en la posición (1,1) más la traslación

efectuada a la posición (2,2). Debemos recordar que nuestro sistema esta ubicado a hora en la posición (2,2) luego si efectuamos posterior a esta traslación otros dibujos están quedarán referidos a la posición (2,2), esto puede causar problemas para el dibujo de varios objetos en nuestra imagen, supongamos que ahora queremos dibujar otro triangulo en la posición original (-3,-3). Si utilizamos el mismo código, nuestro resultado seria el de la figura 1.4.

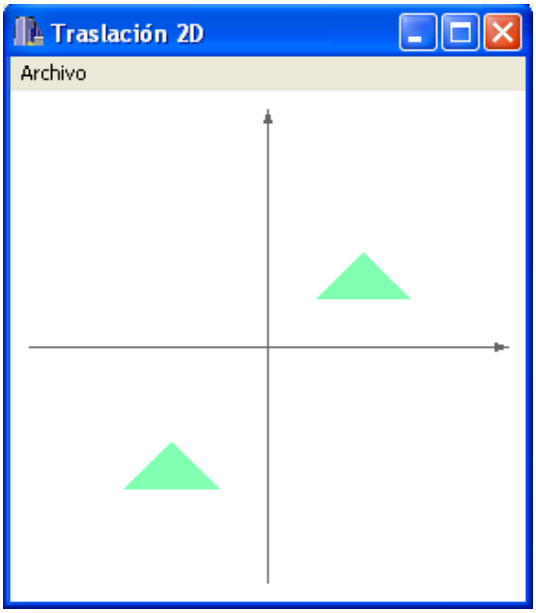


Figura 1.4 : Triángulo en la posición (-3,-3)

Como podemos ver en la figura 1.4, el resultado final no es el resultado deseado, esto es porque nuestra figura original estaba referenciada a nuestra coordenadas originales, para solucionar este problema y poder conservar las coordenadas del modelo podemos utilizar ahora las funciones glPushMatrix y glPopMatrix para guardar y recuperar nuestro sistema de coordenadas. Nuestro código debería ser el siguiente;.

```
glPushMatrix();
glTranslatef(2.0f,2.0f,2.0f);
glBegin(GL_TRIANGLES);
        glVertex2f(1.0f,1.0f);
        glVertex2f(3.0f,1.0f);
        glVertex2f(2.0f,2.0f);
 glEnd();
 glPopMatrix();
 glBegin(GL_TRIANGLES);
        glVertex2f(-3.0f,-3.0f);
        glVertex2f(-1.0f,-3.0f);
        glVertex2f(-2.0f,-2.0f);
 glEnd();
```

El resultado final es el deseado originalmente y se muestra en la figura 1.5 ya que nos hemos asegurado de guardar el sistema de coordenadas antes de dibujar el primer triangulo.

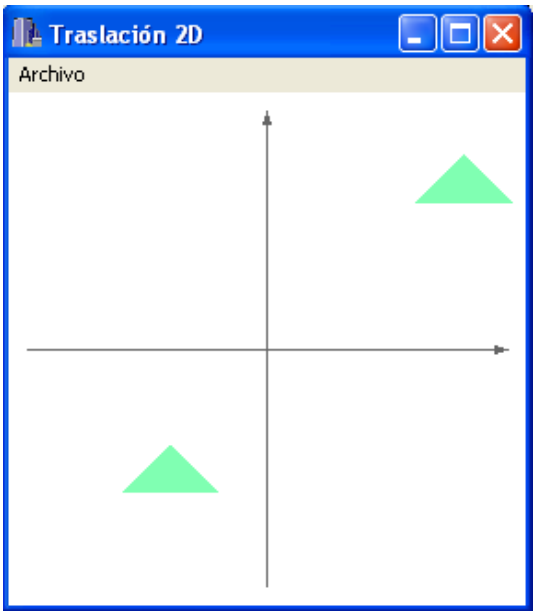
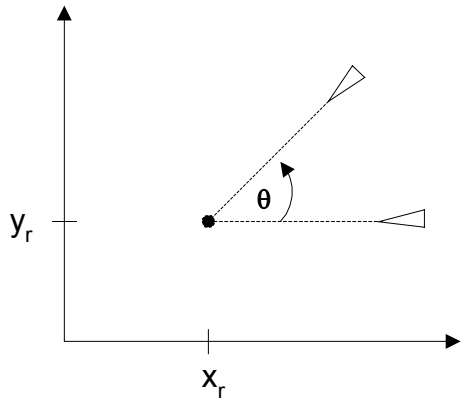


Figura 1.5 : Triangulo en la posición (-3,-3)

1.2 Rotación

La rotación se aplica a un objeto para reposicionarlo a lo largo de un trayecto circular en el plano *xy*.

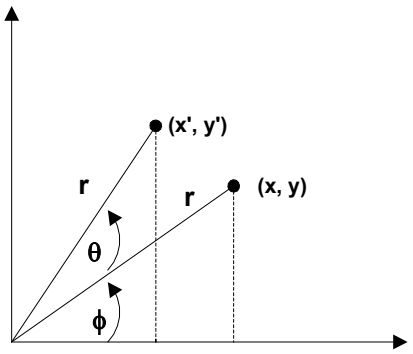
Para generar una rotación se especifica un ángulo de rotación θ y la posición (x_r, y_r) del punto de rotación (punto pivote) acerca del cual el objeto se rota.



Valores positivos del ángulo de rotación definen una rotación en contra de las agujas del reloj y lo contrario para un ángulo negativo.

Esta transformación se puede describir como una rotación alrededor de un eje perpendicular al plano *xy*, y que pasa por el punto pivote.

Primero se determinan las ecuaciones de transformación para la rotación de un punto *P* alrededor del origen del eje de coordenadas.



$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - rsen\phi sen\theta$$
$$y' = rsen(\phi + \theta) = r\cos\phi sen\theta + rsen\phi\cos\theta$$
$$x = r\cos\phi$$
$$y = rsen\phi$$

Substituyendo:

$$x' = x\cos\theta - ysen\theta$$
$$y' = rsen\theta + y\cos\theta$$

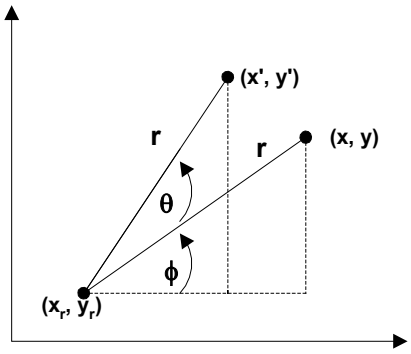
En representación matricial

$$P' = R * P$$

Donde:

$$R = \begin{bmatrix} \cos\theta & -sen\theta \\ sen\theta & \cos\theta \end{bmatrix}$$

En el caso dela rotación alrededor de un pivote las ecuaciones son:



$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)sen\theta$$
$$y' = y_r + (x - x_r)sen\theta + (y - y_r)\cos\theta$$

Las rotaciones son también transformaciones de cuerpos rígidos que mueven cuerpos sin producir deformaciones.

Una línea recta se rota, aplicando las transformaciones a los puntos de término de la línea y generando de nuevo la línea. Para un polígono las ecuaciones se aplican a los vértices y después se

regenera el peligro con los nuevos vértices. Una curva se rota aplicando las ecuaciones a los puntos de control de ésta y después regenerando la curva de acuerdo a los nuevos puntos de control.

Ejemplo Utilizando OpenGl

En *Opengl,* la rotación se trabaja con la instrucción *glRotate,* en sus versiones de parámetros tipo *float, glRotatef,* como en su versión de parámetros tipo *double, glRotated.* Sus prototipos son los siguientes:

void glRotated(GLdouble angulo, GLdouble x, GLdouble y, GLdouble z)
void glRotatef(GLfloat angulo, GLfloat x, GLfloat y, GLfloat z)

glRotate rota el sistema de coordenadas en el sentido de las manecillas del reloj en el Angulo especificado en la dirección del vector representado por las coordenadas x,y,z. La matriz activa es multiplicada por la matriz de rotación. Al igual que en el caso de translación es posible utilizar glPushMatrix y glPopMatrix para guardar y recuperar el sistema de coordenadas.

Supongamos que deseamos rotar él triangulo del capitulo anterior como en la posición indicada en la figura 2.1. Esta rotación corresponde a una rotación de 45 grados en torno al vector (0,0,1). Una vez rotado nuestro sistema para dibujar él triangulo debemos considerar la nueva ubicación del sistema de coordenadas del modelo, luego en base estas nuevas coordenadas realizamos nuestro dibujo. Nuestro sistema de coordenadas del modelo esta rotado 45 grados luego necesitamos calcular las coordenadas del triangulo para la posición deseada.

```
glRotatef(45,0,0,1);
glColor3f(0.5f, 1.0f, 0.7f);
glBegin(GL_TRIANGLES);
      glVertex2f(1.7f,0.0f);
      glVertex2f(3.7f,0.0f);
      glVertex2f(2.6f,1.0f);
glEnd();
```

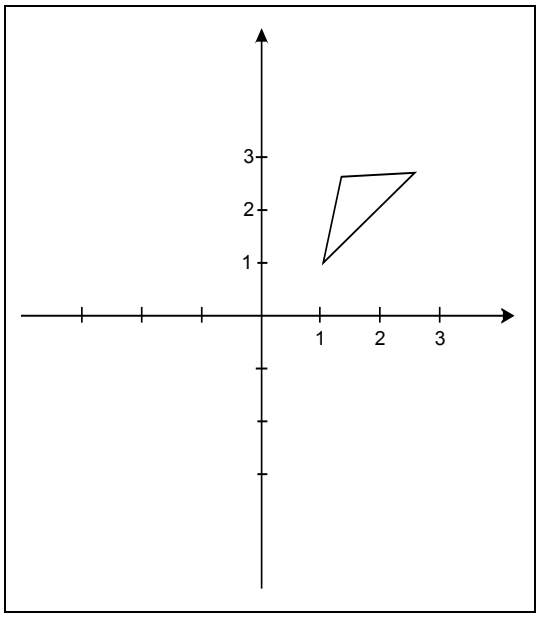


Figura2.1: Rotación deseada de un triangulo

Al ejecutar nuestro programa obtenemos la salida indicada en la figura 2.2. Nuevamente se han agregado un par de ejes coordenados para referencia.

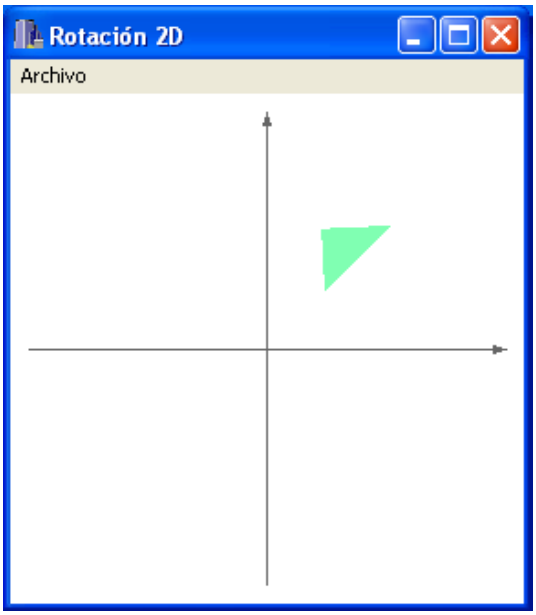


Figura 2.2 : Rotación del triangulo en OpenGl

Si deseamos efectuar una animación de una escena,.en este caso 2D, debemos dibujar los objetos en cada frame de la escena. Supongamos que queremos hacer rotar nuestro triangulo en forma dinámica en el plano xy,. Necesitamos en cada ejecución de nuestra función de dibujo: borrar la escena, rotar nuestro sistema de coordenadas, dibujar el triangulo. Esta función de dibujo debe ejecutarse cada cierto tiempo, esto puede lograrse

mediante la función timer. El siguiente ejemplo muestra el código necesario para esta sencilla animación:

```
Timer()
{
  rotación++;
  repaint();
}
Dibujo()
{
      static int rotación=0;
      glRotatef(45,0,0,1);
      glColor3f(0.5f, 1.0f, 0.7f);
      glBegin(GL_TRIANGLES);
            glVertex2f(1.7f,0.0f);
            glVertex2f(3.7f,0.0f);
            glVertex2f(2.6f,1.0f);
      glEnd();
    rotación++;
}
```

1.3 Escalamiento

Un escalonamiento es una transformación que modifica el tamaño del objeto.

Por ejemplo, para un polígono se aplican los factores de escalamiento a cada vértice para generar los vértices transformados.

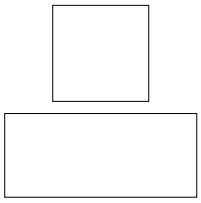
$$x' = x * s_x \qquad y' = y * s_y$$

S_x escala en la dirección **x** y S_y escala en la dirección **y**.

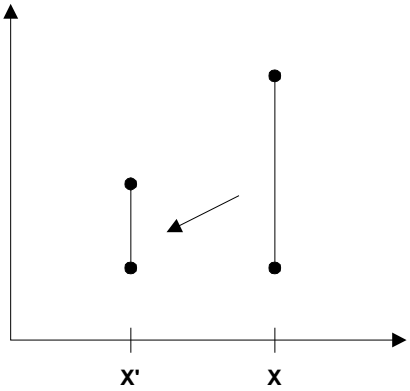
En forma matricial:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

S_x, S_y pueden ser cualquier número positivo, valores <1 reducen el tamaño del objeto.



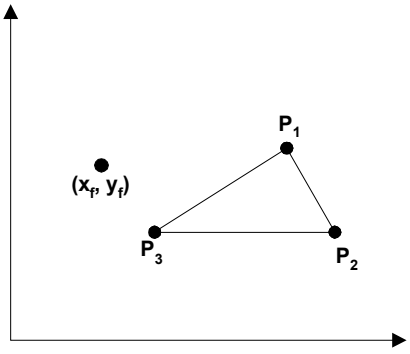
Valores distintos de S_x y S_y cambian el tamaño y forma del cuerpo. Al Aplicar S_x y S_y a un cuerpo, éste es escalado y reposicionado. Por esto, un cuerpo escalado debe posteriormente ser trasladado a su lugar de origen.



Una línea escalada con S_x=0.5 y S_y=0.5 se cambia de tamaño y se traslada en x.

Se puede controlar el lugar de un objeto escalando eligiendo una posición llamado punto fijo, que permanece inalterable durante la transformación. El punto fijo se puede elegir como un vértice, el centro de un objeto, o cualquier otra posición.

Un polígono se escala, entonces, relativo al punto fijo, escalando la diferencia entre el vértice y el punto fijo.



Las ecuaciones en este caso son:

$$x' = x_f + (x - x_f) * s_x$$
$$y' = y_f + (y - y_f) * s_y$$

ó:

$$x' = x * s_x + x_f(1 - s_x)$$
$$y = y * s_y + y_f(1 - s_y)$$

Ejemplo Utilizando OpenGl

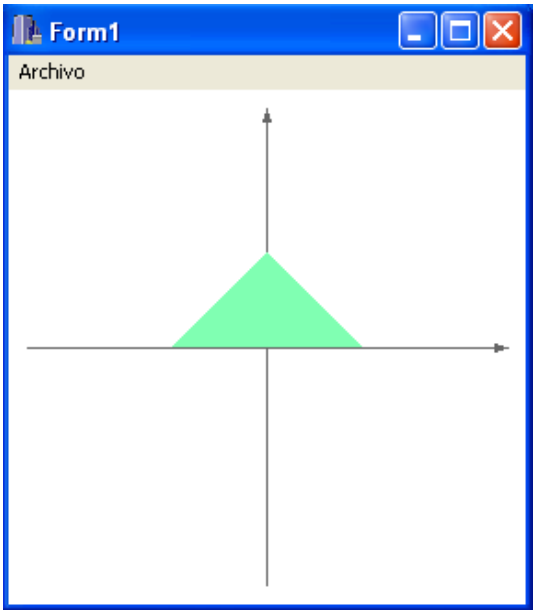
En *Opengl, e*l escalamiento se trabaja con la instrucción *glScale,* en sus versiones de parámetros tipo *float, glScalef,* como en su versión de parámetros tipo *double, glScaled.* Sus prototipos son los siguientes:

void glScaled(GLdouble x, GLdouble y, GLdouble z)
void glScalef(GLfloat x, GLfloat y, GLfloat z)

glScale escala el sistema de coordenadas en los factores indicados por x,y y z. La matriz activa es multiplicada por la matriz de escalamiento. Al igual que en el caso de translación es posible utilizar glPushMatrix y glPopMatrix para guardar y recuperar el sistema de coordenadas.
Supongamos que deseamos escalar nuestro triangulo del capitulo anterior en un factor 2 en todas las direcciones, entonces nuestro código sería el siguiente:

```
glScalef(2.0,2.0,2.0);
glColor3f(0.5f, 1.0f, 0.7f);
glBegin(GL_TRIANGLES);
      glVertex2f(-1.0f,0.0f);
      glVertex2f(1.0f,0.0f);
      glVertex2f(0.0f,1.0f);
glEnd();
```

Al ejecutar nuestro programa obtenemos la salida indicada en la figura 3.2.



1.4 Coordenadas Homogéneas

Hemos visto que en el caso de dos dimensiones, las ecuaciones de transformación son en el caso de traslación la suma de un vector, y en el de rotación y escalamiento, multiplicación de matriz. Con el fin de uniformar las operaciones matemáticas se introduce una tercera coordenada *h*, homogénea dando el triple (x_h, y_h, h)para un punto (x,y), donde:

$$x = \frac{x_h}{h}$$

$$y = \frac{y_h}{h}$$

Generalmente se elige h=1 y entonces un punto cartesiano queda representado por el triple (x, y, 1)

Luego, las ecuaciones de transformaciones se convierten en:

Traslación:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotación:

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

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

Escalamiento:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = S(s_x, s_y) * P$$
$$S^{-1}S(s_x, s_y) = S(1/s_x, 1/s_y)$$

1.5 Transformaciones Compuestas

En la representación matricial anterior, se puede determinar una matriz para una secuencia de transformaciones, y que se llama matriz de Transformación Compuesta.

Traslación:

$$\begin{aligned} P' &= T(tx_2, ty_2) * \{T(tx_1, ty_1) * P\} \\ &= \{T(tx_2, ty_2) * T(tx_1, ty_1)\} * P \end{aligned}$$

ó:

$$\begin{bmatrix} 1 & 0 & tx_2 \\ 0 & 1 & ty_2 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & tx_1 \\ 0 & 1 & ty_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx_1 + tx_2 \\ 0 & 1 & ty_1 + ty_2 \\ 0 & 0 & 1 \end{bmatrix}$$

ó:

$$T(tx_2, ty_2) * T(tx_1, ty_1) = T(tx_1 + tx_2, ty_1 + ty_2)$$

Lo que demuestra que las dos traslaciones sucesivas son aditivas.

Rotaciones:

$$P' = R(\theta_2) * \{R(\theta_1) * P\}$$
$$P' = \{R(\theta_2) * R(\theta_1)\} * P$$

y

$$R(\theta_2) * R(\theta_1) = R(\theta_2 + \theta_1)$$

lo que demuestra que dos rotaciones sucesivas son aditivas.

$$P' = R(\theta_2 + \theta_1) * P$$

Escalamiento:

$$\begin{bmatrix} Sx_2 & 0 & 0 \\ 0 & Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} Sx_1 & 0 & 0 \\ 0 & Sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} Sx_1 * Sx_2 & 0 & 0 \\ 0 & Sy_1 * Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ó

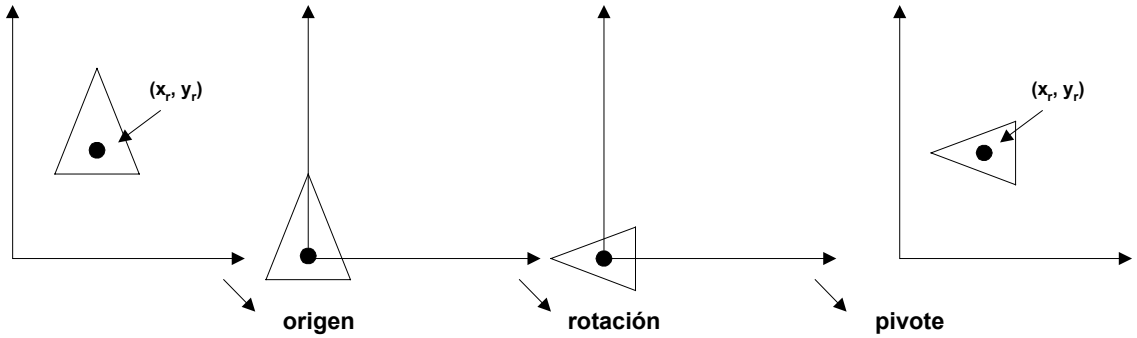
$$S(Sx_2, Sy_2) * S(Sx_1, Sy_1) = S(Sx_1 * Sx_2, Sy_1 * Sy_2)$$

Lo que indica que los escalamientos sucesivos son multiplicativos.

1.5.1 Rotación alrededor de un punto pivote

Algunos paquetes gráficos traen primitivas de rotación sólo para el origen del eje de coordenadas. En estos casos, para rotar alrededor de un pivote (x_r, y_r) se deben hacer los siguientes pasos.

1. Trasladar el objeto desde el pivote al origen del eje de coordenadas.
2. Rotar el Objeto
3. Trasladar el objeto al punto pivote.



La matriz compuesta está dad por:

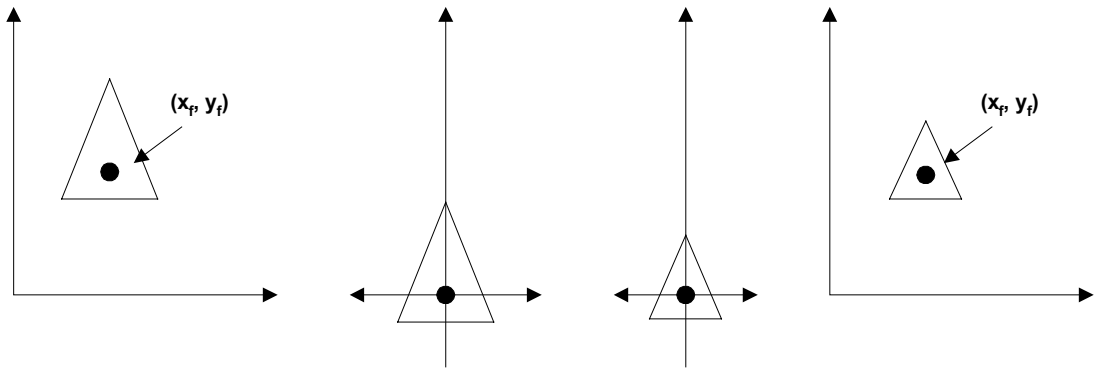
$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -sen\theta & 0 \\ sen\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -sen\theta & x_r(1-\cos\theta)+y_r sen\theta \\ sen\theta & \cos\theta & y_r(1-\cos\theta)-x_r sen\theta \\ 0 & 0 & 1 \end{bmatrix}$$

que puede ser expresada por:

$$T(x_r, y_r) * R(\theta) * T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

1.5.2 Escalamiento para un punto general

La figura muestra un escalamiento respecto a un punto fijo (*xf, yf*).



1. Trasladar el objeto de tal manera que el punto fijo coincida con el origen del sistema de coordenadas.
2. Escalar el objeto con respecto al origen.
3. Trasladar al punto fijo.

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & x_f(1-Sx) \\ 0 & Sy & y_f(1-Sy) \\ 0 & 0 & 1 \end{bmatrix}$$

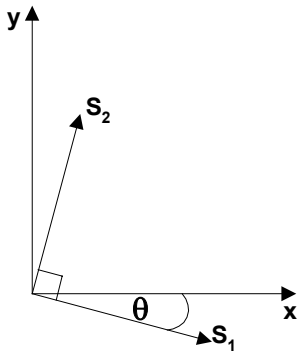
ó

$$T(x_f, y_f) * S(Sx, Sy) * T(-x_f, -y_f) = S(x_f, y_f, Sx, Sy)$$

1.5.3 Escalamiento en cualquier dirección

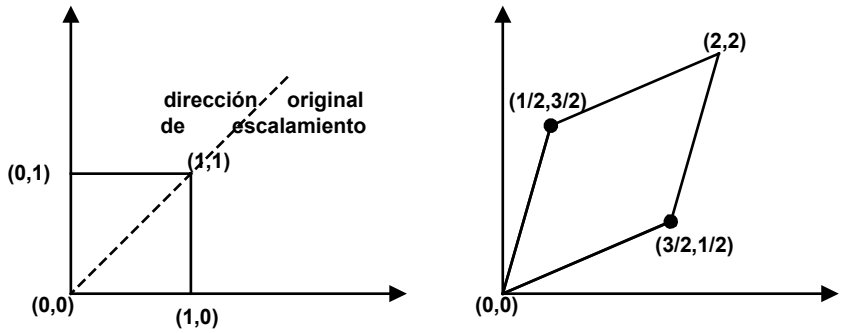
Los parámetros *Sx* y *Sy* escalan un objeto en las direcciones x e *y* respectivamente.

Se puede escalar un objeto en otra dirección, rotando el objeto y alienándolo con el sistema de coordenadas y luego rotándolo en sentido inverso.



$$R^{-1}(\theta) * S(S_1, S_2) * R(\theta) = \begin{bmatrix} S_1 \cos^2\theta + S_2 sen^2\theta & (s2 - S1)\cos\theta * sen\theta & 0 \\ (S_2 - S_1)\cos\theta * sen\theta & S_1 sen^2\theta + S_2 \cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ejemplo:



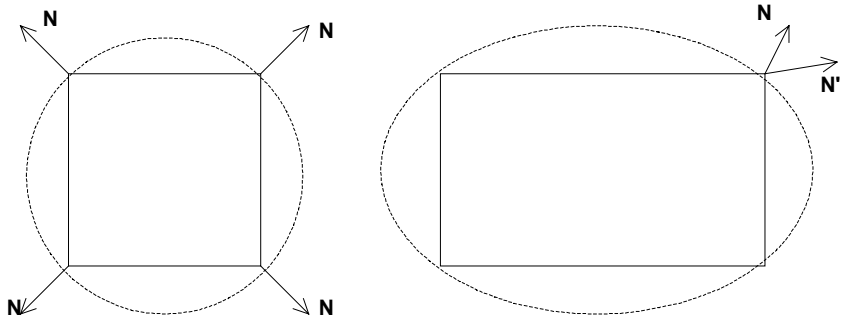
Con S_1=1 S_2=2 θ =45º

1.6 Transformación de una normal

En computación gráfica la normal a una curva, un plano o una superficie, es de gran utilidad para variadas operaciones. La pregunta que surge entonces es ¿Qué operaciones de transformación dejan invariante la normal (vector normal)?

Rápidamente se deduce que la traslación no afecta a la normal, ya que ésta es similar a un cambio de coordenadas.

Sin embargo en el caso del escalamiento:



La normal no es invariante al proceso de escalamiento. ¿Cómo se encuentra *N'*?

Una tangente en un plano es la diferencia entre dos puntos que están en el plano, esto es:

$$t = P_1 - P_2$$

Si *P1* y *P2* están en coordenadas homogéneas, la coordenada homogénea se elimina al calcular la tangente.

Sin embargo, siempre se debe cumplir que:

$$t * N = t'*N' = 0 \quad \text{en todos los casos sea}$$

$$t' = t * M$$

Donde *M* es matriz de transformación cualquiera.

Luego

$$t * N = t'*N' = tMM^{-1}N^T = (t * M)(N * M^{-1^T})$$

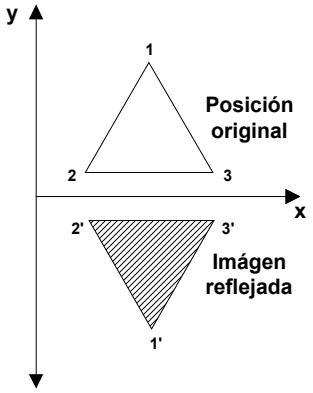
luego

$$N' = N * M^{-1^T} \quad \text{(caso general)}$$

1.7 Otras transformaciones

1.7.1 Reflexión

Una reflexión es una transformación que produce una imagen de espejo de un objeto.



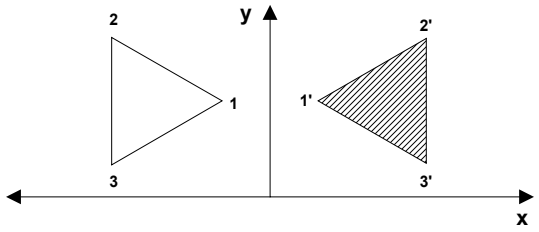
Una reflexión acerca la línea *y*=0, está dada por la siguiente matriz

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

Esto se puede conseguir con una rotación en 180°, alrededor del eje x.

Acerca del eje *y*

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



1.7.2 Shear

Una transformación que distorsiona la forma de un objeto se llama shear.

Shear en la dirección x se obtiene con la matriz

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

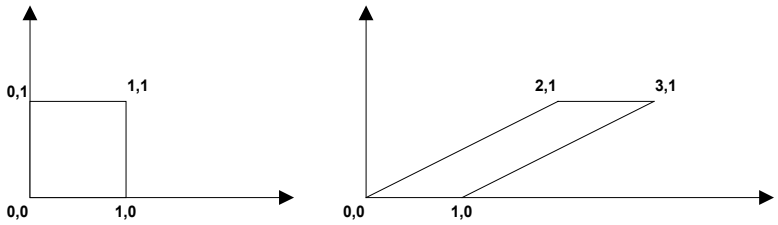
donde

$x' = x + Sh_x * y$

$y' = y$

Sh_x es cualquier número real

Ejemplo: con Sh_x=2



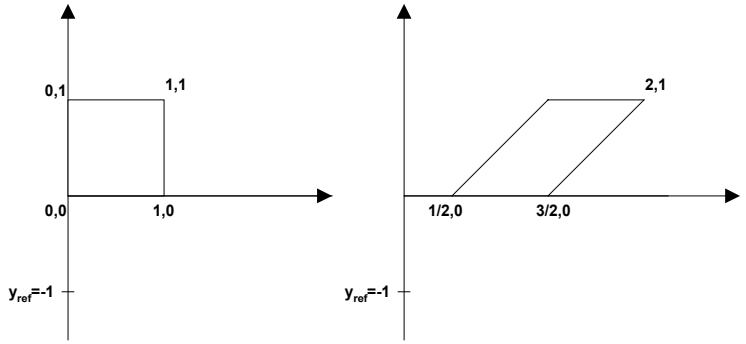
Shear relativa a una línea de referencia

$$\begin{bmatrix} 1 & Sh_x & -Sh_x * y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = x + Sh_x(y - y_{ref})$$

$$y' = y$$

Ejemplo: y_{ref}=-1



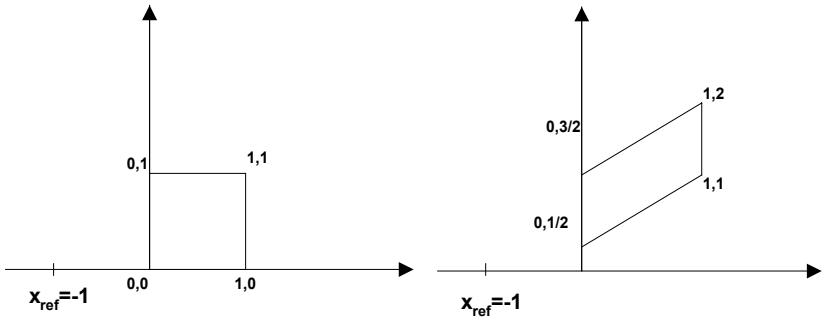
Shear es la dirección y relativo a una línea x=x_{ref}

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y * x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = x$$

$$y' = Sh_y(x - x_{ref}) + y$$

Ejemplo: Sh_y=1/2 x_{ref}=-1

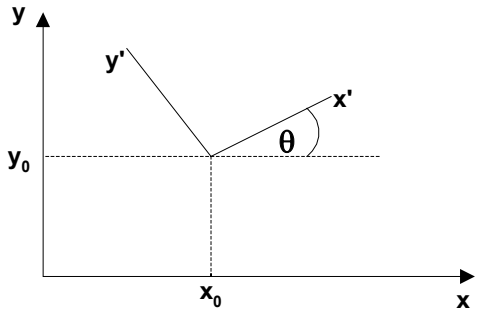


1.8 Transformaciones entre sistemas de coordenadas

A menudo se requiere la transformación desde un sistema de coordenadas a otro.

De hecho, los objetos en coordenadas globales deben ser transformados a coordenadas de cámara.

Sean los siguientes sistemas de coordenadas:



Para transformar un objeto desde coordenadas *xy* a *x'y'*, deben efectuarse los siguientes pasos:

1. Mover el origen del sistema *x'y'* al origen del sistema *xy*.
2. Rotar de tal manera que coincida x' con x.

Matricialmente esto se expresa como:

$$T(-x_0,-y_0)=\begin{bmatrix}1 & 0 & -x_0\\ 0 & 1 & -y_0\\ 0 & 0 & 1\end{bmatrix}$$

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

Concatenando se obtiene:

$$M_{xy,x'y'}=R(-\theta)*T(-x_0,-y_0)$$

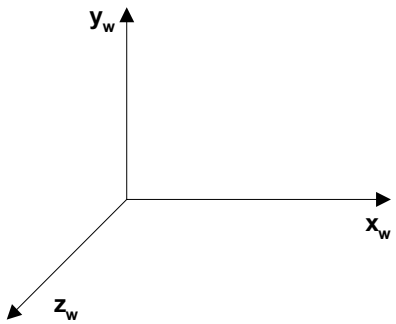
1.8.1 Sistemas de coordenadas

En general, en computación gráfica se usan los siguientes sistemas de coordenadas referenciales:

- Sistema de coordenadas globales.
- Sistema de coordenadas de modelación de un objeto.
- Sistema de coordenadas de cámara (visual).
- Sistema de coordenadas de pantalla o dispositivo.

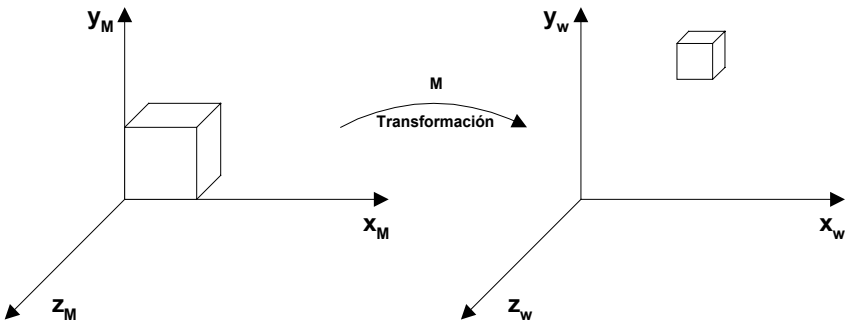
1.8.1.1 Sistema de coordenadas globales

En este sistema se representan todos los objetos de una escena. Generalmente se denomina por la letra *W*, esto es [x_w, y_w, z_w] representa un punto en el sistema de coordenadas.



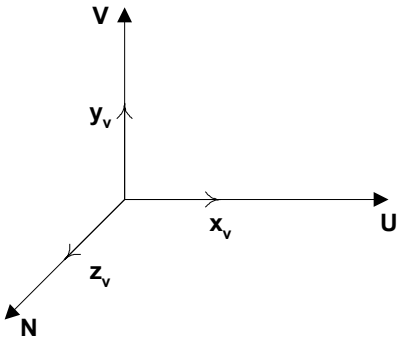
1.8.1.2 Sistema de coordenadas de modelación

Corresponde, en general, a las coordenadas en las cuales se modela un objeto de la escena. El objeto una vez modelado debe ser transformado al sistema de coordenadas globales.



1.8.1.3 Sistema de coordenadas de cámara (Visual)

Corresponde al sistema de coordenadas en el cual se visualiza un objeto, esto es, desde el punto de vista del observador. Por convención, usa los siguientes vectores:



Donde los vectores *U*, *V* crean el plano donde se proyecta el objeto desde la escena y *N* representa la dirección de la proyección, o la dirección donde se dirige la vista hacia la escena.

$$P_v = P_w * M_{w,v}$$
$$M_{w,v} = Matriz\ \det ransformación$$

1.8.1.4 Vista en dos Dimensiones

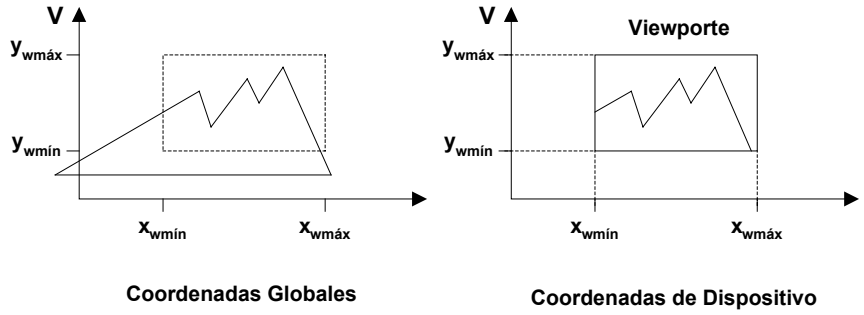
Un área que se selecciona para desplegar en coordenadas globales, se llama ventana (window).

Un área sobre la pantalla, sobre la cual se mapea la ventana, se llama "viewport".

La ventana define qué se va a ver; el viewport define dónde se va a desplegar.

A menudo, tanto la ventana como el viewport, es un rectángulo. Pueden usarse otras formas, pero son más difíciles de procesar.

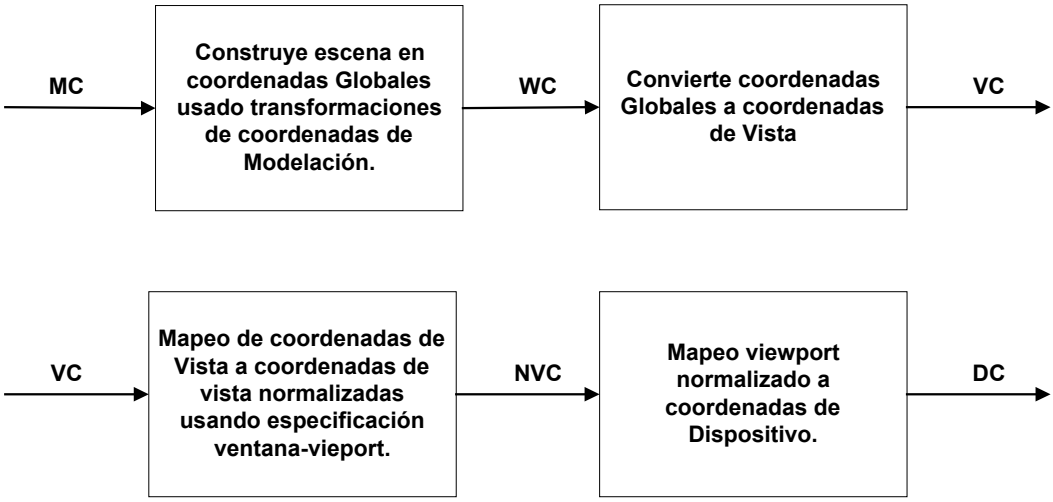
Usualmente, una transformación de vista se refiere como una transformación ventana-a-viewort.

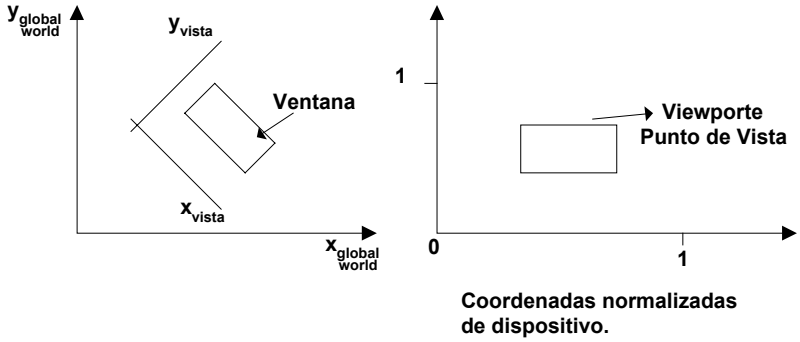


En general, la ventana puede tener cualquier orientación. En este caso, la transformación de vista se realiza en varios pasos.

Primero se construye la escena en coordenadas globales. Después, para obtener una orientación específica, se define un sistema de coordenadas de vista en el plano de coordenadas globales y se define una ventana en el sistema de coordenadas de vista. El sistema de coordenadas de vista se utiliza como un marco de referencia para definir orientaciones arbitrarias de una ventana.

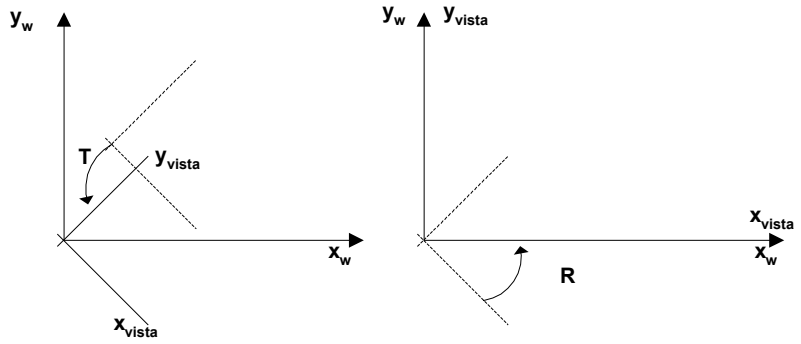
Una vez que se ha establecido el marco de referencia de vista, se pueden transformar objetos en coordenadas globales a coordenadas de vista. Entonces se define un viewport en coordenadas normalizadas (0 a 1) y se mapea el objeto en coordenadas de vista a coordenadas normalizadas.





1.8.1.5 Marco de referencia de coordenadas de vista

Este sistema de coordenadas proporciona la referencia para especificar la ventana de coordenadas globales.
Para transformar un punto en coordenadas globales a coordenadas de vista se debe hacer lo siguiente:

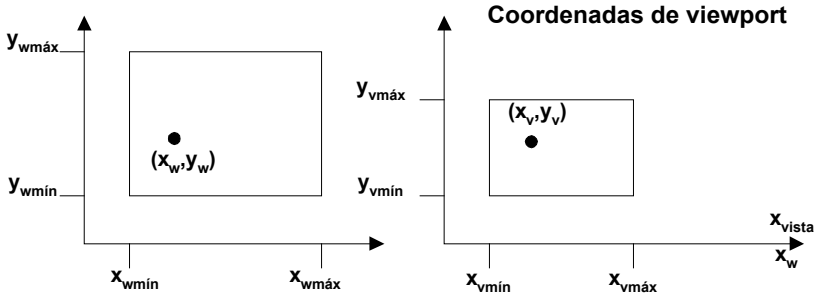


$$M_{wc,vc} = R * T$$

1.8.2 Transformación de coordenadas de ventana a viewport

Una vez que el objeto ha sido transferido al marco de referencia de vista, se elige la ventana en coordenadas de vista y se relaciona el viewport en coordenadas normalizadas.

Esto se hace usando una transformación que mantiene las proporciones de ventana a viewport.



Para mantener posiciones relativas n el viewport se requiere que:

$$\frac{x_v - x_{vmín}}{x_{vmáx} - x_{v\min}} = \frac{x_w - x_{wmín}}{x_{wmáx} - x_{wmín}}$$

$$\frac{y_v - y_{vmín}}{y_{vmáx} - y_{vmín}} = \frac{y_w - y_{w\min}}{y_{wmáx} - y_{wmín}}$$

luego:

$$x_v = x_{vmín} + (x_w - x_{wmín})Sx$$
$$y_v = y_{vmín} + (y_w - y_{wmín})Sy$$

donde:

$$Sx = \frac{x_{vmáx} - x_{vmín}}{x_{wmáx} - x_{wmín}} \qquad Sy = \frac{y_{vmáx} - y_{vmín}}{y_{wmáx} - y_{wmín}}$$

Esto es equivalente a:

1. Realizar una transformación de escala usando un punto fijo de ($x_{wmín}$, $y_{wmín}$) que escala el área de ventana al tamaño del viewport.
2. Trasladar el área de la ventana escalada a la posición del viewport.