COMPUTACION GRAFICA I

(705016O100141)->M

(705015O100141)->E

Prof.: Arq.Iván Burgos, Mg.Sc.

(Ing.Luz-1989)

Abril, 2000

COMPUTACION GRAFICA I

- **BIBLIOGRAFIA** :

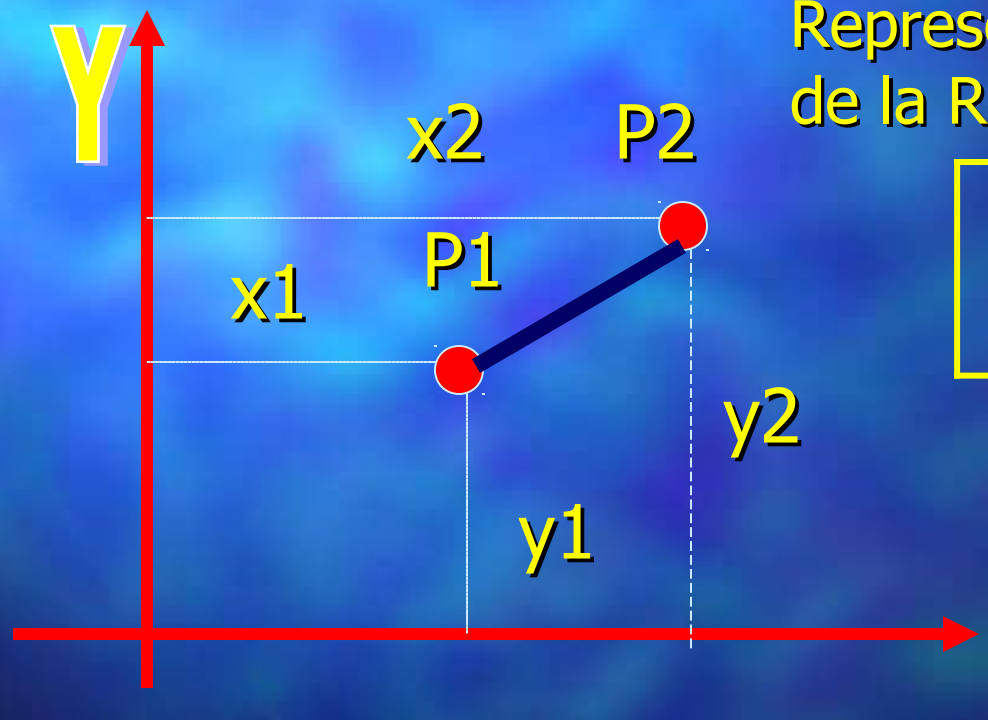
- *Principles of Interactive Computer Graphics*. **W.Neumann & R.Sproull**. Ed.McGraw-Hill, Inc. 1979.
- *Interactive Microcomputer Graphics*. **Chan S. Park**. Ed.Addison Wesley, Inc. 1985.
- *Graficacion Por Computador con Pascal*. **Marc Berger**. Ed.Addison-Wesley Iberoamericana, 1991.
- *Computer Graphics*. **D.Hearn & P.Baker**. Ed.Prentice Hall, 1986.
- *Programming with C++*. **John Hubbard**. Ed.MacGraw Hill, Shaum's Outline Series , 1996.
- *C++*. **Ivor Horton**. Wrox Press Ltd.,1998.
- *Your First C/ C++ Program*. **Alan Neibauer**. Ed.Sybex, Inc. , 1994.
- *C++*. **Ivor Horton**. Wrox Press Ltd.,1998.
- *OpenGL Programming Guide*. **M.Woo, J.Neider & T.Davis**. Ed.Addison-Wesley Developer Press, 1997.
- *OpenGL Programming for Windows 95 & Windows NT*. **Ron Fosner**. Ed.Addison-Wesley Developer Press, 1998
- *Algorithms, Data Structures and problem solving with C++*. **Mark Allen Weiss**. Ed.Addison-Wesley Publisher, 1995.

Representación Gráfica Bidimensional

Representación matricial del Punto P1:

$$\begin{bmatrix} x1 \\ y1 \end{bmatrix}$$

Representación Gráfica Bidimensional

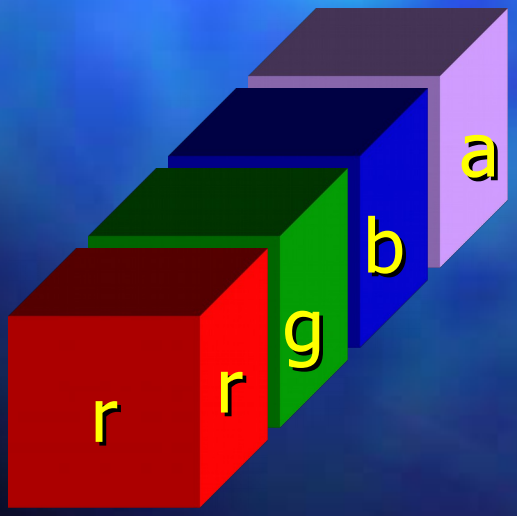


Representación matricial de la Recta P1P2:

$$\begin{bmatrix} x1 & x2 \\ y1 & y2 \end{bmatrix}$$

Periféricos y Dispositivos Gráficos

- Vector Display(Anteriores)
- Raster Display(Actuales)



Cada pixel (punto del monitor), contiene información y puede ser manejado individualmente, de allí que dependiendo del dispositivo cada pixel tiene de 24 a 32 bits de información (8 bits por color+información Alpha)

Periféricos y Dispositivos Gráficos

Al producirse el refrescamiento de la pantalla o monitor (50/60-120 Mhz), el gráfico animado tiende a parpadear por efecto de la vista humana por lo que es necesario poseer a nivel de Hardware, un doble buffer; el cual se alterna evitando de esta manera el mencionado parpadeo.

De manera que el refrescamiento alternado se produce en un Color Map Mode utilizando ambos buffers, el Front Buffer y el Back Buffer.

Matrices y Transformaciones

El Punto o la Recta al ser representados matricialmente, es permitido manipularlo usando operaciones de matrices, la mas utilizada es la multiplicación, por lo que las posiciones de los elementos en el plano ó el espacio pueden ser controladas.

Si tenemos las siguientes dos ecuaciones:

$x' = a*x + b*y$

$y' = c*x + d*y$; éstas tambien pueden ser expresadas de esta forma:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

Para que estas operaciones sean válidas se debe cumplir que $A_{ik}=B_{kj}$ y recordar que estas operaciones no son CONMUTATIVAS.

Matrices y Transformaciones

COORDENADAS HOMOGENEAS:

Todas las operaciones matriciales a excepcion de la traslacion se pueden representar en matrices de 2*2, sin embargo, es recomendable usar matrices de 3*3 ya que ello nos permite combinar diferentes operaciones en una misma matriz, para ello, nos valdremos del recurso matematico de la C.H., que es la representacion tridimensional de un vector bidimensional agregando una fila (o columa adicional) equivalente a uno(1); de la misma manera manera se procede con las matrices contentivas de las transformaciones u operaciones. Recordemos que por analogia todos los problemas en un espacio (N), tienen su correspondiente en un espacio (N+1).

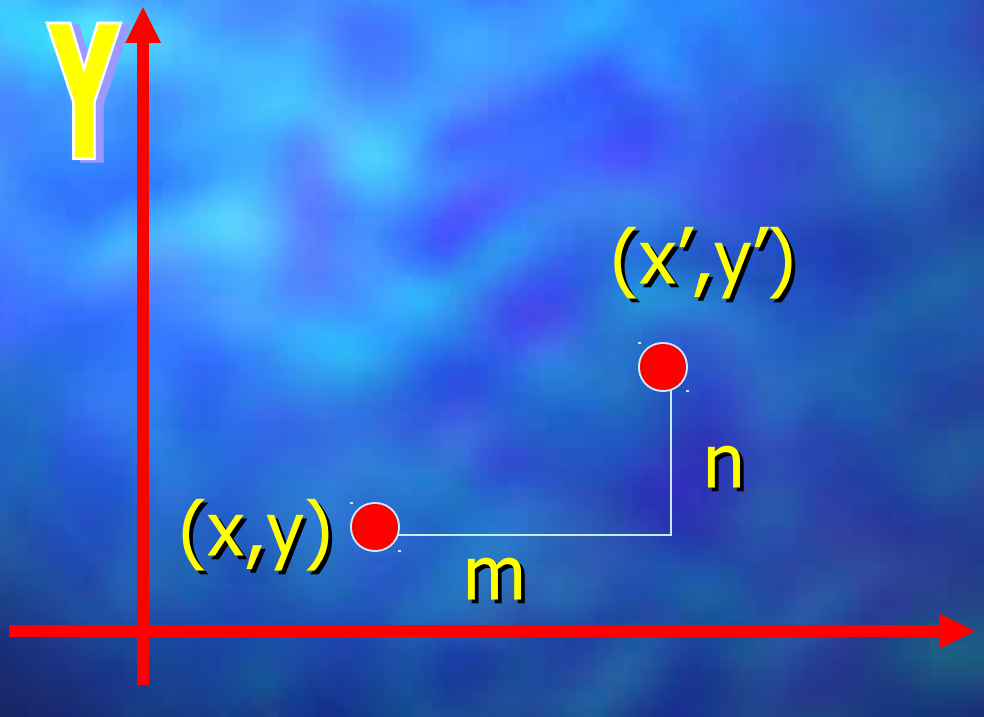
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad ; \quad \text{asi} \quad \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Cumpliendo con una de las condiciones para la validacion de las multiplicaciones matriciales.

Matrices y Transformaciones

Traslacion:

$$x' = x + m$$
$$y' = y + n$$



matriz traslacion:

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

Matrices y Transformaciones

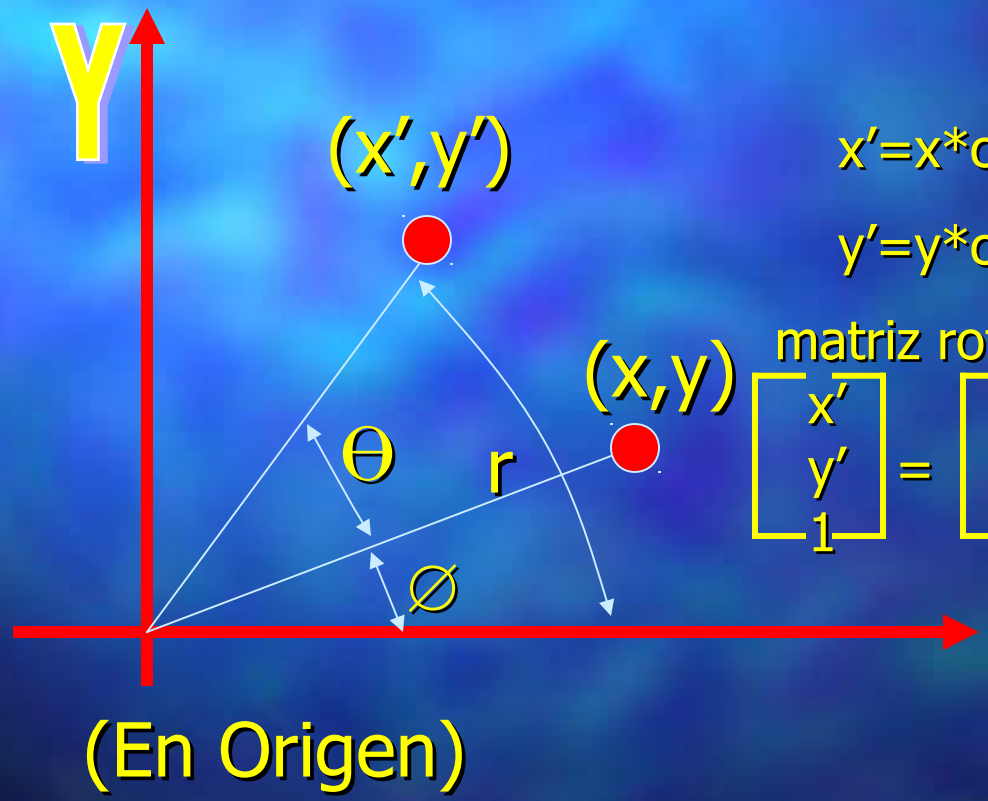
Rotacion:

$x = r*\cos(\varnothing)$

$y = r*\text{sen}(\varnothing)$

$x' = r*\cos(\varnothing + \theta)$

$y' = r*\text{sen}(\varnothing + \theta)$



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

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

matriz rotacion:

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

Matrices y Transformaciones

Rotacion: (En Pivote $\neq$ Origen)

Se requieren tres pasos fundamentales:

1. Trasladar el pivote (xp,yp) al origen.

   $x' = x - m$; $y' = y - n$

2. Rotar los puntos trasladados θ grados alrededor del origen, obteniendo el nuevo punto (x"y")

   $x'' = x' * \cos(\theta) - y' * \sin(\theta)$

   $y'' = y' * \cos(\theta) + x' * \sin(\theta)$; sustituyendo :

   $x'' = (x-m) * \cos(\theta) - (y-n) * \sin(\theta)$

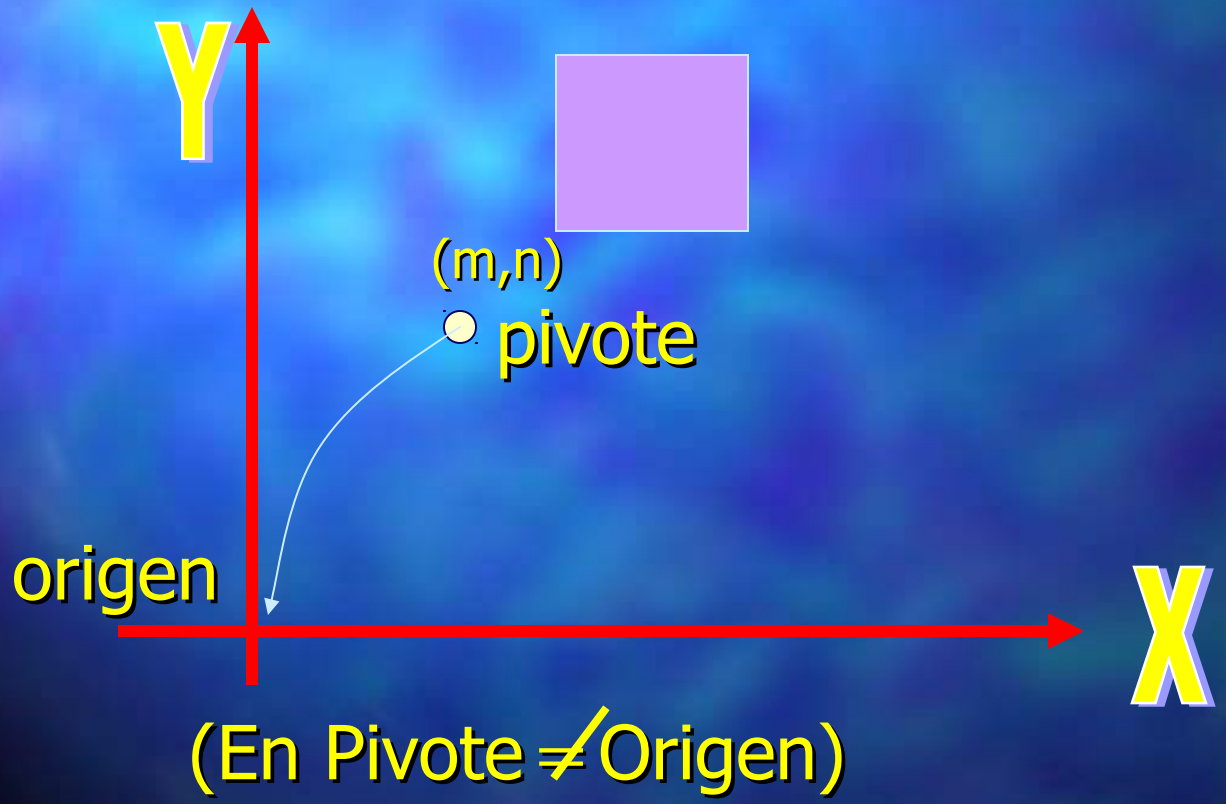
   $y'' = (y-n) * \cos(\theta) + (x-m) * \sin(\theta)$

3. Trasladar inversamente el pivote a su lugar de origen.

   $x''' = x'' + m$; $y''' = y'' + n$

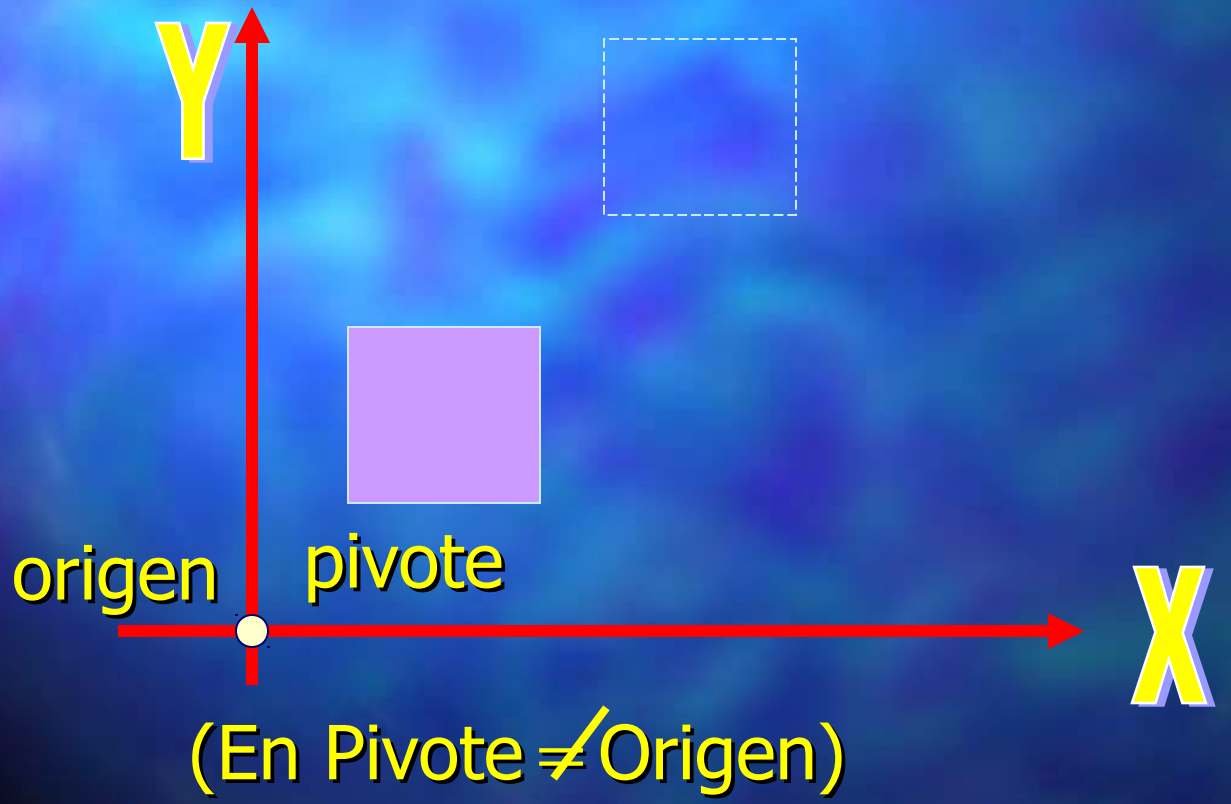
Matrices y Transformaciones

Rotacion: Paso 1a (Traslacion al origen)



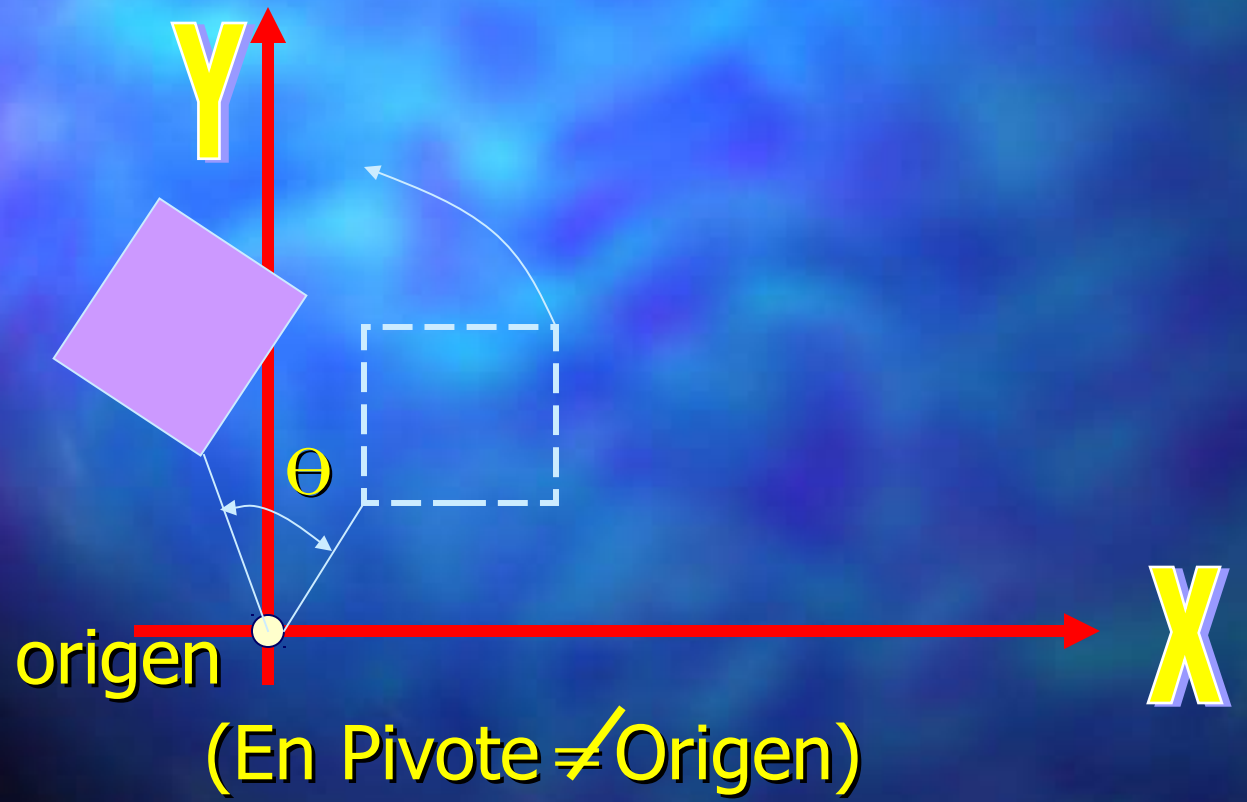
Matrices y Transformaciones

Rotacion: Paso 1b (Traslacion al origen)



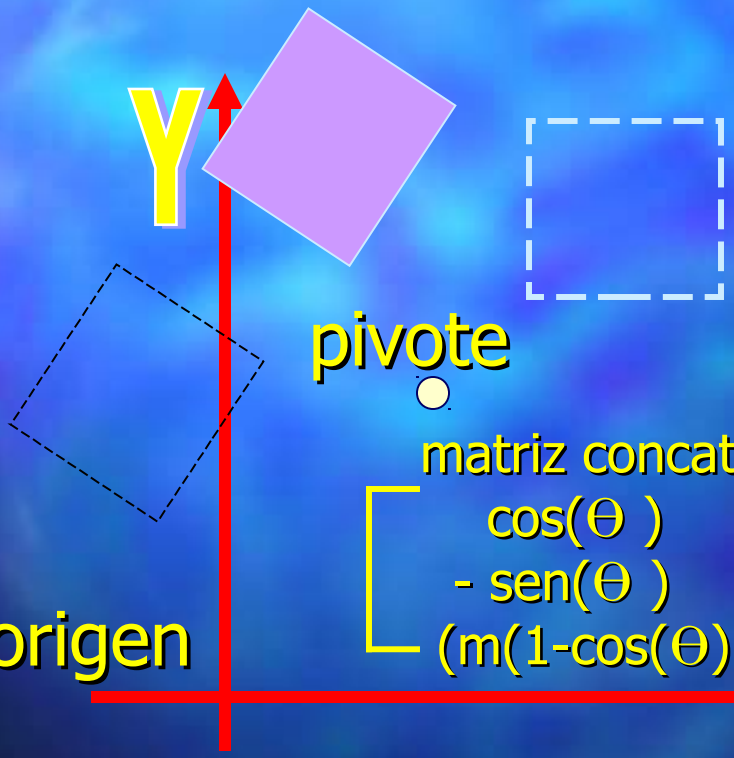
Matrices y Transformaciones

Rotacion: Paso 2 (rotacion alrededor del origen)



Matrices y Transformaciones

Rotacion: Paso 3 (traslacion inversa)



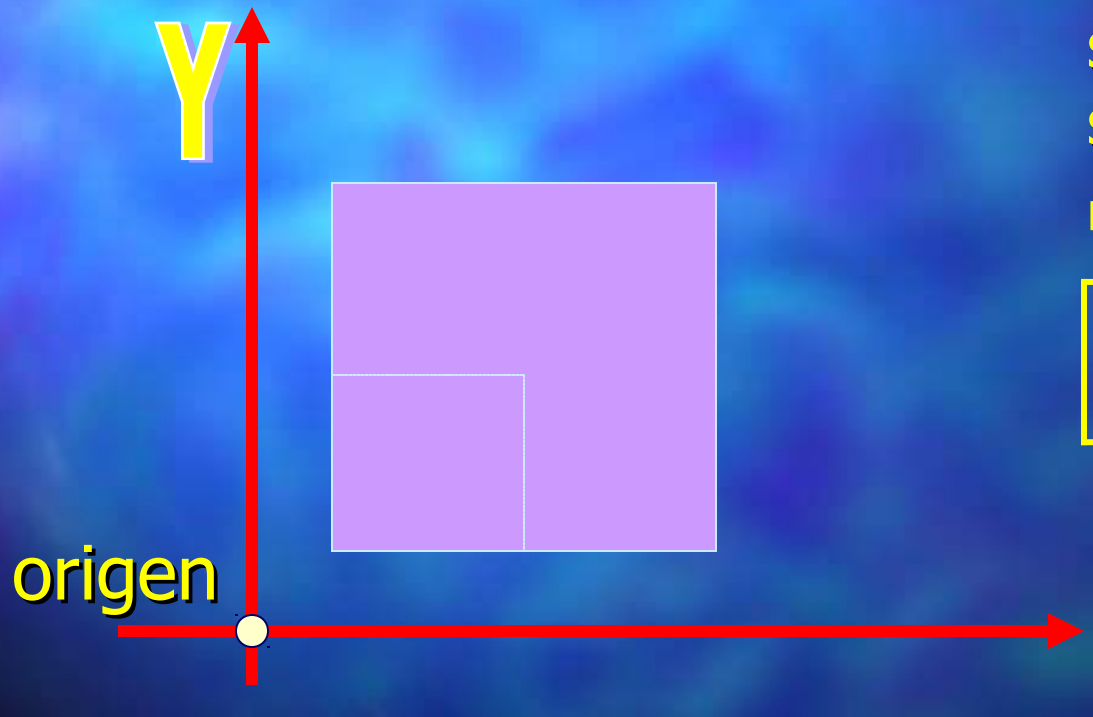
matriz concatenada de rotacion en pivote:

$$\begin{bmatrix} \cos(\theta) & \text{sen}(\theta) & 0 \\ -\text{sen}(\theta) & \cos(\theta) & 0 \\ (m(1-\cos(\theta)+n*\text{sen}(\theta)) & (n(1-\cos(\theta)+m*\text{sen}(\theta)) & 1 \end{bmatrix}$$

(En Pivote $\neq$ Origen)

Matrices y Transformaciones

Escalamiento: (Origen)



$x' = x * Sx$

$y' = y * Sy$

si Sx=Sy>1 aumenta

si Sx=Sy<1 disminuye

matriz escalamiento:

$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrices y Transformaciones

Escalamiento: (En Punto $\neq$ Origen)

Se requieren al igual que la rotacion de tres pasos fundamentales:

1. Trasladar el pivote (xp,yp) al origen.

   $x'=x-m; \quad y'=y-n$

2. Escalar los puntos trasladados segun el factor asignado (Sx y/o Sy), obteniendo el nuevo punto (x"y")

   $x''=x'*Sx$

   $y''=y'*Sy \quad ; \qquad$ sustituyendo :

3. Trasladar inversamente el pivote a su lugar de origen.

   $x'''=x''+m \quad ; \quad y'''=y''+n$

$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ (1-Sx)m & (1-Sy)n & 1 \end{bmatrix}$$ = MATRIZ CONCATENADA

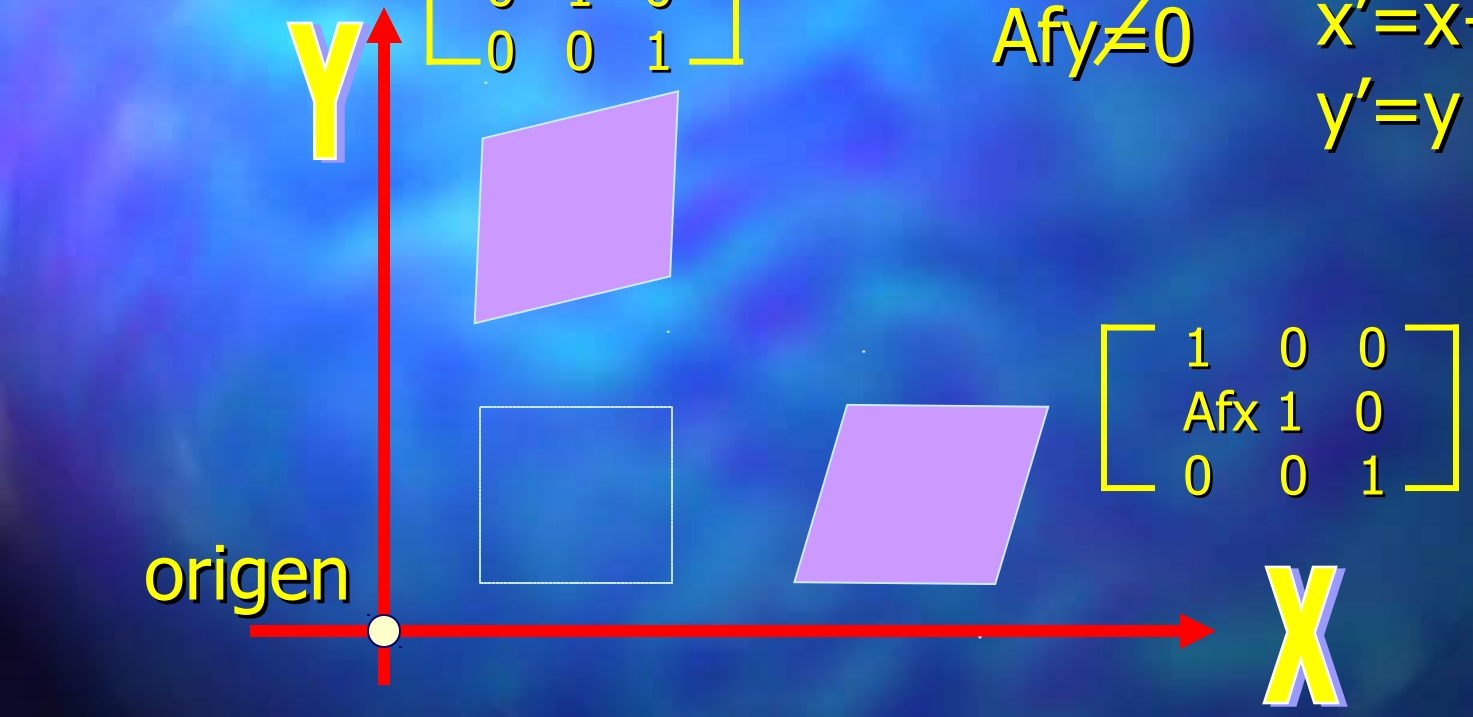
Matrices y Transformaciones

Afilamiento: (distorsion)

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

$Afx \neq 0;$
$Afy \neq 0$

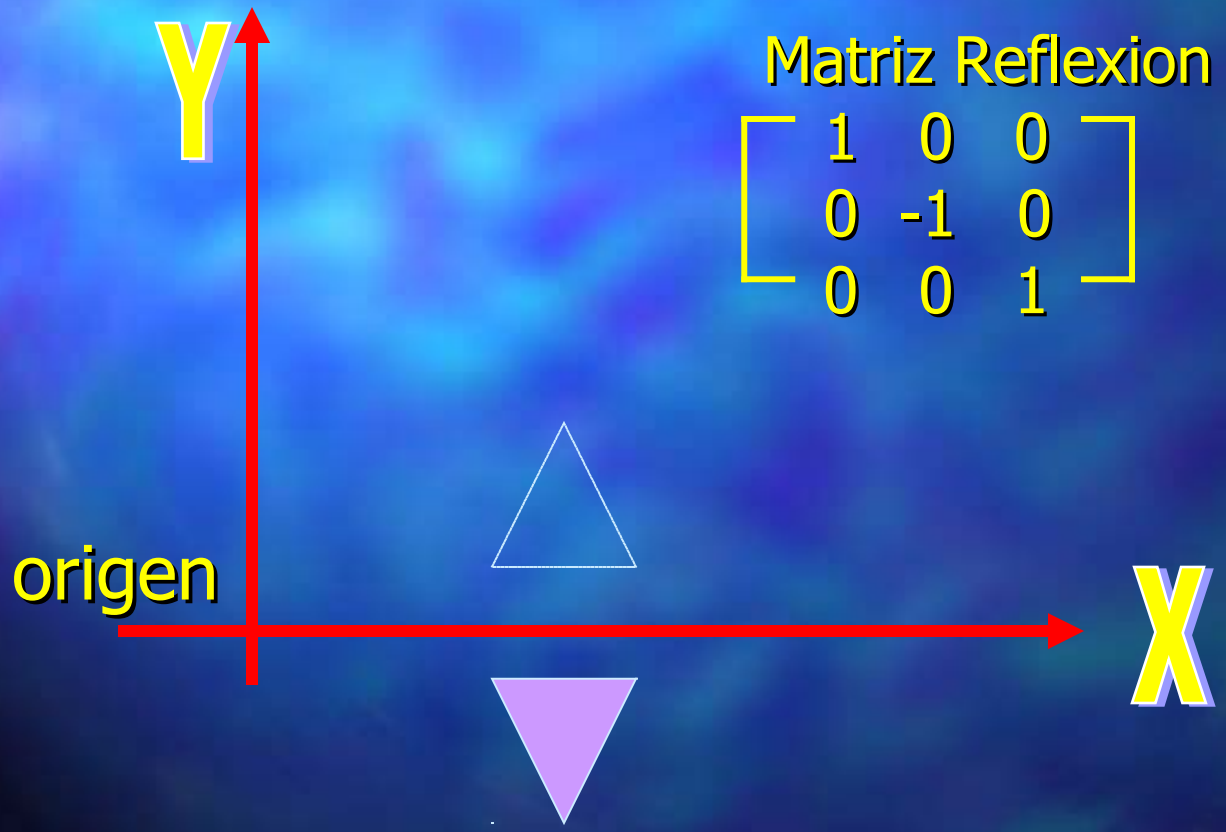
$x' = x$
$y' = Afy * x + y;$
$x' = x + Afx * y$
$y' = y$



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

Matrices y Transformaciones

Reflexion: (sobre X)

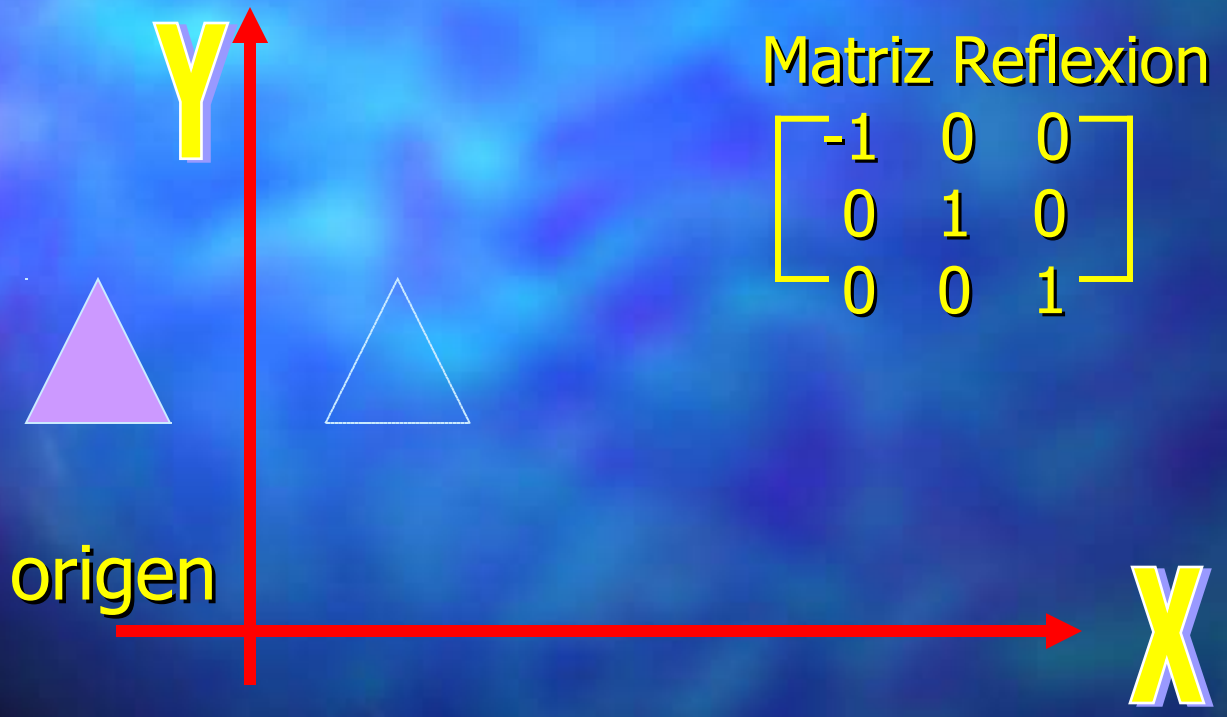


Matriz Reflexion en X:

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

Matrices y Transformaciones

Reflexion: (sobre Y)

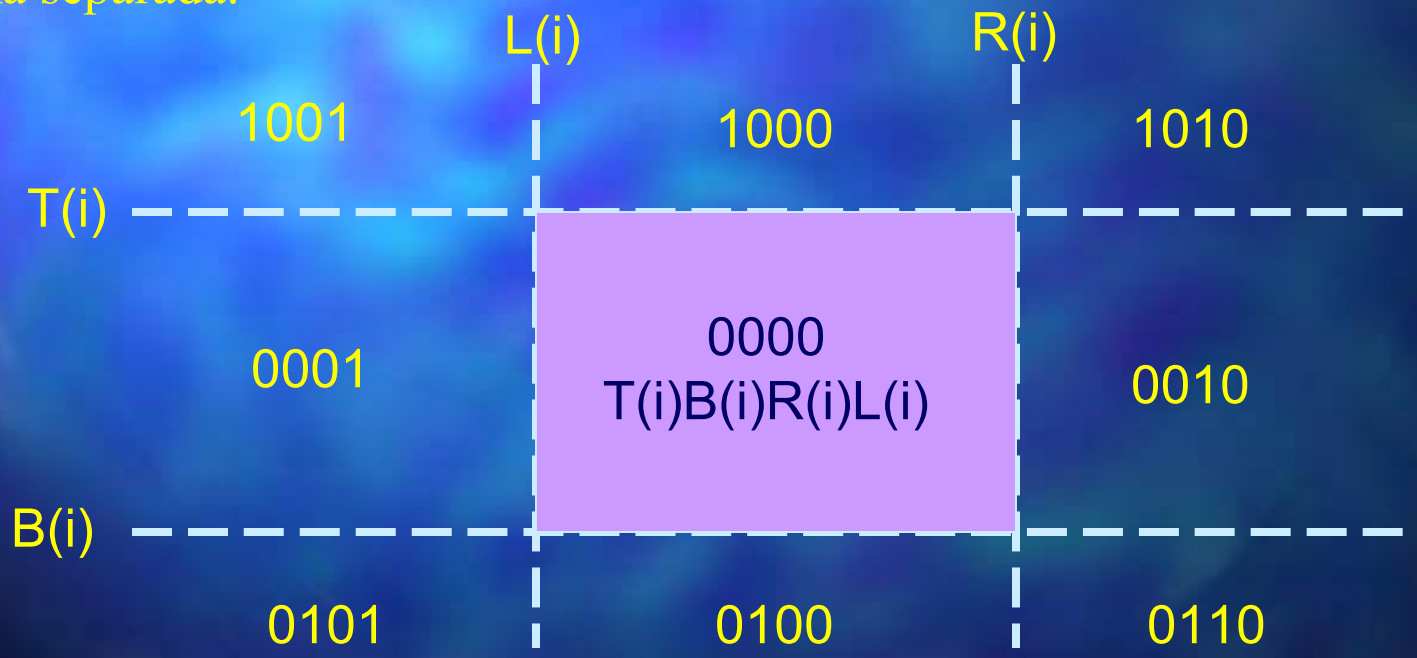


Matriz Reflexion en Y:

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

Recorte Bidimensional (Clipping)

La dificultad del recorte 2dim. se basa en el chequeo y clasificacion de una linea como visible, parcialmente visible o invisible. Para estos efectos se tomara el **algoritmo de Cohen-Sutherland,** el cual se basa en la division de la ventana en nueve regiones y cada punto de la rectas en estudio se analiza de forma separada.



Recorte Bidimensional (Clipping)

Cada extremo de la linea se le asigna un codigo binario que indica si esta en la ventana de visualizacion, arriba, debajo, derecha o izquierda. Cada punto extremo (P1 y P2) esta codificado por cuatro variables: T(i), B(i), R(i) y L(i).

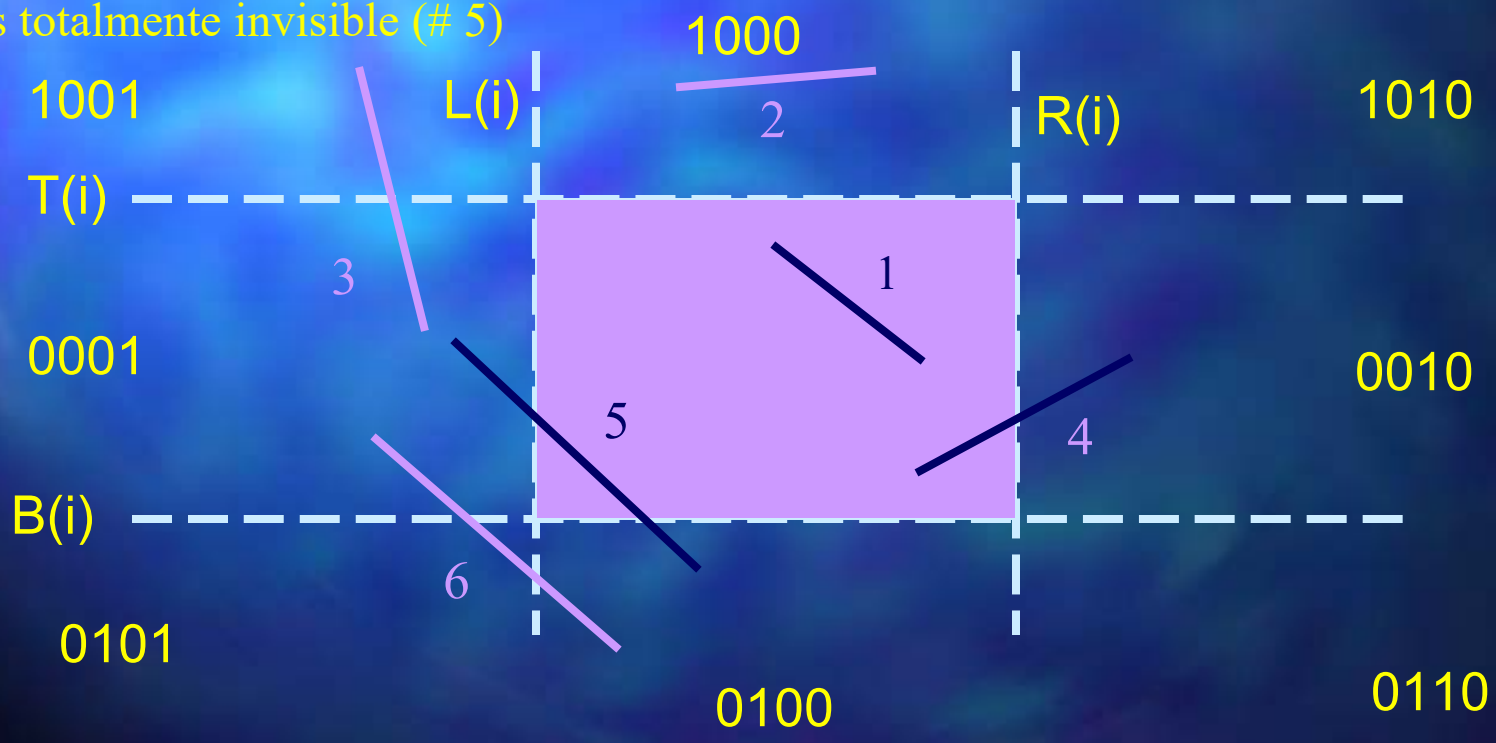
1) Si T(1)+B(1)+R(1)+L(1)+ T(2)+B(2)+R(2)+L(2)=0 **entonces es visible.**

2) Si [T(1)*T(2)]+ [B(1)*B(2)]+ [R(1)*R(2)]+ [L(1)*L(2)] < > 0 **entonces es invisible.**

3) Si [T(1)*T(2)]+ [B(1)*B(2)]+ [R(1)*R(2)]+ [L(1)*L(2)] = 0 **entonces dos opciones:**

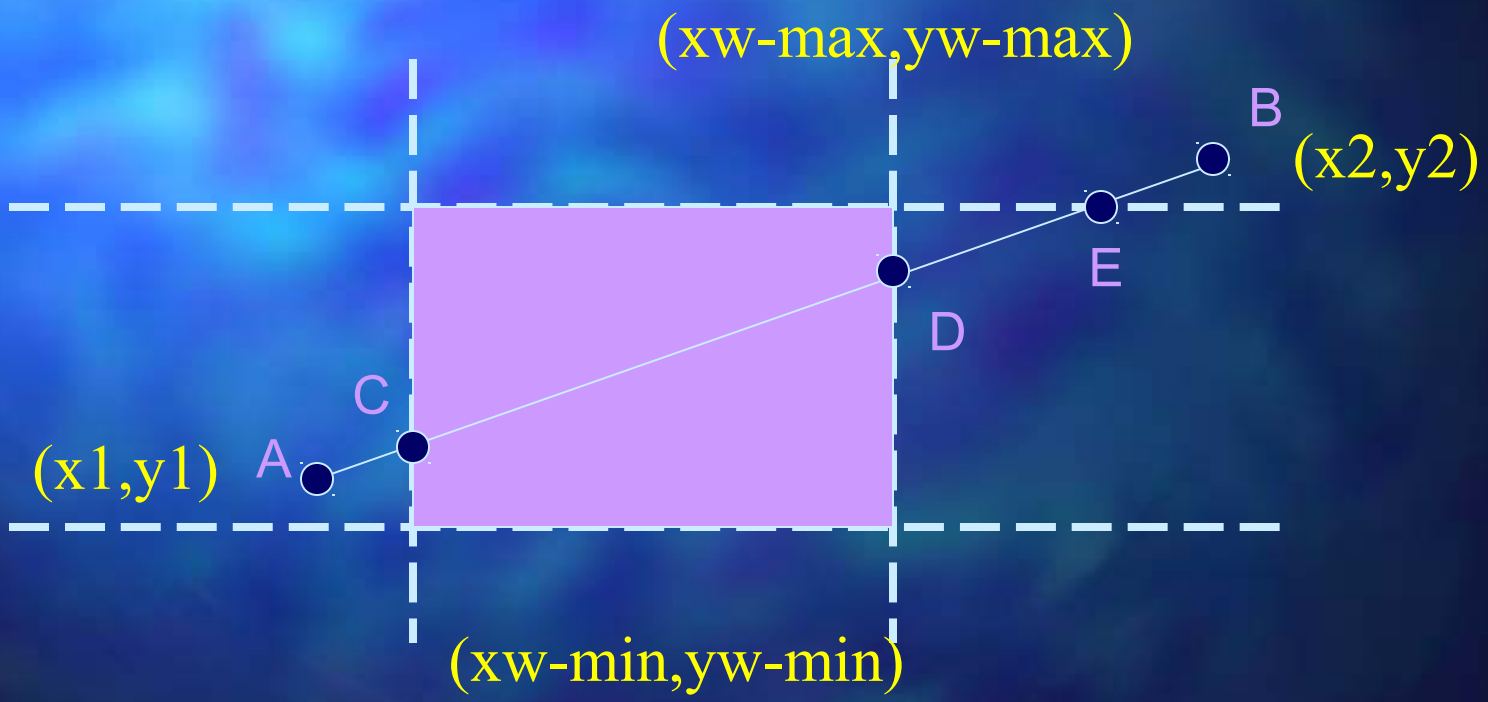
-a) La linea es parcialmente visible (#4 y 5) y deben determinarse las intersecciones o

-b) es totalmente invisible (# 5)



Recorte Bidimensional (Clipping)

Cada vez que se determine una interseccion se debe rechequear si el punto en estudio cumple con la condicion de ser visible o no. Asi el punto A se intersecta con el borde limite en C y el cual es chequeado en su condicion, el punto B intersecta dos veces, en el punto E se determina la primera interseccion y se chequea si es visible o no, al no cumplir la condicion de visibilidad , se codifica nuevamnete y se chequea la siguiente interseccion, punto D, el cual finalmente cumple ser visto.



COMPUTACION GRAFICA I

- **BIBLIOGRAFIA** :

- *Computer Graphics, A Programming Approach*. **Steven Harrington**. Ed.McGraw-Hill, Inc 1983.

Curvas Planas:

La manipulacion de las curvas va a depender de la aplicacion requerida. Generalmente estas aplicaciones tienen dos usos especificos: Analitico y Modelaje.

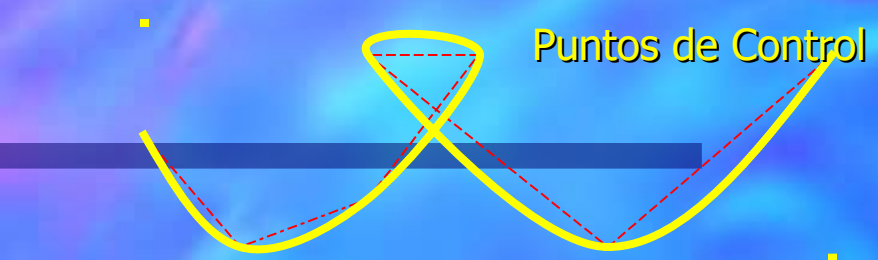
El uso analitico son curvas medibles y relacionadas a conjuntos de puntos como datos de generacion de la misma.

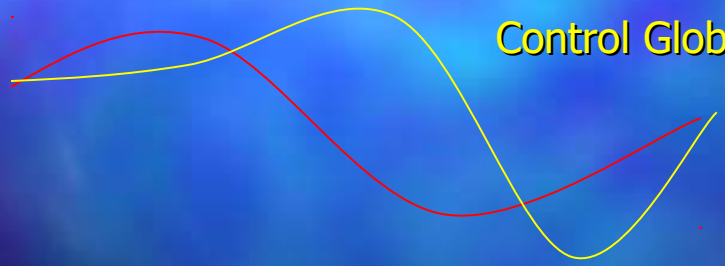
El uso sintetico son curvas utilizadas en el proceso de diseño en general, el cual puede modificar la forma de la curva interactivamente, de esta manera, la nueva curva puede ser almacenada en imagenes o en conjunto de valores para ser procesadas en la industria manufacturera.

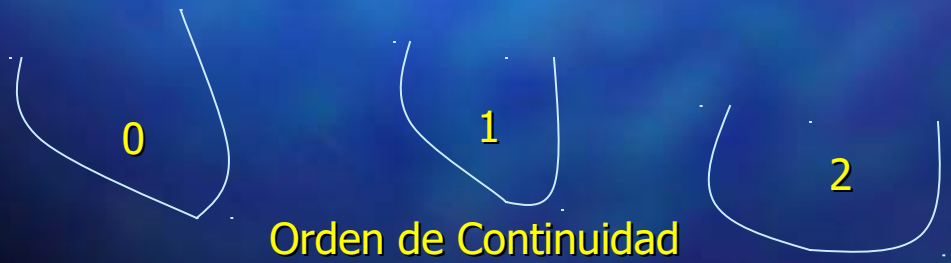
CARACTERISTICAS EN EL DISEÑO DE CURVAS:

- Puntos de Control: a traves de los cuales se permiten pronosticar el comportamiento grafico.
- Valores Multiples: Una curva no tiene valores simples, es una combinacion compleja de datos.
- Independencia de Ejes: Permite mantener la forma de la misma, sin importat el eje utilizado.
- Control Local o Global: Cambios en la forma de manera total o parcial de la curva.
- Propiedades de Variacion Minima: Evitar exageradas oscilaciones al calcular la interpolacion.
- Versatilidad: Un marco adecuado de trabajo para el diseñador, simple y sin confusion.
- Orden de la Continuidad: Permitiendo la generacion de curvas de alta complejidad.

COMPUTACION GRAFICA I







COMPUTACION GRAFICA I

Curvas Planas:

Como expresar una curva sin definicion matematica? Teniendo un arreglo de puntos, podemos generar la aproximacion de la curva, es decir, asumimos como se vera la curva entre dos puntos, todo dependera de cuan suave o acentuada sea la curva entre los puntos asignados.

Existen muchas formas de establecer el comportamiento de la curva, funciones polinomiales, trigonometricas, exponenciales, etc. Una forma es la concocida relacion y = f(x); sin embargo, se usara la forma parametrica que es una funcion polinomial :

x = fx (u) ;

y = fy (u) ; este tipo de funcion no produce dependencia de ejes.

La funcion se representa como:

$$\mathbf{fx\ (u) = \Sigma^{n}_{i=1}\ x_i B_i\ (u);}$$

$$\mathbf{fy\ (u) = \Sigma^{n}_{i=1}\ y_i B_i\ (u);}$$

la funcion $\mathbf{B_i\ (u);}$ se denomina mezcla de funciones para cada valor de u se establece que tanto afecta el punto I la posicion de la curva, pensemos como si cada punto esta tratando de atraer la curva hacia si mismo

COMPUTACION GRAFICA I
curvas planas

Curvas Bezier:

Bezier define la curva **P(u)** en terminos de la localizacion de **n+1** puntos de control p_i

$P(u) = \Sigma^{n}_{i=0} P_i B_{in}(u);$

donde $B_{in}(u) = C(n,i)u^{i}(1-u)^{n-i}$ y $C(n,i) = n! / (i!\,(n-i)!)$; esta ecuacion puede ser escrita para cada punto como:

$x(u) = \Sigma^{n}_{i=0} x_i B_{in}(u);$

$y(u) = \Sigma^{n}_{i=0} y_i B_{in}(u);$

Analisis de Bezier:

- Puntos de Control: Es previsible pero no pasa por todos los puntos de control solo el inicial y el final
- Valores Multiples: Permite multivalor, de hecho si el punto O y el punto n coinciden, la curva se cierra.
- Independencia de Ejes: Si permite independencia de ejes.
- Control Local o Global: No permite control local solo global.
- Propiedades de Variacion Minima: Es de variacion minimizada.
- Versatilidad: Es versatil, pero al incrementar los puntos de control el calculo es mas complejo.
- Orden de la Continuidad: Es limitado.

COMPUTACION GRAFICA I
curvas planas

Curvas B-Splines:

B-Splines define la curva **P(u)** en funcion de un conjunto de funciones de mezcla para algunos puntos de control

$P(u) = \Sigma^{n}_{i=0} P_i N_{ik} (u)$; asi **$N_{ik} (u)$** es la mezcla de funciones y el parametro k controla el orden de continuidad de la curva el parametro **u** a diferencia de Bezier **(0<u<1)** toma valores que van de **0** a **n-k+2;** la representacion por cada punto es:

$x(u) = \Sigma^{n}_{i=0} x_i N_{ik} (u)$;

$y(u) = \Sigma^{n}_{i=0} y_i N_{ik} (u)$;

Analisis de B-Splines:

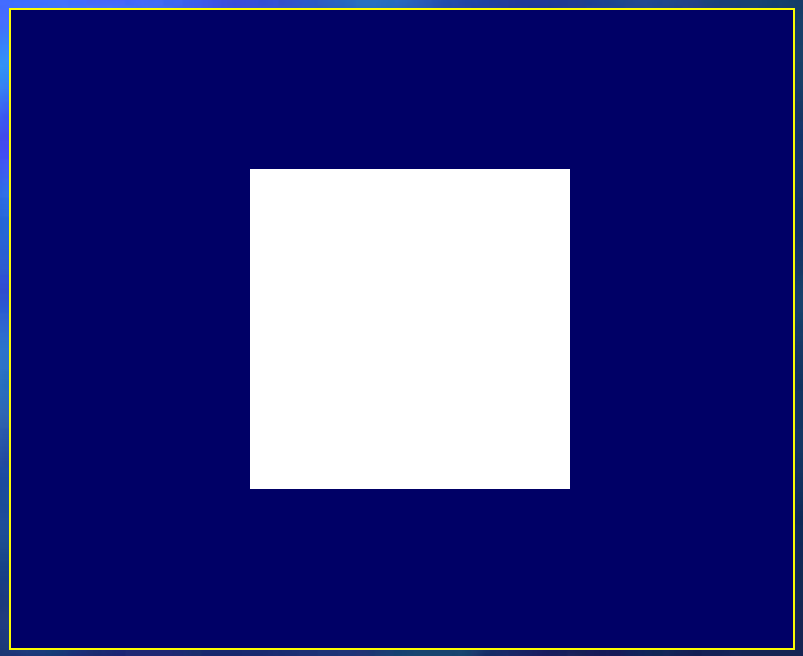
- Puntos de Control: Pasa por todos los puntos de control deseados.
- Valores Multiples: Permite multivalor
- Independencia de Ejes: Si permite independencia de ejes.
- Control Local o Global: Permite control local.
- Propiedades de Variacion Minima: Es de variacion minimizada.
- Versatilidad: Es muy versatil.
- Orden de la Continuidad: Es ilimitado.

OpenGL

Es un sistema de interfase grafico, estandarizado por Silicon Graphics para trabajar en cualquier plataforma (UNIX, Lynux, Windows).

El Open GL es particularmente util para modelos, los cuales son generados de primitivas geometricas, puntos, lineas y poligonos, que a su vez son especificadas por sus vertices.

Dibujo Cuadrado Blanco en Fondo Negro



OpenGL

Programa para generar un cuadrado blanco en fondo negro.

```
#include <los_programas_headers_que_se_necesiten.h>
main()  {
  Inicializa_Ventana_Grafica();   // depende del sistema operativo
  glClearColor (0.0,0.0,0.0,0.0);  // color de ventana
  glClear(GL_COLOR_BUFFER_BIT);  // Limpia Ventana
  glColor3f (1.0,1.0,1.0);  // Establece el color del Trazo
  glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0); // Proyeccion y Sistema  Coordenado
  glBegin (GL_POLYGON);  // Inicia Dibujo de la figura plana
            glVertex3f(0.25,0.25,0.0);
            glVertex3f(0.75,0.25,0.0);
            glVertex3f(0.75,0.75,0.0);
            glVertex3f(0.25,0.75,0.0);
  glEnd();
  glFlush();//Asegura ejecucion del Dibujo en lugar de almacenar en buffer
  Actualiza_Ventana_y_Chequea_Eventos( );
}
```

OpenGL

Todos los comandos e instrucciones del OpenGl utilizan el prefijo **gl** seguidos de una letra mayuscula para diferenciar los comandos (**glClearColor( )**); igualmente utiliza letras mayusculas para designar las constantes, que son valores asociados al lenguaje OpenGL, seguidos del simbolo _ (**GL_COLOR_BUFFER_BIT**).

Otras caracteristicas en la sintaxis es la utilizacion de algunas letras y numeros a los comandos utilizados, por ejemplo el **3f** en **glColor3f( )** y **glVertex3f( ),** en donde el **3** significa el numero de parametros pasados a la funcion y la **f** es el tipo de parametro en este caso es real de punto flotante; asi entonces, tendremos que **glColor2i( )** y **glVertex2i( )** son validos.

Algunos comandos toman al final la letra **v**, que indica un apuntador a un vector o arreglo matricial, por ejemplo:

glColor3f(1.0,0.0,0.0);

equivale a :

Glfloat color_array { } = {1.0,0.0,0.0 };

glColor3fv(color_array);

OpenGL

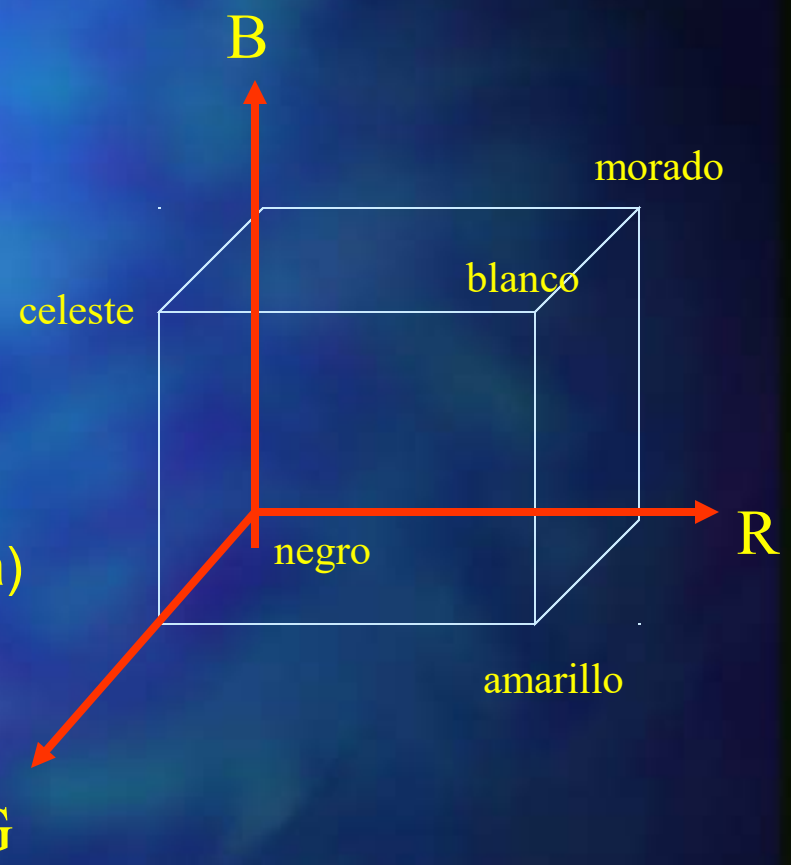
Tipos de Variables Numericas en OpenGL:

GLbyte - b - 8 bits
GLshort - s - 16 bits
GLint,GLsizei - i - 32 bits
GLfloat,GLclampf - f - 32 bits floating
GLdouble,GLclampd - d - 64 bits floating
GLubyte,GLboolean -ub - 8 bits unsigned integer
GLushort - us - 16 bits unsigned integer
GLuint,GLenum,GLbitfield - ui - 32 bits unsigned integer

OpenGL

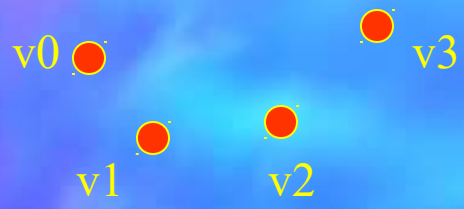
Colores en OpenGL:

```
glColor3f(0.0,0.0,0.0);  // negro
glColor3f(1.0,0.0,0.0);  // rojo
glColor3f(0.0,1.0,0.0);  // verde
glColor3f(1.0,1.0,0.0);  // amarillo
glColor3f(0.0,0.0,1.0);  // azul
glColor3f(1.0,0.0,1.0);  // morado (magenta)
glColor3f(0.0,1.0,1.0);  // celeste (cyan)
glColor3f(1.0,1.0,1.0);  // blanco
```

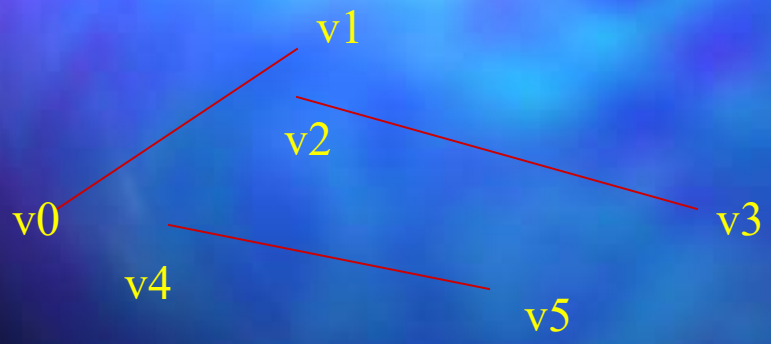


OpenGL

Comandos de Figuras Geometricas Primitivas Convexas en OpenGL:



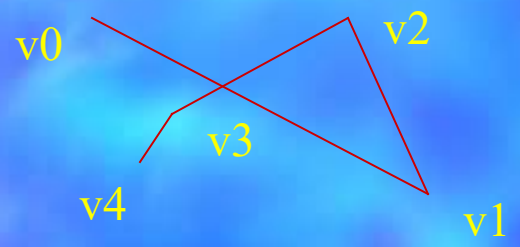
GL_POINTS



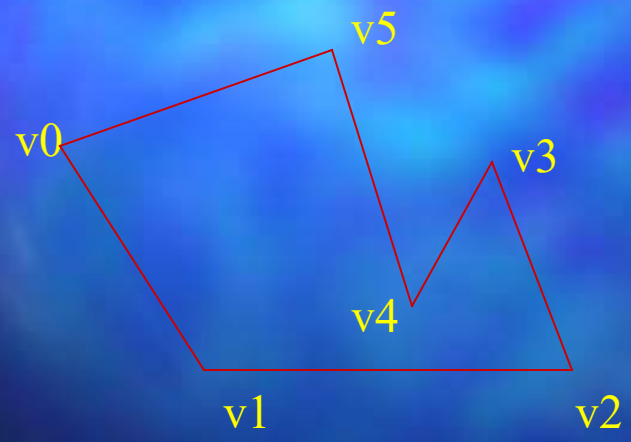
GL_LINES

OpenGL

Comandos de Figuras Geometricas Primitivas Convexas en OpenGL:



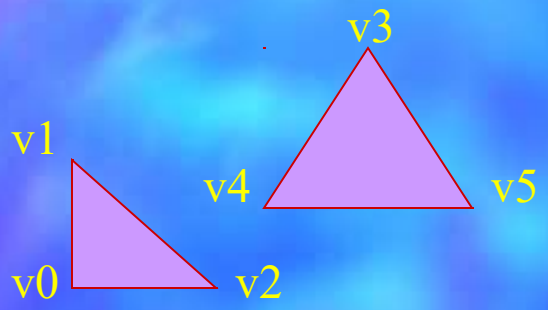
GL_LINE_STRIP



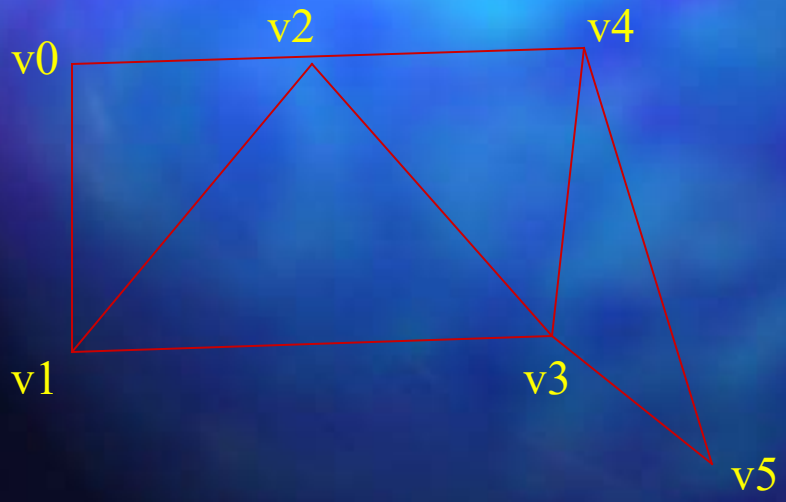
GL_LINE_LOOP

OpenGL

Comandos de Figuras Geometricas Primitivas Convexas en OpenGL:



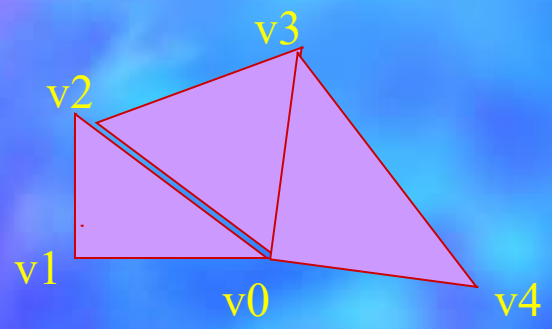
GL_TRIANGLES



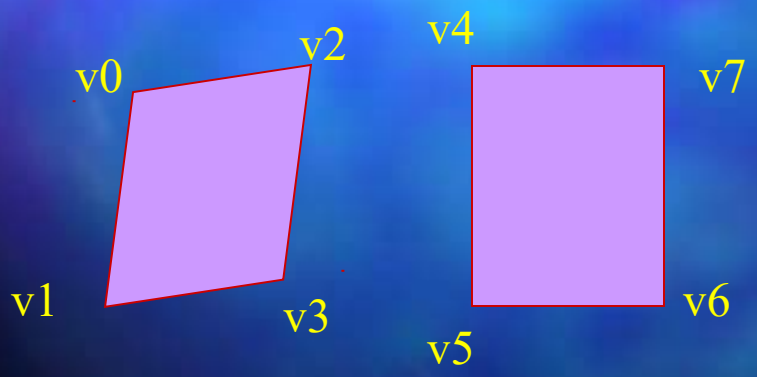
GL_TRIANGLE_STRIP

OpenGL

Comandos de Figuras Geometricas Primitivas Convexas en OpenGL:



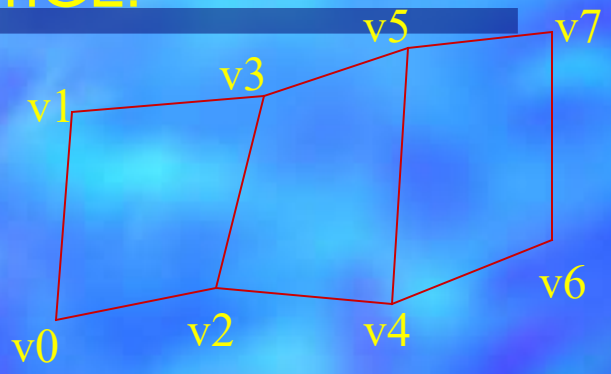
GL_TRIANGLE_FAN



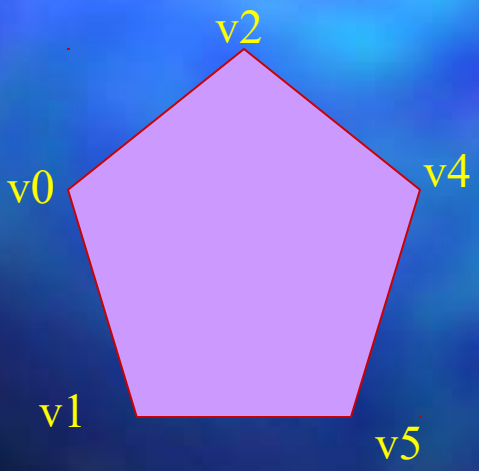
GL_QUADS

OpenGL

Comandos de Figuras Geometricas Primitivas Convexas en OpenGL:



GL_QUAD_STRIP



GL_POLYGON

OpenGL

Comandos Validos entre glBegin y glEnd:

glVertex*();
glColor*();
glIndex*();
glNormal*();
glTexCoord*();
glEdgeFlag*();
glMaterial*();
glArrayElement();
glEvalCoord*() , glEvalPoint*();
glCallList() , glCallLists();

Estos comandos aseguran el correcto funcionamiento del OpenGL en este tipo de estructura, otros diferentes pueden o no generar errores, y tambien tener comportamientos no establecidos de manera aleatoria.

OpenGL

Otros Comandos :

void glPolygonMode(Glenum ***face*** , Glenum ***mode***);
Controla la manera de generar el grafico tanto para la parte frontal como posterior. El primer parametro (face) puede ser GL_FRONT_AND_BACK, GL_FRONT, o GL_BACK ; el segundo parametro (mode) alterna valores con GL_POINT, GL_LINE o GL_FILL, por ejemplo:

glPolygonMode(GL_FRONT, GL_FILL); *presenta el poligono frontal y relleno*
glPolygonMode(GL_BACK , GL_LINE); *presenta el poligono posterior y con linea sin rellenar*
Por convencion los poligonos se dibujan sentido contrario a las agujas del reloj para hacerlo orientable; sin embargo en caso de desear cambiar este criterio se usa la funcion:
void glFrontFace(Glenum ***mode***);
donde mode toma valores GL_CCW (valor por defecto) = sentido horario inverso o GL_CW = sentido horario. Para descartar poligonos de cara posterior en el grafico usese la funcion : void glCullFace(Glenum ***mode***); donde mode toma valores como GL_FRONT_AND_BACK, GL_FRONT, o GL_BACK ; para que esta funcion surta efecto debemos habilitarla con : glEnable (GL_CULL_FACE) y deshabilitarla con glDisable (GL_CULL_FACE)

OpenGL

Matrices de Transformacion :

glTranslate{fd}(TYPE l,TYPE m,TYPE n); l,m,n es el punto de traslado

glRotate{fd}(TYPE angulo, TYPE x,TYPE y,TYPE z); ejemplo: (45.0,0.0,0.0,1.0) a 45 grados alrededor del eje z

glScale{fd}(TYPE x,TYPE y,TYPE z);se usa tanto para escalar como producir reflexion.
Valores>1.0 aumenta ; Valores <1.0 disminuye.
Valores= -1.0 producen reflexion

glLoadMatrix{fd}(const TYPE *m); carga los valores de una matriz determinada.
glMultMatrix{fd}(const TYPE *s); multiplica la matriz s por la matriz (m).
Ejemplo de codigo:

```
glLoadIdentity( );
glColor(1.0,1.0,1.0);
drawTriangle( );  // dibuja el trinagulo en linea solida por defecto

glEnable(GL_LINE_STIPPLE);
glLineStipple(1,0xF0F0);  // linea interrumpida
glLoadIdentity( );
glTranslatef(-3.0,0.0,0.0);
drawTriangle( );
```

OpenGL

Matrices de Transformacion :

Continua ejemplo de codigo:

```
glLineStipple(1,0xF00F);  // linea interrumpida larga
glLoadIdentity( );
glTranslatef(1.2,0.3,1.0);
drawTriangle( );

glLineStipple(1,0x8888);  // linea punteada
glLoadIdentity( );
glRotatef(60.0,0.0,0.0,1.0);  // 60 grados con respecto al eje z
drawTriangle( );
glDisable(GL_LINE_STIPPLE);
```