UMSNH
Facultad de Ing. Eléctrica
Carrera: Ingeniería en Computación
Materia: Graficación

Dr. José Antonio Camarena Ibarrola

Septiembre de 2010

Introducción

Aplicaciones: Simulación

Introducción

- Video-juegos

http://www.acclaim.com/games/crazytaxi/

Introducción

- Animación por computadora

http://www.pixar.com

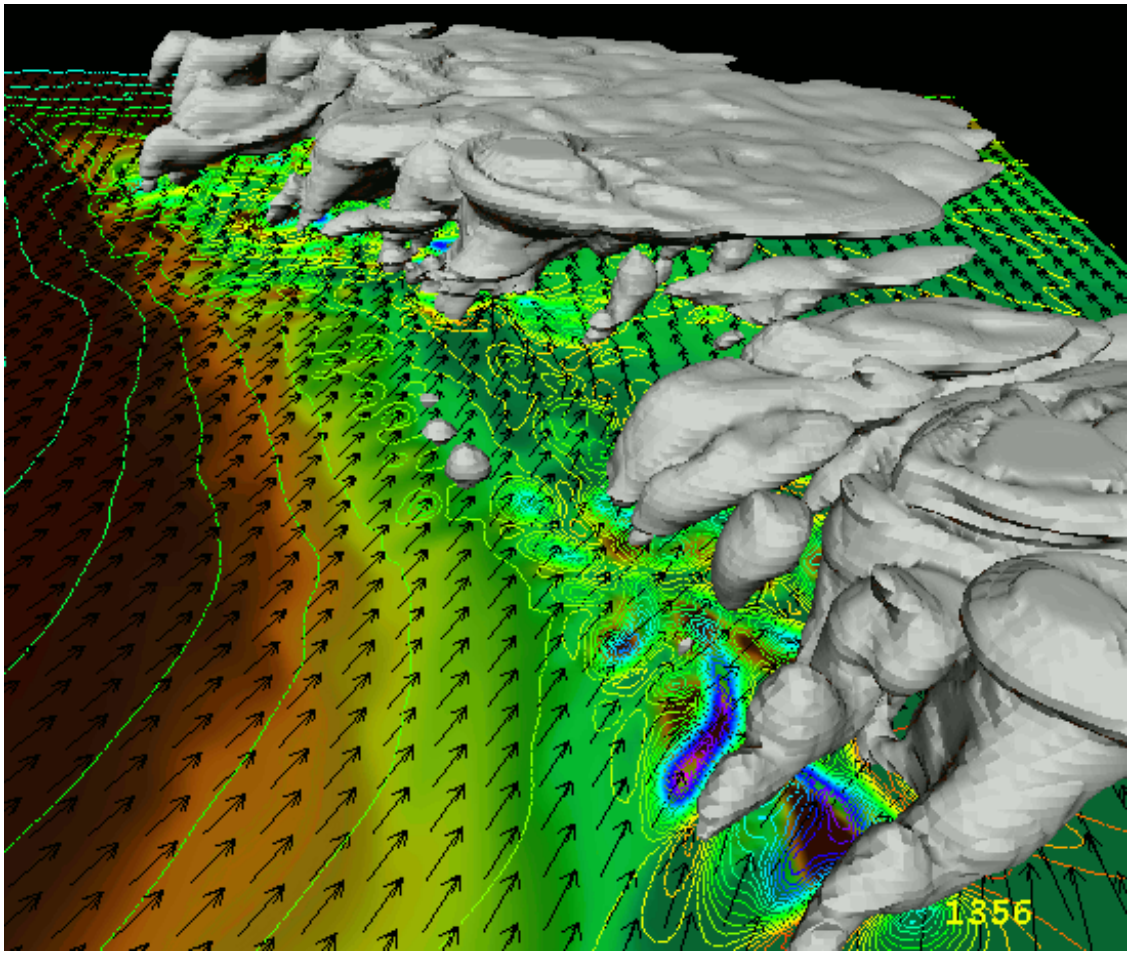
Introducción

- Diseño (recorridos virtuales 3D)

The William gates Building
http://www3.arct.cam.ac.uk/westC/cl/cl.html

Introducción

- Visualización de gráficas 3D

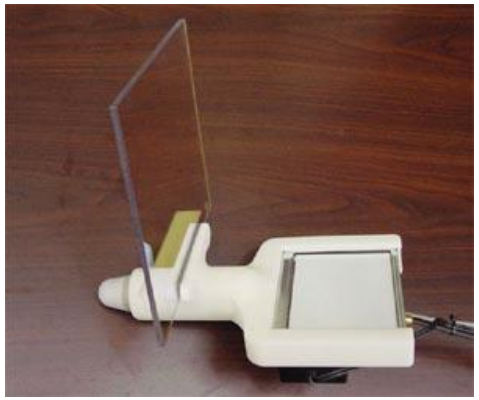


Convective modelling group, AOS, Univ Illinois at Urbana-Champaign
http://redrock.ncsa.uiuc.edu/AOS/home.html
(NCSA storm model)

Introducción

- Interfaces en la medicina

The Sonic Flashlight, MRT Center, The Robotics Institute,
Carnegie Mellon University,
http://www.ri.cmu.edu/projects/project_438.html



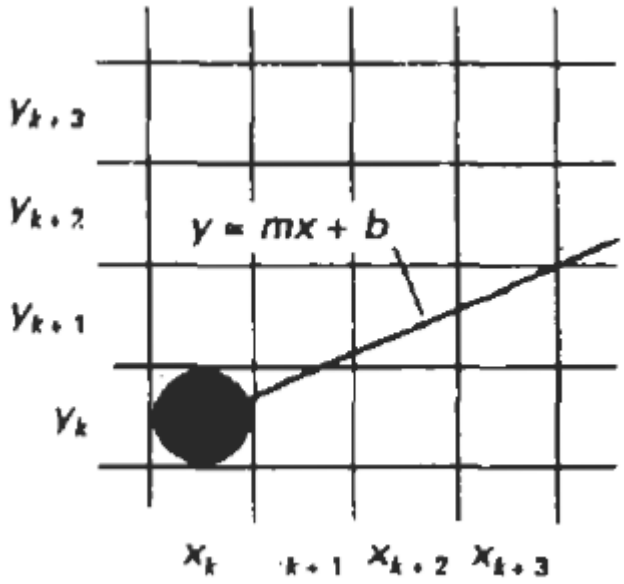
Primitivas de Gráficación

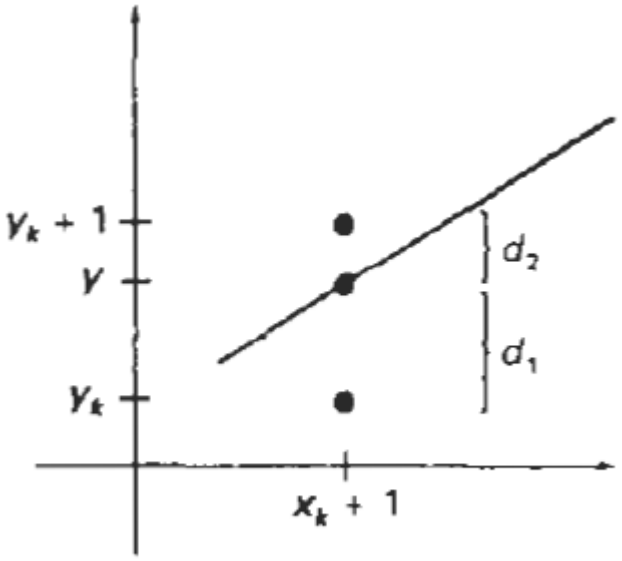
- Algoritmos de trazado de líneas. Algoritmo DDA (Digital Diferential Analizer), Algoritmo de Bresenham.
- Algoritmo de Bresenham para trazado de circunferencias, Algoritmo del punto medio para trazado de circunferencias
- Algoritmo del punto medio para generación de elipses
- Polilíneas
- Curvas Splines cúbicas naturales, Splines de Hermite, Curvas de Bezier.
- Estructura de un Programa OpenGL
- Despliege de lineas, triángulos, cuadrados, circunferencias, etc mediante OpenGL

Primitivas de Graficación
Trazado de líneas
Algoritmo DDA (Digital Diferential Analizer)

Input: x_{ini},y_{ini},x_{fin},y_{fin}
Output: Traza Línea recta
if $|x_{fin} - x_{ini}| \geq |y_{fin} - y_{ini}|$ **then**
  longitud=$|x_{fin} - x_{ini}|$;
else
  longitud=$|y_{fin} - y_{ini}|$;
end
$\Delta x = (x_{fin} - x_{ini})/longitud$;
$\Delta y = (y_{fin} - y_{ini})/longitud$;
$x = x_{ini} + 0{,}5 * Signo(\Delta x)$;
$y = y_{ini} + 0{,}5 * Signo(\Delta y)$;
$k = 1$;
while $k \leq longitud$ **do**
  plot(floor(x),floor(y));
  $x = x + \Delta x$;
  $y = y + \Delta y$;
  $k = k + 1$;
end

Primitivas de Graficación
Trazado de líneas
Algoritmo de Bresenham





Con el signo de d1-d2 podemos decidir cual pixel
Está mas cerca de la línea ideal

Primitivas de Graficación
Trazado de líneas
Algoritmo de Bresenham

Input: x_{ini},y_{ini},x_{fin},y_{fin}
Output: Traza Línea recta
$x = x_{ini}$;
$y = y_{ini}$;
$\Delta x = x_{fin} - x_{ini}$;
$\Delta y = y_{fin} - y_{ini}$;
$e = \Delta y/\Delta x - 1/2$;
for $i = 1$ **to** Δx **do**
    Plot(floor(x),floor(y));
    while $e \geq 0$ **do**
        $y = y + 1$;
        $e = e - 1$;
    end
    $x = x + 1$;
    $e = e + \Delta y/\Delta x$;
end

Problema: Requiere modificarse para evitar aritmética de flotantes

Primitivas de Graficación
Trazado de líneas
Algoritmo de Bresenham

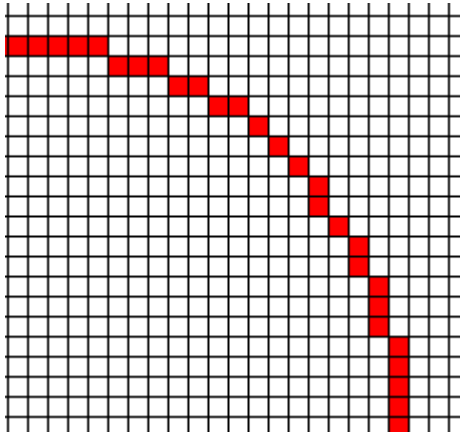
Input: x_{ini},y_{ini},x_{fin},y_{fin}
Output: Traza Línea recta
$x = x_{ini}$;
$y = y_{ini}$;
$\Delta x = x_{fin} - x_{ini}$;
$\Delta y = y_{fin} - y_{ini}$;
$\hat{e} = 2\Delta y - \Delta x$;
for $i = 1$ **to** Δx **do**
    Plot(x,y);
    while $\hat{e} \geq 0$ **do**
        $y = y + 1$;
        $\hat{e} = \hat{e} - 2\Delta x$;
    end
    $x = x + 1$;
    $\hat{e} = \hat{e} + 2\Delta y$;
end

Esta versión usa solo aritmética entera

Primitivas de Graficación
Trazado de circunferencias
Algoritmo de Bresenham

- La idea básica es usar solo el signo del error en que se incurre para decidir cual pixel encender

```
y=r;
 d= -r;
pixel(0,r);
for(x=1;x<r/sqrt(2);x++) {
    d+= 2x-1;
   if (d>=0) {
        y--;
        d -= 2y;
   }
   pixel(x,y);
}
```

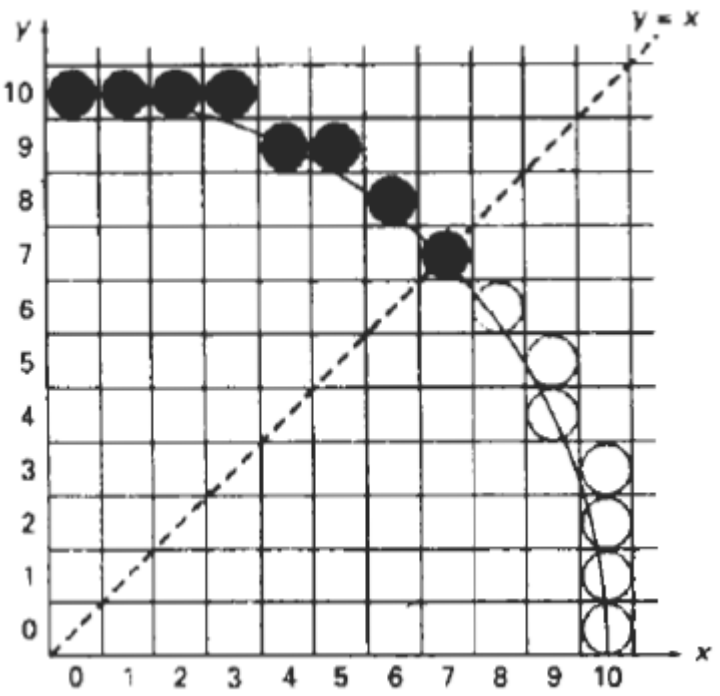


Primitivas de Graficación
Trazado de circunferencias
Algoritmo del punto medio

Input: x_c,y_c,R
Output: Traza Circunferencia de radio R centrada en $(x_c, y_c$
$x = 0$;
$y = R$;
p=1-R;
while $x \leq y$ **do**
  Plot$(x_c + x, y_c + y)$;
  Plot$(x_c + x, y_c - y)$;
  Plot$(x_c - x, y_c + y)$;
  Plot$(x_c - x, y_c - y)$;
  Plot$(x_c + y, y_c + x)$;
  Plot$(x_c + y, y_c - x)$;
  Plot$(x_c - y, y_c + x)$;
  Plot$(x_c - y, y_c - x)$;
  $x = x + 1$;
  if $p \leq 0$ **then**
    p=p+2x+1;
  else
    y=y-1;
    p=p+2x-2y+1;
  end
end

$$f_{circ}(x,y) = x^2 + y^2 - R^2 = \begin{cases} < 0 & \text{(x,y) está dentro del círculo} \\ 0 & \text{(x,y) está justo en el círculo} \\ > 0 & \text{(x,y) está fuera del círculo} \end{cases}$$

$$p_k = f_{circ}(x_k + 1, y_k - 1/2)$$



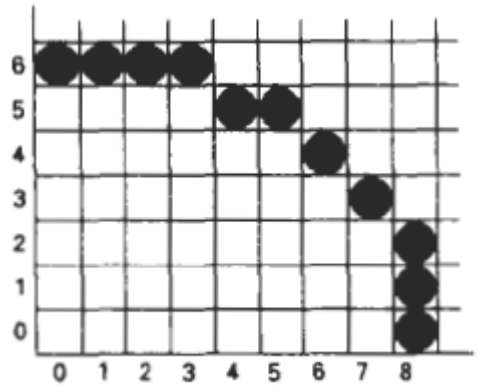
Primitivas de Graficación
Algoritmo del punto medio para trazado de elipses

Input: x_c,y_c,R_x,R_y
Output: Traza elipse de radios R_x, R_y centrada en $(x_c, y_c$

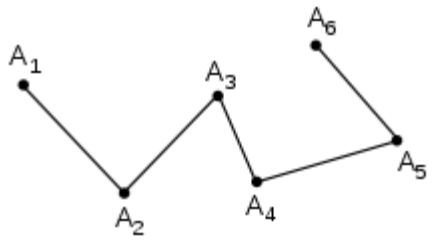
$x = 0;$
$y = R_y;$
$p1 = R_y^2 - R_y R_x^2 + \frac{1}{4}R_x^2;$
while $2R_y^2 x \leq 2R_x^2 y$ **do**
  $\text{Plot}(x_c + x, y_c + y);$
  $\text{Plot}(x_c + x, y_c - y);$
  $\text{Plot}(x_c - x, y_c + y);$
  $\text{Plot}(x_c - x, y_c - y);$
  $x = x + 1;$
  if $p1 \leq 0$ **then**
    $p1 = p1 + 2xR_y^2 + R_y^2;$
  else
    y=y-1;
    $p1 = p1 + 2xR_y^2 + R_y^2 - 2R_x^2 y;$
  end
end
$p2 = R_y^2(x + 1/2)^2 + R_x^2(y - 1)^2 - R_x^2 R_y^2;$
while $y > 0$ **do**
  $y = y - 1;$
  if $p2 \leq 0$ **then**
    $x = x + 1;$
    $p2 = p2 + 2xR_y^2 - 2R_x^2 y + R_x^2;$
  else
    $p2 = p2 - 2yR_x^2 + R_x^2;$
  end
  $\text{Plot}(x_c + x, y_c + y);$
  $\text{Plot}(x_c + x, y_c - y);$
  $\text{Plot}(x_c - x, y_c + y);$
  $\text{Plot}(x_c - x, y_c - y);$
end

-Usamos la ecuación de la elipse para decidir si el punto medio está dentro o fuera de la elipse

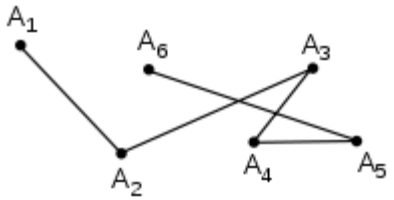
-Como no tenemos simetría a nivel octante, tenemos que generar todo un cuadrante



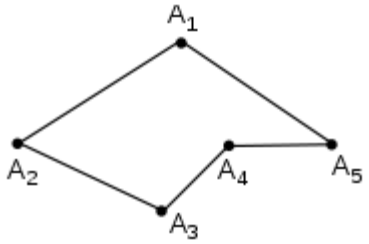
Primitivas de Graficación
Polilíneas



Abierta



Autointersectada



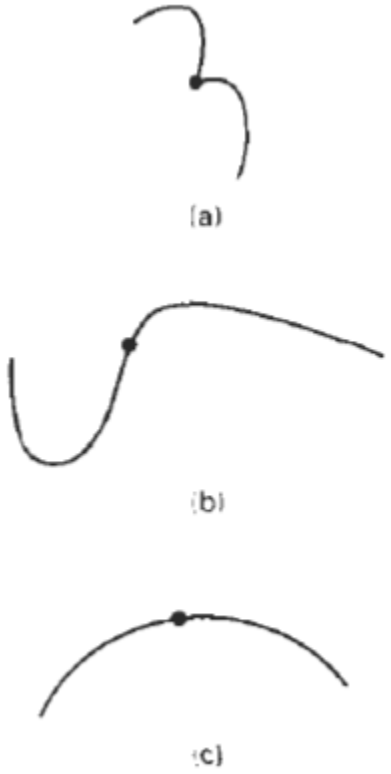
Cerrada

Primitivas de Graficación
Splines

De interpolación

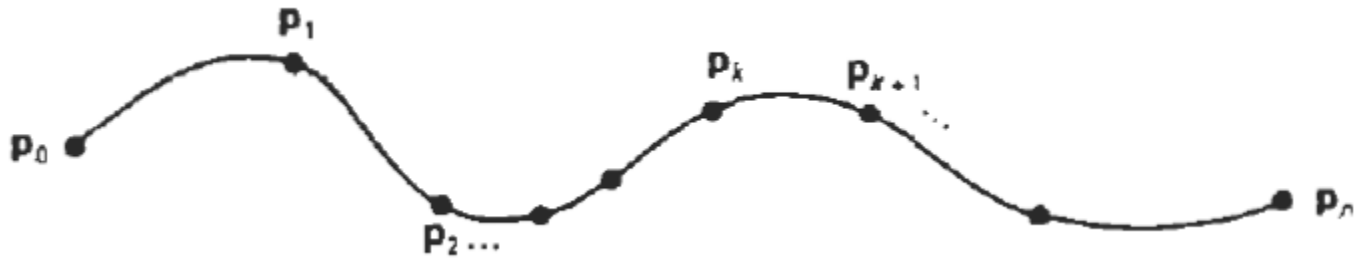
De aproximación

Primitivas de Graficación
Splines



(a) Continuidad de orden cero. (b) de primer orden c) de segundo orden

Primitivas de Graficación
Curvas Splines cúbicas naturales



Cada segmento se representa por un polinomio de tercer grado (cúbico)

Por cada polinomio cúbico hay 4 incógnitas

Hay que resolver un sistema de 4n ecuaciones con 4n incógnitas

Si un punto de control se modifica hay que volver a solucionar el sistema de ecuaciones

En estas splines no hay control local

Tienen continuidad de segundo orden

Primitivas de Graficación
Splines de Hermite

Tienen continuidad de primer orden

Permiten control local

Restricciones

$$\begin{aligned} P(0) &= p_k \\ P(1) &= p_{k+1} \\ P'(0) &= DP_k \\ P'(1) &= DP_{k+1} \end{aligned}$$

$$P(u) = au^3 + bu^2 + cu + d$$

Por cada segmento hay que resolver un sistema de 4x4

$$\begin{bmatrix} p_k \\ p_{k+1} \\ DP_k \\ DP_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Primitivas de Graficación
Splines de Hermite

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_k \\ p_{k+1} \\ DP_k \\ DP_{k+1} \end{bmatrix}$$

Como la matriz de coeficientes no cambia en realidad solo se requiere de una simple multiplicación matricial por cada segmento

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2p_k - 2p_{k+1} + DP_k + DP_{k+1} \\ 3p_k - 3p_{k+1} - 2DP_k - DP_{k+1} \\ DP_k \\ p_k \end{bmatrix}$$

$$P(u) = u^3(2p_k - 2p_{k+1} + DP_k + DP_{k+1}) + u^2(3p_k - 3p_{k+1} - 2DP_k - DP_{k+1}) + uDP_k + p_k$$

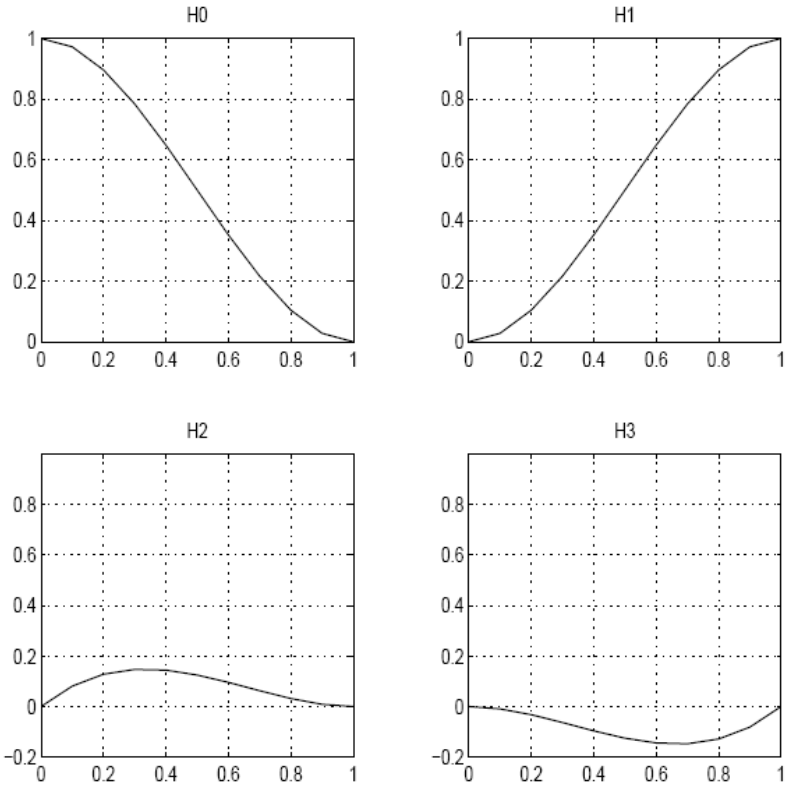
Primitivas de Graficación
Splines de Hermite

Reagrupando:

$$P(u) = p_k(2u^3-3u^2+1)+p_{k+1}(-2u^3+3u^2)+DP_k(u^3-2u^2+u)+DP_{k+1}(u^3-u^2)$$

$$P(u) = p_k H_0(u) + p_{k+1} H_1(u) + DP_k H_2(u) + DP_{k+1} H_3(u)$$

Funciones de ponderación



Primitivas de Graficación
Splines cardinales

Problema: las Splines de Hermite requieren que el usuario especifique la pendiente En cada punto de control

Solución: Estimar las pendientes usando las formulas

$$P'(0) = \frac{1}{2}(1-t)(p_{k+1} - p_{k-1})$$

$$P'(1) = \frac{1}{2}(1-t)(p_{k+2} - p_{k})$$

Donde t es el parámetro de tensión

Primitivas de Graficación
Curvas de Bezier

Pierre Bézier trabajaba para la Renault diseñando carrocerías

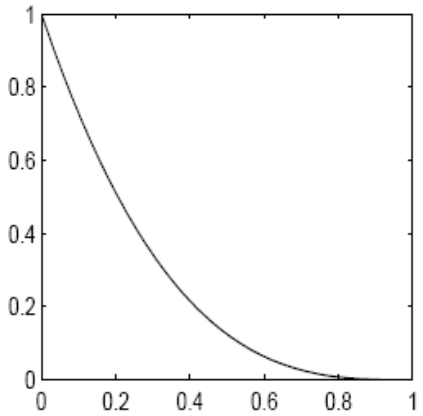
$$P(u) = \sum_{k=0}^{n} p_k B_{k,n}(u)$$

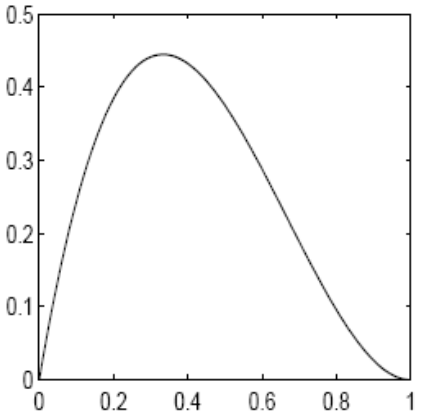
Donde: $B_{k,n}(u) = \binom{n}{k} u^k (1-u)^{n-k}$

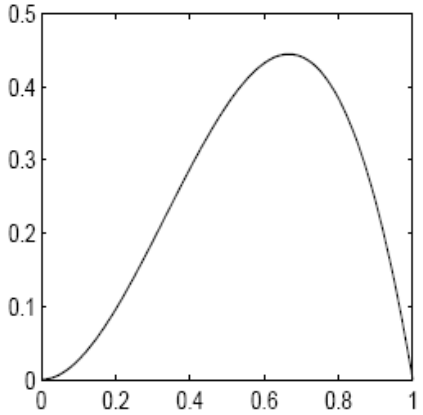
Primitivas de Graficación
Curvas de Bézier

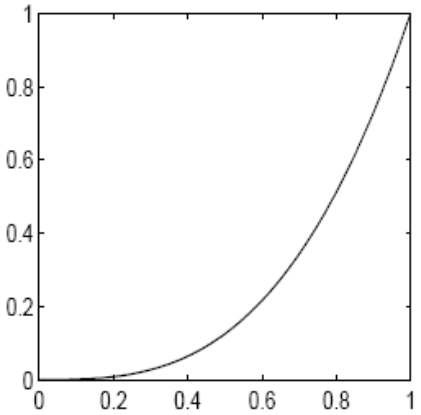
Por ejemplo, para n=4 (número de puntos de control)

$$
\begin{aligned}
B_{0,3}(u) &= (1-u)^3 \\
B_{1,3}(u) &= 3u(1-u)^2 \\
B_{2,3}(u) &= 3u^2(1-u) \\
B_{3,3}(u) &= u^3
\end{aligned}
$$









Primitivas de Graficación
Estructura de un programa en OpenGL

```
import net.java.games.jogl.*

public static void main(String[] args) {
    Frame frame = new Frame("Hello World");
    GLCanvas canvas = GLDrawableFactory.getFactory().createGLCanvas(new
    GLCapabilities());

    frame.add(canvas);
    frame.setSize(300, 300);
    frame.setBackground(Color.WHITE);

    frame.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
    });
    frame.show();
```

Primitivas de Graficación
Despliegue de líneas, triángulos y cuadrados en OpenGL

Despliega dos líneas

```
gl.glBegin (GL.GL_LINES);
gl.glVertex2i (50, 200);
gl.glVertex2i (75, 250);
gl.glVertex2i (60, 200);
gl.glVertex2i (85, 250);
gl.glEnd();
```

Despliega polilínea

```
gl.glBegin (GL.GL_LINE_STRIP);
gl.glVertex2i (100, 200);
gl.glVertex2i (150, 250);
gl.glVertex2i (100, 250);
gl.glVertex2i (150, 200);
gl.glEnd();
```

Primitivas de Graficación
Despliegue de líneas, triángulos y cuadrados en OpenGL

Despliega polilínea cerrada

```
gl.glBegin (GL.GL_LINE_LOOP);
gl.glVertex2i (200, 200);
gl.glVertex2i (250, 250);
gl.glVertex2i (200, 250);
gl.glVertex2i (250, 200);
gl.glEnd();
```

Despliega dos triángulos

```
gl.glBegin (GL.GL_TRIANGLES);
gl.glVertex2i (400, 50);
gl.glVertex2i (400, 100);
gl.glVertex2i (420, 75);
gl.glVertex2i (425, 50);
gl.glVertex2i (425, 100);
gl.glVertex2i (445, 75);
gl.glEnd();
```

Primitivas de Graficación
Despliegue de líneas, triángulos y cuadrados en OpenGL

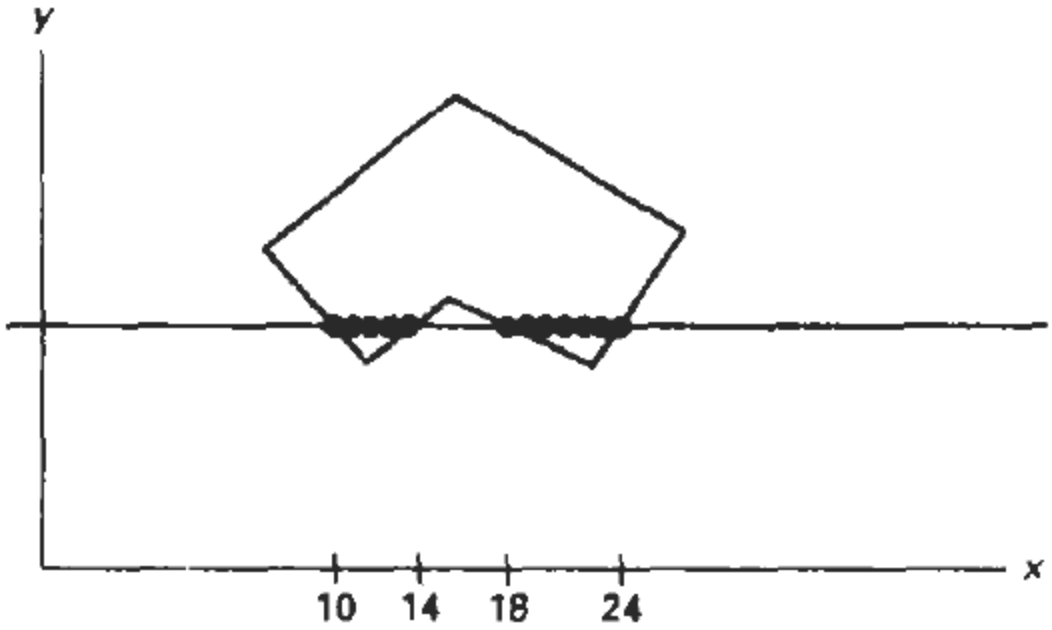
Despliega polígono

```
gl.glBegin (GL.GL_POLYGON);
gl.glVertex2i (300, 50);
gl.glVertex2i (350, 60);
gl.glVertex2i (375, 100);
gl.glVertex2i (325, 115);
gl.glVertex2i (300, 75);
gl.glEnd();
```

Algoritmos de relleno de áreas

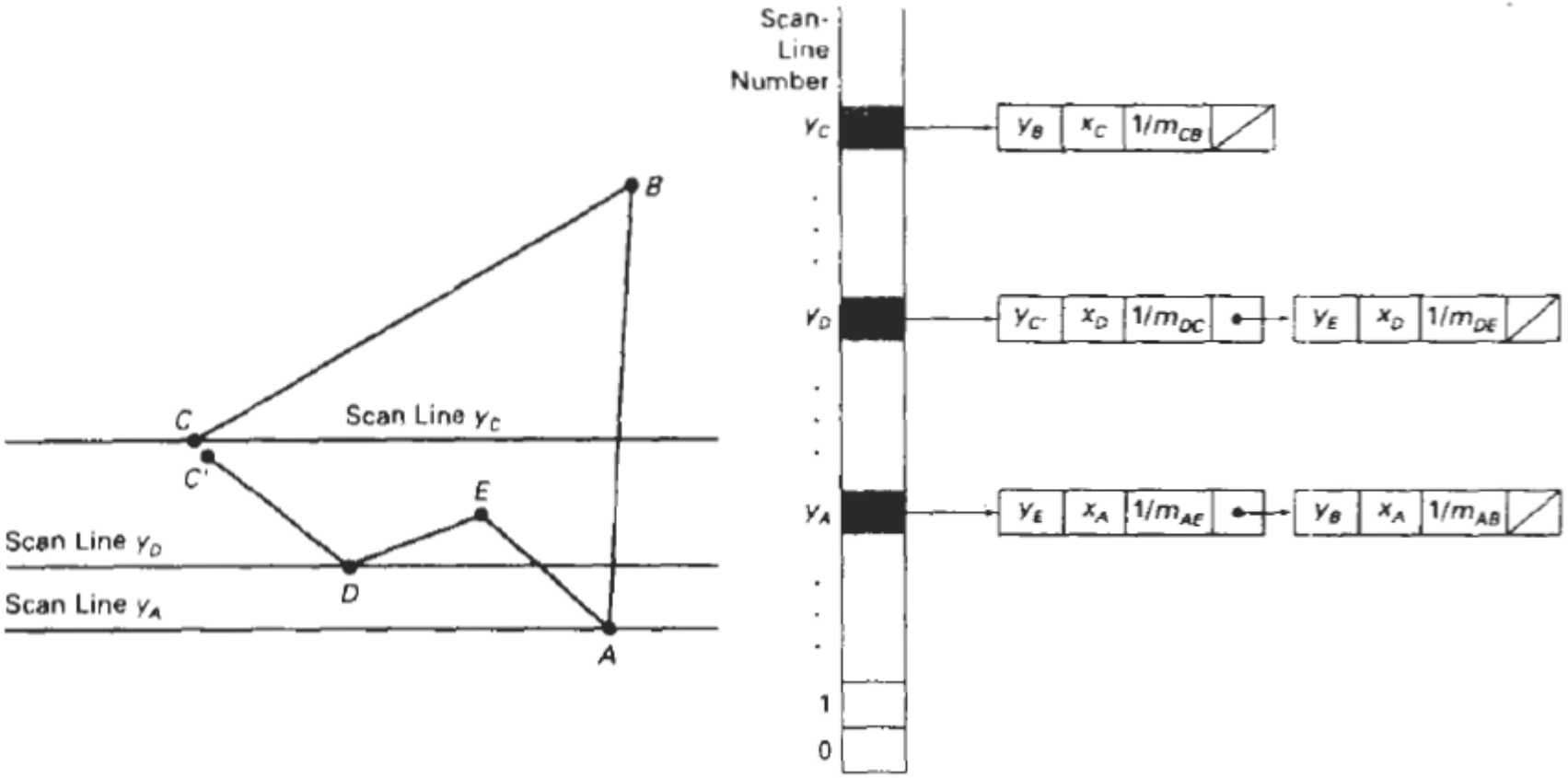
- Relleno mediante ordenamiento de aristas
- Relleno mediante complementación
- Relleno mediante complementación usando una cerca
- Algoritmo simple de siembra de semilla
- Siembra de semilla por línea de rastreo
- Funciones de OpenGL para manejo del color de las figuras y del fondo

Algoritmos de relleno de áreas
Relleno mediante ordenamiento de aristas



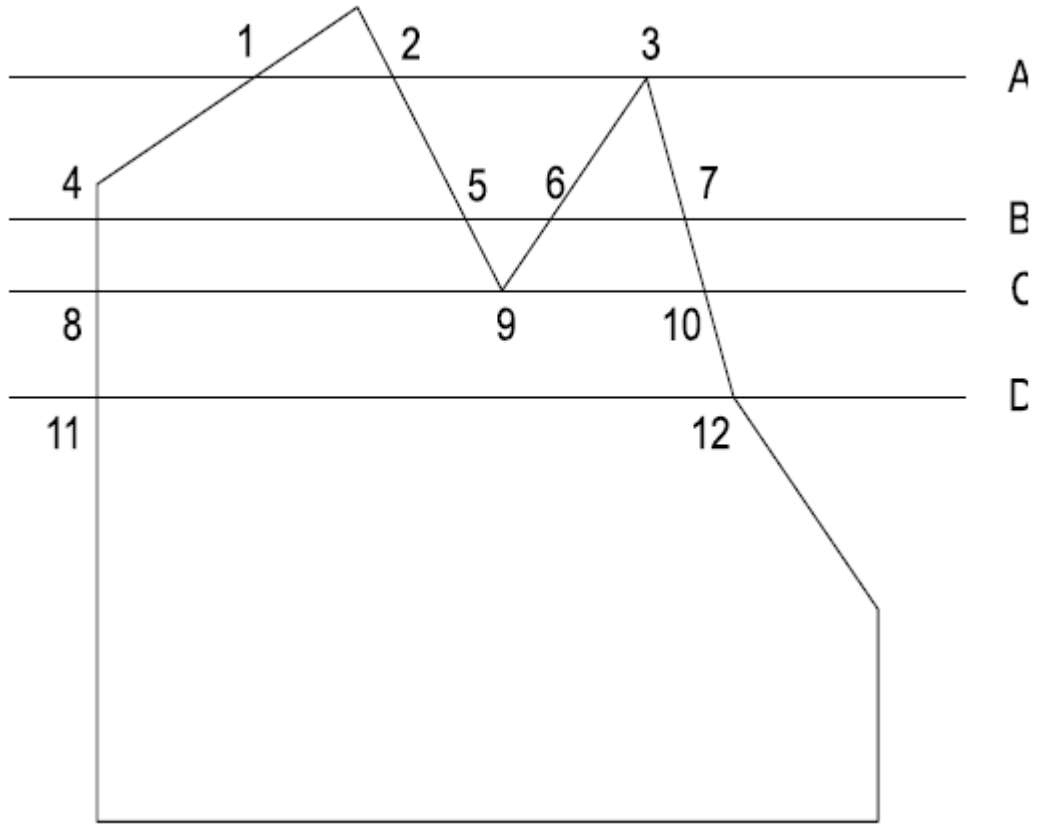
-Procesar por cada línea de rastreo
-ordenar aristas respecto a y y luego respecto a x
-Rellenar de la primera arista a la segunda, luego de la tercera a la cuarta , etc

Algoritmos de relleno de áreas
Relleno mediante ordenamiento de aristas



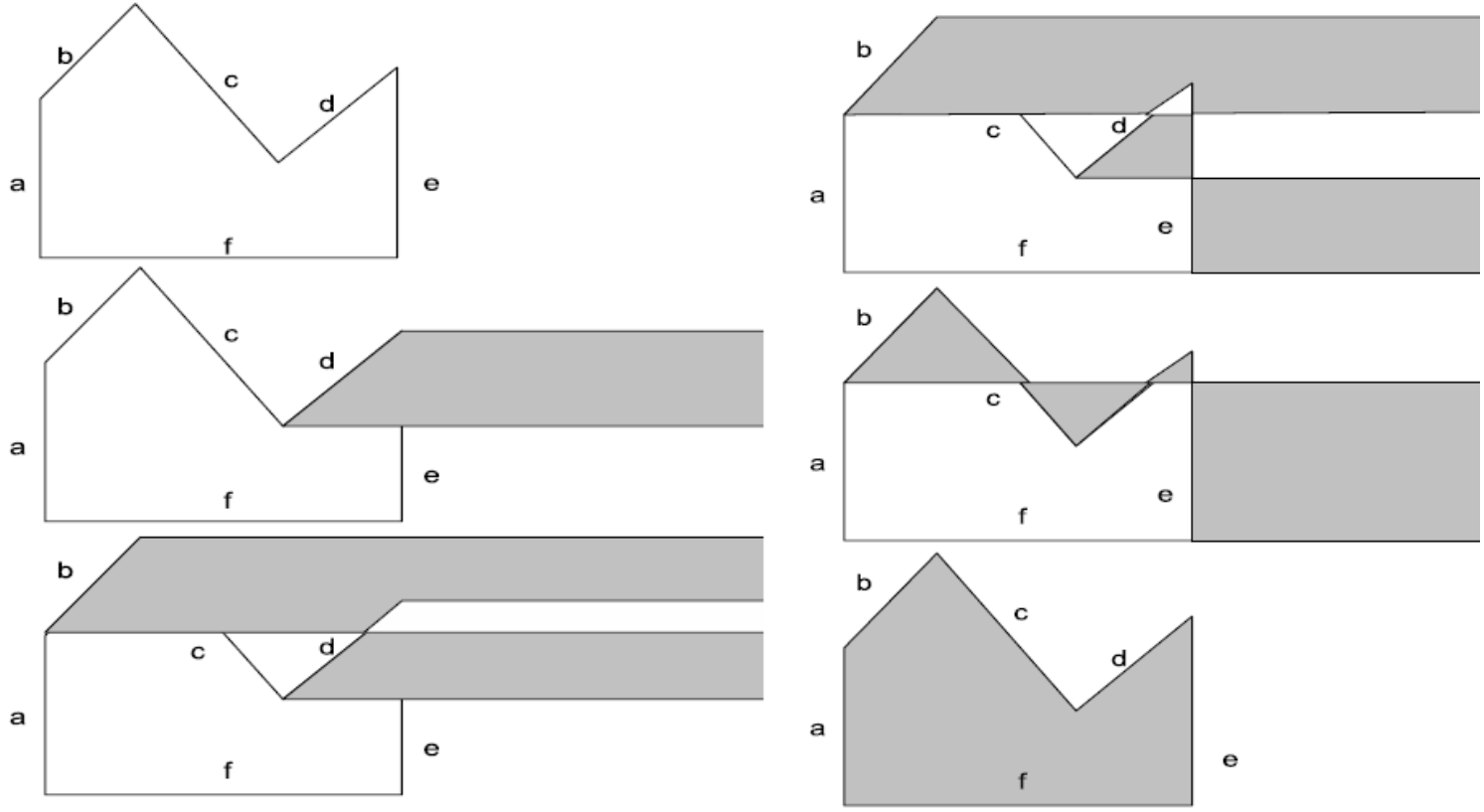
Mantener la lista de aristas ordenada en lugar de ordenarla al final

Algoritmos de relleno de áreas
Relleno mediante ordenamiento de aristas



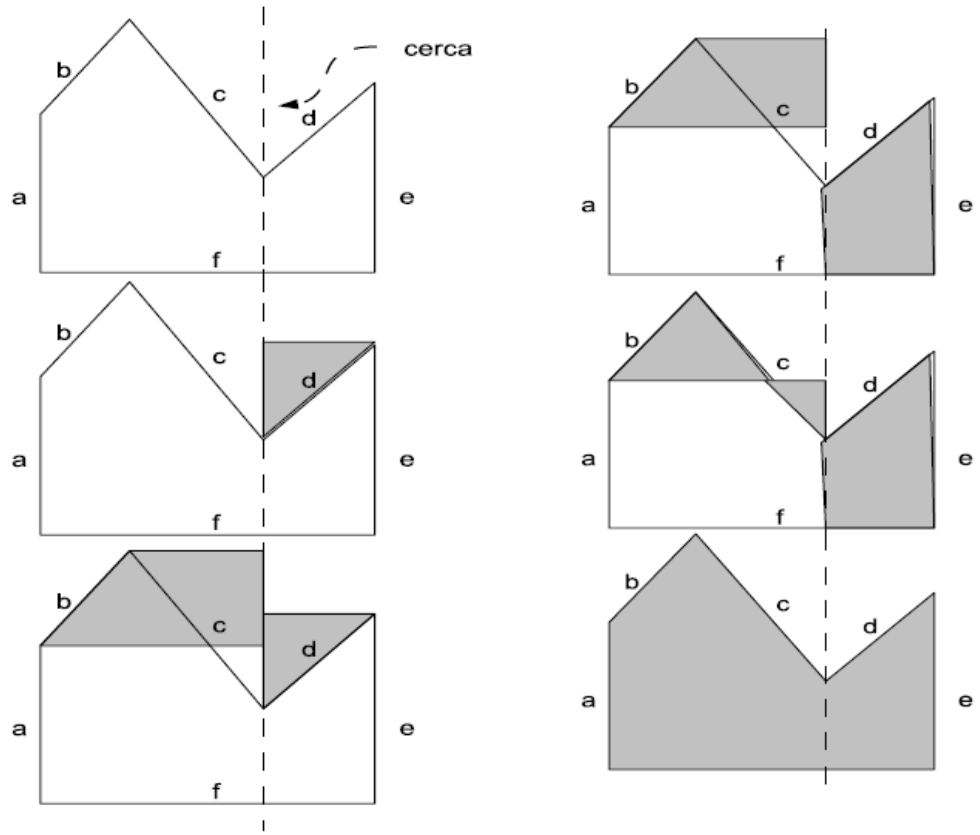
Eliminar duplicados correspondientes a los vértices que no son Máximos ni mínimos (Ej vértice 4 y 12)

Algoritmos de relleno de áreas
Relleno mediante complementación



En este ejemplo se procesaron las aristas en el sig. orden: d,b,e,c,a
Cualquier orden funciona
No se procesan las aristas horizontales

Algoritmos de relleno de áreas
Relleno mediante complementación
Modificación usando una cerca

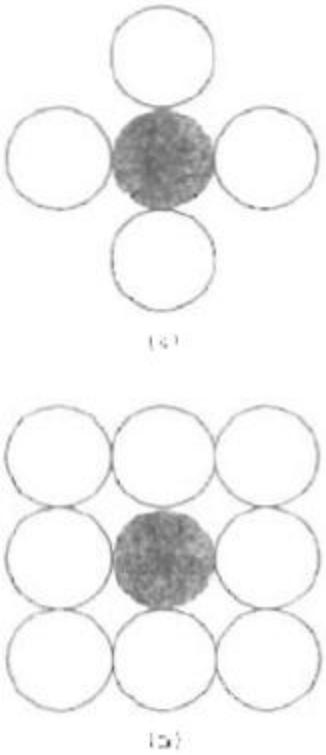


En este ejemplo se procesaron las aristas en el sig. orden: d,b,e,c,a

Cualquier orden funciona

No se procesan las aristas horizontales

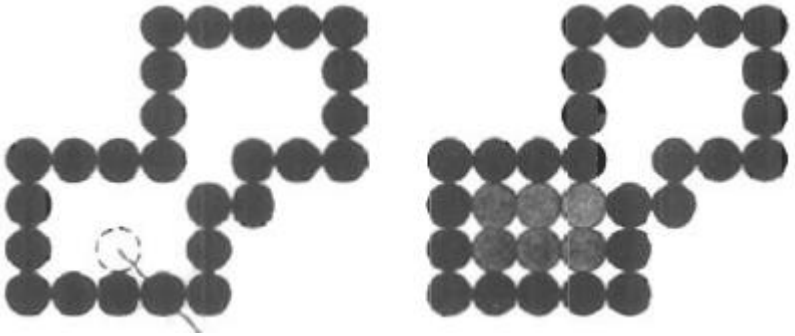
Algoritmos de relleno de áreas
Algoritmo simple de siembra de semilla



Conectividad 4

Conectividad 8

Algoritmos de relleno de áreas
Algoritmo simple de siembra de semilla



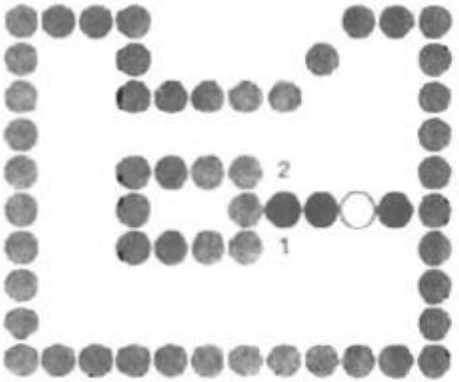
Semilla inicial

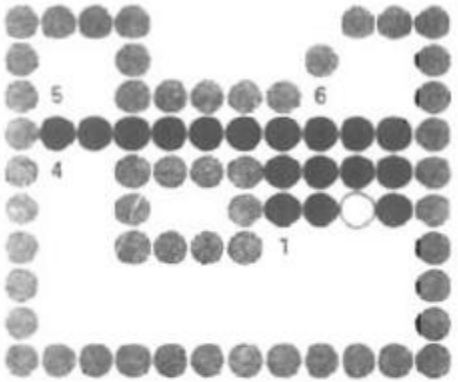
Se detiene porque se a
sumió conectividad 4

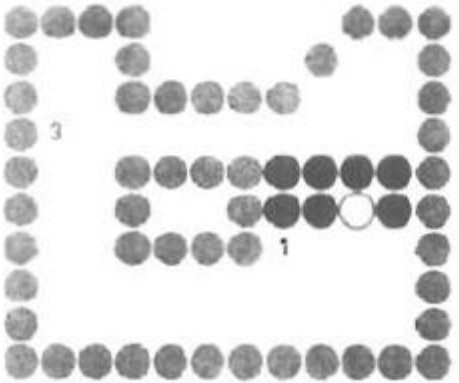
Algoritmos de relleno de áreas
Algoritmo simple de siembra de semilla

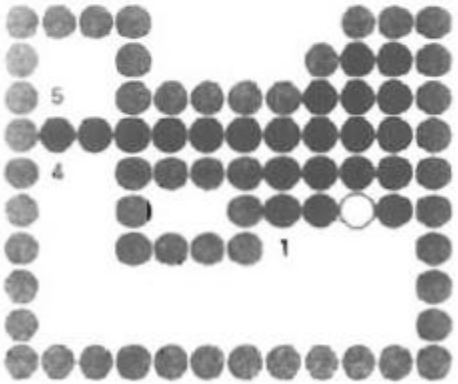
```
rellena(int x,int y) {
    setPixel(x,y,colorRelleno);
    color=getPixel(x+1,y);
    if ((color!=colorRelleno)&&(color!=colorBorde)) rellena(x+1,y);
    color=getPixel(x-1,y);
    if ((color!=colorRelleno)&&(color!=colorBorde)) rellena(x-1,y);
    color=getPixel(x,y+1);
    if ((color!=colorRelleno)&&(color!=colorBorde)) rellena(x,y+1);
    color=getPixel(x,y-1);
    if ((color!=colorRelleno)&&(color!=colorBorde)) rellena(x,y-1);
}
```

Algoritmos de relleno de áreas
Algoritmo de siembra de semilla por línea de rastreo









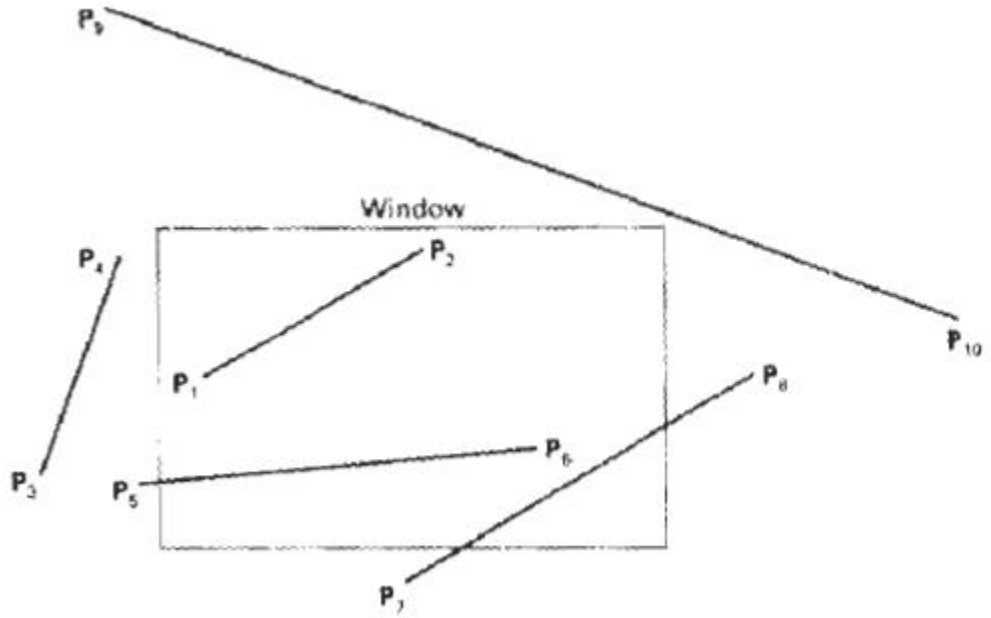
Algoritmos de relleno de área
Funciones de OpenGL para manejo del color de las figuras y del fondo

- void glColor3f(GLfoat red, GLfoat green, GLfoat blue)
- Una vez ejecutada esta instrucción las figuras que se dibujen tendrán este color
- void glColor4f(GLfoat red, GLfoat green, GLfoat blue, GLfoat alpha)
- Igual que la anterior pero tiene un parametro de transparencia
- void glClearColor3f(GLfoat red, GLfoat green, GLfoat blue)
- Para especificar el color del fondo

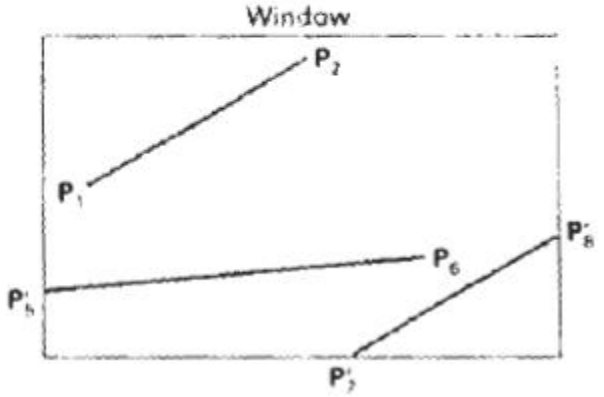
Algoritmos de recorte

- Códigos de región para determinar la visibilidad de líneas
- Algoritmo de recorte explícito en 2D
- Algoritmo de Sutherland-Cohen
- Algoritmo de la subdivisión del punto medio
- Algoritmo de Cyrus-Beck para recorte de regiones convexas

El concepto de recorte

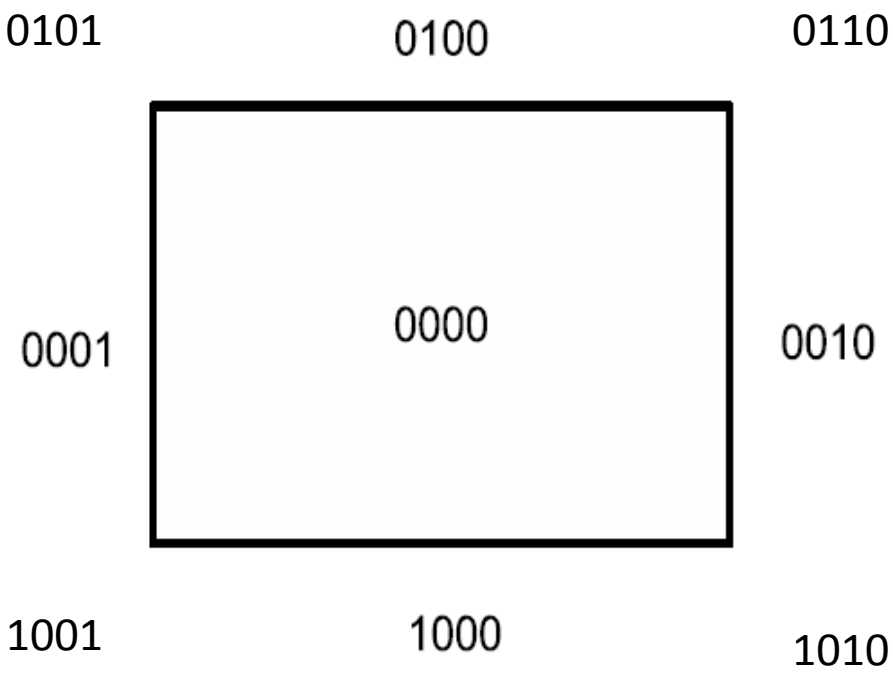


Antes del recorte



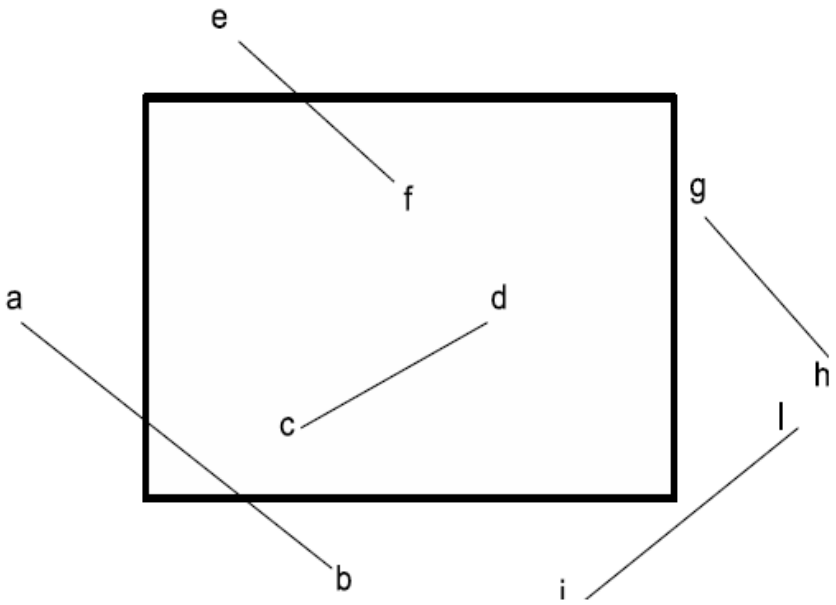
Después del recorte

Algoritmos de recorte
Códigos de Región para determinar la visibilidad de líneas



Bit 0 Izquierda
Bit 1 Derecha
Bit 2 Arriba
Bit 3 Abajo

Algoritmos de recorte
Códigos de Región para determinar la visibilidad de líneas



línea	Código 1	Código 2	AND	Observación
ab	0001	1000	0000	Parcialmente visible
cd	0000	0000	0000	Totalmente visible
ef	0100	0000	0000	Parcialmente visible
gh	0010	0010	0010	Trivialmente invisible
il	1000	0010	0000	Totalmente invisible

Algoritmos de recorte
Algoritmo de recorte explícito en 2D

Deduciendo la ec de la recta que va de (x1,y2) a (x2,y2)

$$y = mx + b$$

donde $m = \frac{y_2 - y_1}{x_2 - x_1}$

Como la línea pasa por (x_1, y_1), entonces:

$$y_1 = mx_1 + b$$

Por lo que:

$$b = y_1 - mx_1$$

Por tanto:

$$y = mx + y_1 - mx_1$$

Agrupando:

$$y = m(x - x_1) + y_1$$

Algoritmos de recorte
Algoritmo de recorte explícito en 2D

$y = y_{sup}$ extremo superior
$y = y_{inf}$ extremo inferior
$x = x_{izq}$ extremo izquierda
$x = x_{der}$ extremo derecho

De manera Similar

$$x = \frac{y}{m} - \frac{b}{m}$$

$$x = \frac{y}{m} - \frac{y_1 - mx_1}{m}$$

De donde:

$$x = \frac{y - y_1}{m} + x_1$$

$$(x_{der}, \frac{y_2 - y_1}{x_2 - x_1}(x_{der} - x_1) + y_1)$$

La intersección con el extremo izquierdo ocurre en:

$$(x_{izq}, \frac{y_2 - y_1}{x_2 - x_1}(x_{izq} - x_1) + y_1)$$

Algoritmos de recorte
Algoritmo de recorte explícito en 2D

La intersección con el extremo superior ocurre en:

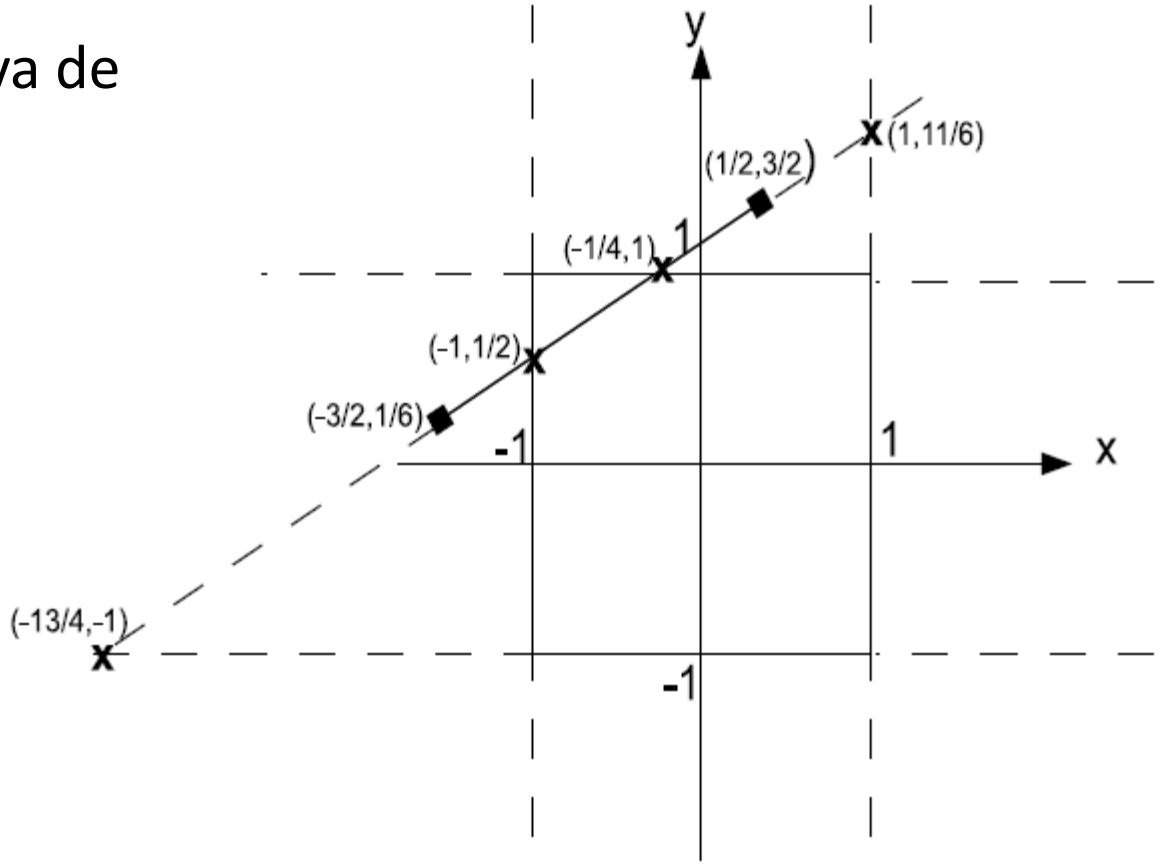
$$((y_{sup} - y_1)\frac{x_2 - x_1}{y_2 - y_1} + x_1, y_{sup})$$

La intersección con el extremo inferior ocurre en:

$$((y_{inf} - y_1)\frac{x_2 - x_1}{y_2 - y_1} + x_1, y_{inf})$$

Algoritmos de recorte
Algoritmo de recorte explícito en 2D. Ejemplo

Recorte la línea que va de (-3/2,1/6) a (1/2,3/2)



Algoritmos de recorte
Algoritmo de recorte explícito en 2D. Ejemplo

La intersección con el extremo derecho ocurre en:

$$\left(1, \frac{3/2 - 1/6}{1/2 - (-3/2)}(1 - (-3/2)) + 1/6\right) = \left(1, (\frac{2}{3})(\frac{5}{2}) + \frac{1}{6}\right) = \left(1, \frac{11}{6}\right)$$

la cual se descarta porque ocurre mas a la derecha del extremo derecho de la línea $(11/6 > 1)$. La intersección con el extremo izquierdo ocurre en:

$$\left(-1, \frac{3/2 - 1/6}{1/2 - (-3/2)}(-1 - (-3/2)) + 1/6\right) = \left(-1, (\frac{2}{3})(\frac{1}{2}) + \frac{1}{6}\right) = \left(-1, \frac{1}{2}\right)$$

La intersección con el extremo superior ocurre en:

$$\left((1 - 1/6)\frac{1/2 - (-3/2)}{3/2 - 1/6} + (-3/2), 1\right) = \left((\frac{5}{6})(\frac{3}{2}) - \frac{3}{2}, 1\right) = \left(-\frac{1}{4}, 1\right)$$

Algoritmos de recorte
Algoritmo de recorte explícito en 2D. Ejemplo

Finalmente, la intersección con el extremo inferior ocurre en:

$$\left((-1-1/6)\frac{1/2-(-3/2)}{3/2-1/6}+(-3/2),-1\right)=\left((-\frac{7}{6})(\frac{3}{2})-\frac{3}{2},-1\right)=\left(-\frac{13}{4},-1\right)$$

El cual también se descarta porque ocurre mas a la izquierda del principio de la línea ($-13/4 < -3/2$). Los puntos de intersección no descartados nos indican que la línea recortada va de (-1,1/2) a (-1/4,1).

Algoritmos de recorte
Algoritmo de Sutherland-Cohen

- Por cada extremo de la ventana rectangular de recorte efectuar los pasos (a), (b) y (c)

  - (a) Para la línea P_1P_2, determine si la línea es totalmente visible o si puede ser descartada como trivialmente invisible

  - (b) Si P_1 está fuera de la ventana de recorte continúa, de lo contrario intercambia P_1 y P_2

  - (c) Reemplaza P_1 por la intersección de P_1P_2 con el extremo en turno

Algoritmos de recorte
Algoritmo de la subdivisión del punto medio

Ideal para implementar en hardware

```
Recorta(P1,P2)
if P1P2 es una línea trivialmente invisible then
    Descarta el segmento de línea (P1,P2);
    Termina;
end
if P1P2 es una línea totalmente visible then
    Incluye como visible el segmento de línea (P1,P2);
    Termina;
end
Pm = (P1 + P2)/2
Recorta(P1Pm);
Recorta(PmP2);
```

Algoritmos de recorte
Algoritmo de Cyrus-Beck para recorte de regiones convexas

El Algoritmo de Cyrus-Beck hace uso del hecho de que un punto *a* está dentro de un área de recorte convexa respecto a cierta área que define un borde si se cumple la desigualdad:

$$n \cdot (b - a) > 0$$

donde *n es un vector normal a la línea que define el borde y b es cualquier* punto en ese borde

El Algoritmo de Cyrus-Beck hace uso de la ecuación paramétrica de una línea recta que va de *P1 a P2*

$$P(t) = P_1 + (P_2 - P_1)t$$

Algoritmos de recorte
Algoritmo de Cyrus-Beck para recorte de regiones convexas

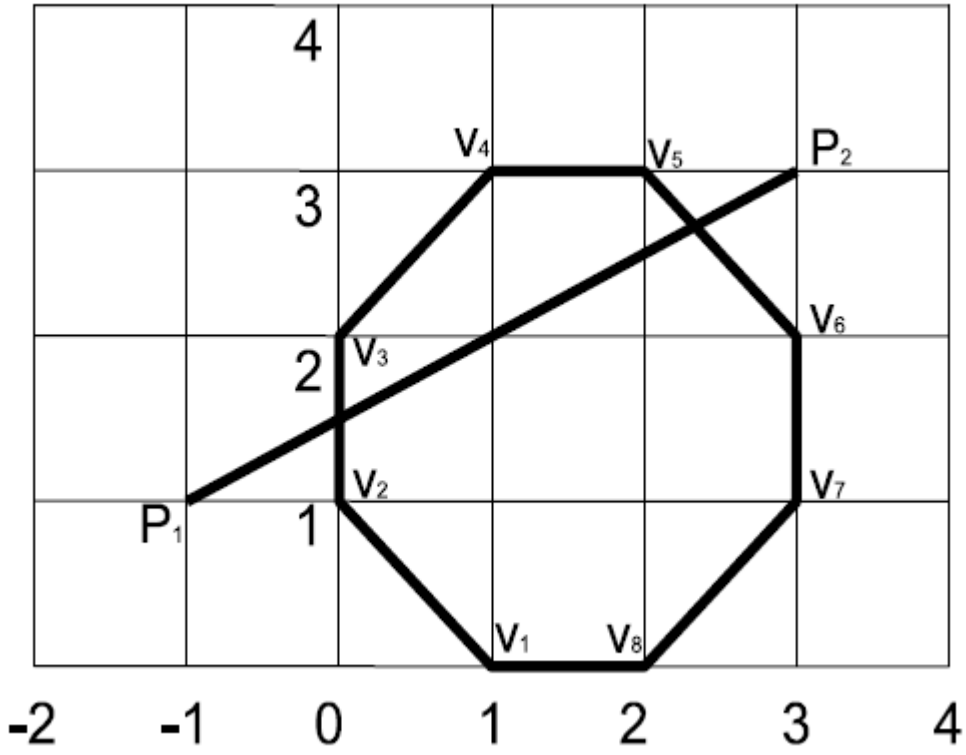
podemos obtener el valor de *t para el cual la línea coin*cide con la frontera del área de recorte despejándolo de:

$$n \cdot (P(t) - f) = 0$$

Donde f es cualquiér punto de esa frontera

Algoritmos de recorte
Algoritmo de Cyrus-Beck para recorte de regiones convexas. Ejemplo

Recortar la línea que va de (-1,1) a (3,3)



Algoritmos de recorte
Algoritmo de Cyrus-Beck para recorte de regiones convexas. Ejemplo

La directríz es: $D = P_2 - P_1 = \begin{bmatrix} 3 \\ 3 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$

Para la arista V5V6 la normal es:

$$n = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

Entonces:

$$n \cdot D = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 2 \end{bmatrix} = -6 \leq 0$$

Algoritmos de recorte
Algoritmo de Cyrus-Beck para recorte de regiones convexas. Ejemplo

tomando a $f = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ como un punto que forma parte de la arist

determinamos w:

$$w = P_1 - f = \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \end{bmatrix}$$

De ahí que:

$$n \cdot w = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \cdot \begin{bmatrix} -3 \\ -2 \end{bmatrix} = 5$$

Finalmente:

$$t = -\frac{n \cdot w}{n \cdot D} = -\frac{5}{-6} = \frac{5}{6}$$

Lo cual significa que esta arista intersecta a la línea en:

$$P(\frac{5}{6}) = P_1 + (P_2 - P_1)\frac{5}{6} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \left(\begin{bmatrix} 3 \\ 3 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right)\frac{5}{6} = \begin{bmatrix} 7/3 \\ 8/3 \end{bmatrix}$$

Algoritmos de recorte
Algoritmo de Cyrus-Beck para recorte de regiones convexas. Ejemplo

Repitiendo para el
Resto de las aristas
Completamos la tabla

Arista	n	f	w	$n \cdot w$	$n \cdot D$	t_i	t_f
V_1V_2	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -2 \\ 1 \end{bmatrix}$	-1	6	$\frac{1}{6}$	
V_2V_3	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 2 \end{bmatrix}$	$\begin{bmatrix} -1 \\ -1 \end{bmatrix}$	-1	4	$\frac{1}{4}$	
V_3V_4	$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 2 \end{bmatrix}$	$\begin{bmatrix} -1 \\ -1 \end{bmatrix}$	0	2	0	
V_4V_5	$\begin{bmatrix} 0 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 3 \end{bmatrix}$	$\begin{bmatrix} -3 \\ -2 \end{bmatrix}$	2	-2		1
V_5V_6	$\begin{bmatrix} -1 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 3 \end{bmatrix}$	$\begin{bmatrix} -3 \\ -2 \end{bmatrix}$	5	-6		$\frac{5}{6}$
V_6V_7	$\begin{bmatrix} -1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 3 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -4 \\ 0 \end{bmatrix}$	4	-4		1
V_7V_8	$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 3 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -4 \\ 0 \end{bmatrix}$	4	-2		2
V_8V_1	$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -2 \\ 1 \end{bmatrix}$	1	2	$-\frac{1}{2}$	

Para terminar el ejemplo solo resta decir que la línea recortada comienza en *t = 1=4 y termina en t = 5=6, lo cual corresponde a una nueva línea que* va de (0,3/2) a (7/3,8/3)

Pipeline de visualización bidimensional

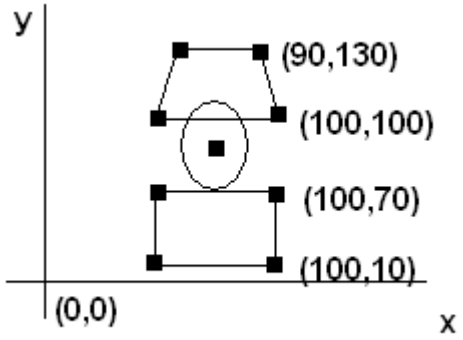
- Coordenadas locales
- Coordenadas mundiales
- Puerto de visión
- Funciones OpenGL para visualización 2D

Pipeline de visualización bidimensional Coordenadas locales

- También se llaman “coordenadas de modelado”
- Una figura se define en coordenadas relativas a un punto de la misma figura, por ejemplo el centro de la misma
- Por ejemplo los vértices de un rectángulo pudieran establecerse relativos a la esquina inferior e izquierda del mismo

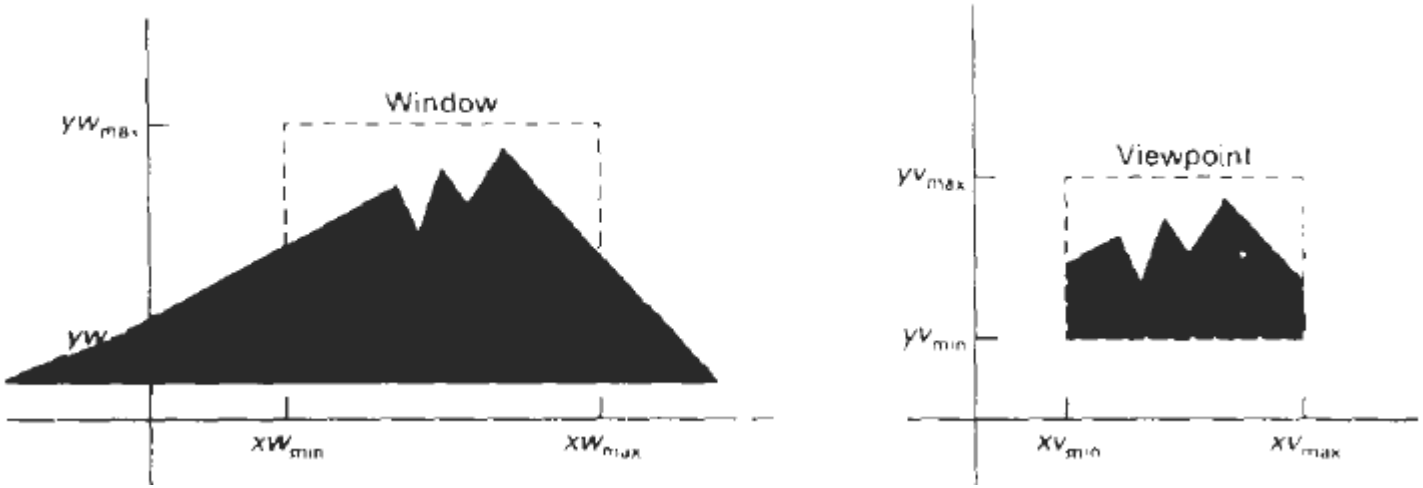
Pipeline de visualización bidimensional Coordenadas mundiales

- Todas las figuras que forman un dibujo deben transformarse a un único sistema de coordenadas.



Pipeline de visualización bidimensional puerto de visión

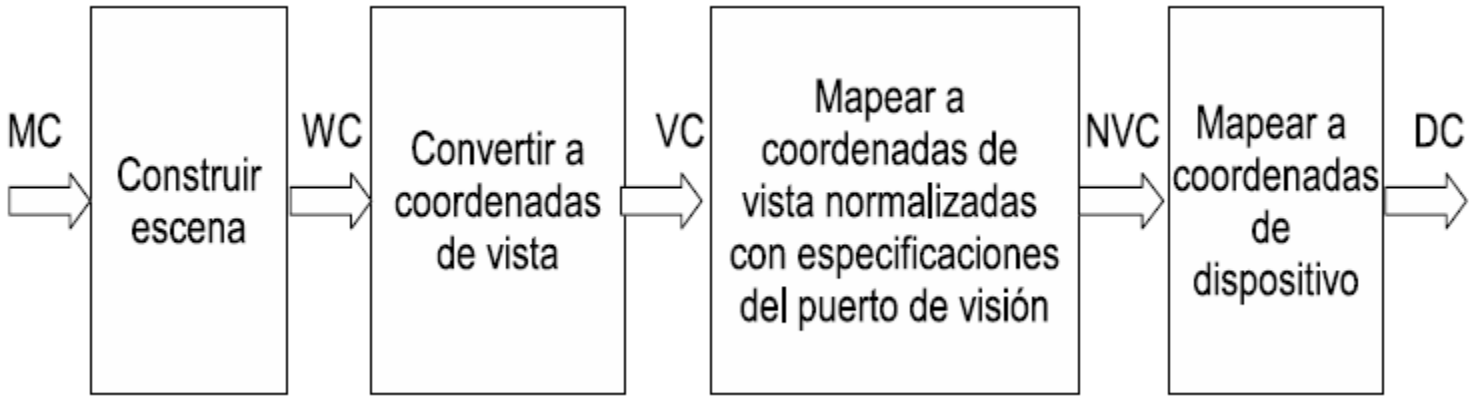
- La parte de una gráfica que se desea desplegar se denomina “ventana”
- La ventana se especifica en coordenadas mundiales
- El puerto de visión especifica donde se desplegará el contenido de la ventana
- El puerto de visión se especifica en “coordenadas de dispositivo”



Pipeline de visualización bidimensional puerto de visión

- Manteniendo fijo el tamaño de la ventana y aumentando el tamaño del puerto de visión se obtiene un efecto de “zoom in” (acercamiento)
- Manteniendo fijo el puerto de visión y cambiando la ventana de posición se obtiene un efecto de “paneo” de diferentes partes de una gráfica
- Se requiere una transformación de ventana a puerto de visión

Pipeline de visualización bidimensional



Pipeline de visualización bidimensional
Funciones OpenGL de visualización bidimensional

- Para especificar la ubicación y tamaño del puerto de visión en OpenGL usamos la función
- glViewport(int x, int y, int ancho, int alto)
- La función gluOrtho2D sirve para poder especificar directamente en coordenadas mundiales al dibujar, por ejemplo:
- gluOrtho2D(0.0,450.0,0.0,375.0)
- Nos permite especificar vértices (x,y) tales que

x esté en el rango [0,450] y y en [0,375]

Transformaciones geométricas

- Transformaciones afines
- Transformaciones geométricas bidimensionales básicas: Traslación, escalamiento, rotación
- Coordenadas homogéneas
- Transformaciones compuestas: Escalamiento respecto a un punto fijo; Rotación respecto a un punto arbitrario
- Reflexiones
- Transformaciones geométricas en 3D simples: Escalamiento, traslación, rotación respecto a los ejes X, Y y Z
- Rotación respecto a un eje arbitrario por: (a) Composición de matrices, (b) Cjto de vectores ortogonales, (c) cuaternios
- Transformaciones geométricas con OpenGL
- Manejo de Pilas de matrices con OpenGL

Transformaciones geométricas
Transformaciones afines

- Una transformación afín es aquella que puede expresarse de la siguiente manera:

$$Q_x = a.P_x + b.P_y + t_x$$
$$Q_y = c.P_x + d.P_y + t_y$$

O bien:

$$\begin{pmatrix} Q_x \\ Q_y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} P_x \\ P_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Transformaciones geométricas
Transformaciones afines

- Las transformaciones afines son lineales
- si se aplica la transformación a todos los puntos que forman una línea se obtiene otra línea que también se puede obtener aplicando la transformación solo a los extremos de la línea y se generan los puntos intermedios mediante algún algoritmo de trazado de líneas como el de Bresenham
- si se transforma un punto que está a la mitad de la línea, este punto de ubicarla justamente a la mitad de la línea transformada

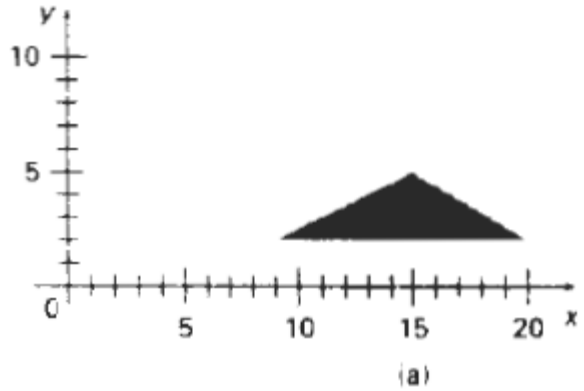
Transformaciones geométricas
Transformaciones afines

- Ejemplos de transformaciones afines:
- Traslación
- Escalamiento
- Rotación
- Reflexión
- Cizallamiento

Transformaciones geométricas
transformaciones geométricas bidimensionales básicas
Traslación

Trasladamos un punto (x,y) mediante:

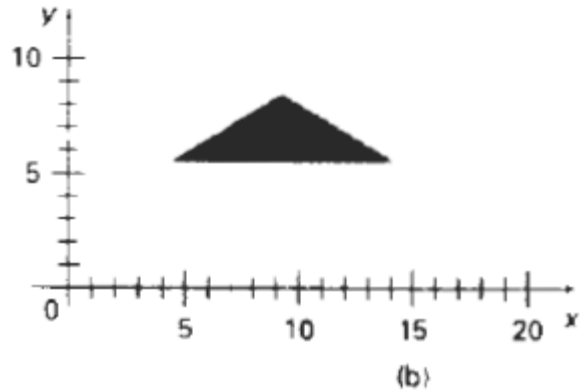
$$x' = x + t_x$$
$$y' = y + t_y$$



(a)

En forma vectorial

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

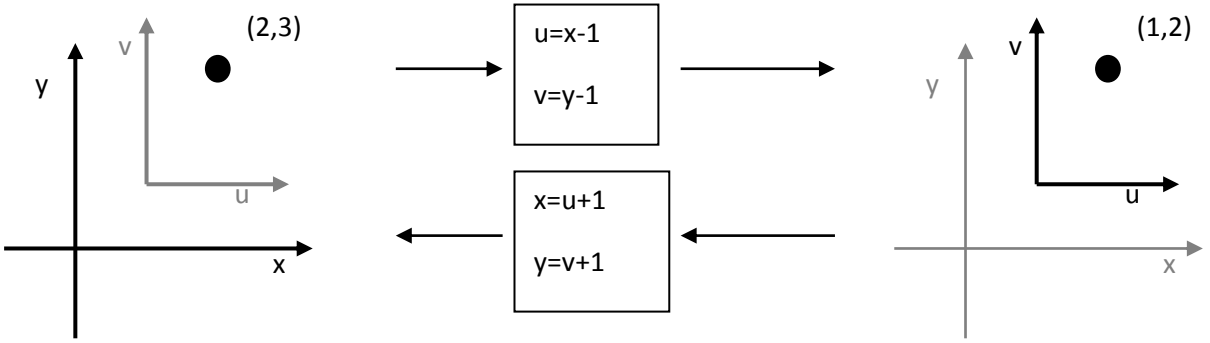


(b)

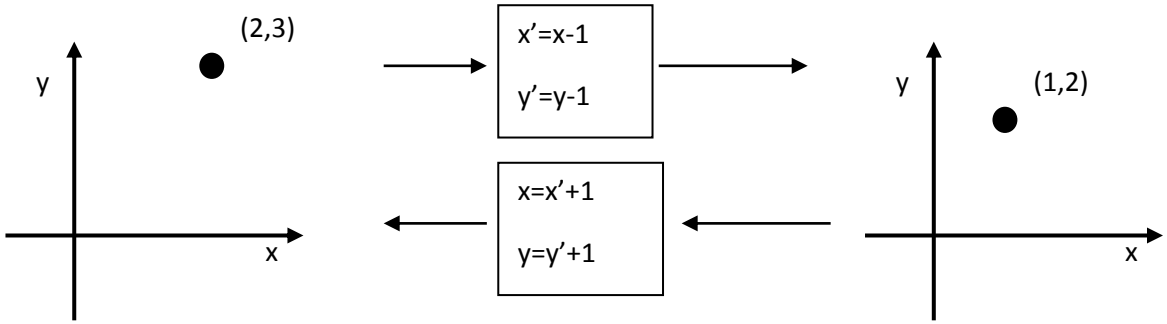
Transformaciones geométricas
transformaciones geométricas bidimensionales básicas
Traslación

- Toda transformación geométrica tiene dos interpretaciones:

Pasar de un sistema de Coordenadas a otro



Modificar la forma o posición de objetos en un sistema de coordenadas



Transformaciones geométricas
transformaciones geométricas bidimensionales básicas
Traslación

Escalamos mediante la operación:

$$x' = S_x x$$
$$y' = S_y y$$

En forma matricial

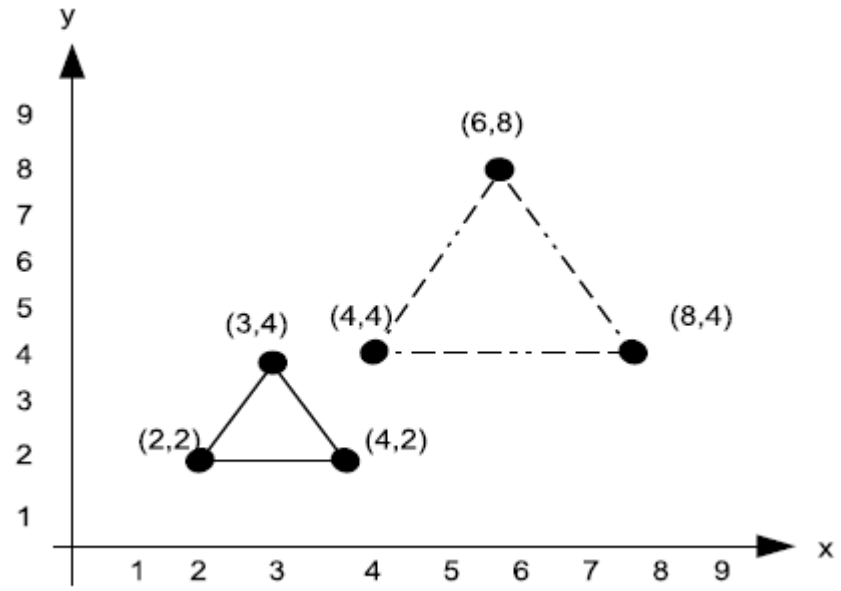
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

que en forma compacta puede representarse como:

$$p' = Sp$$

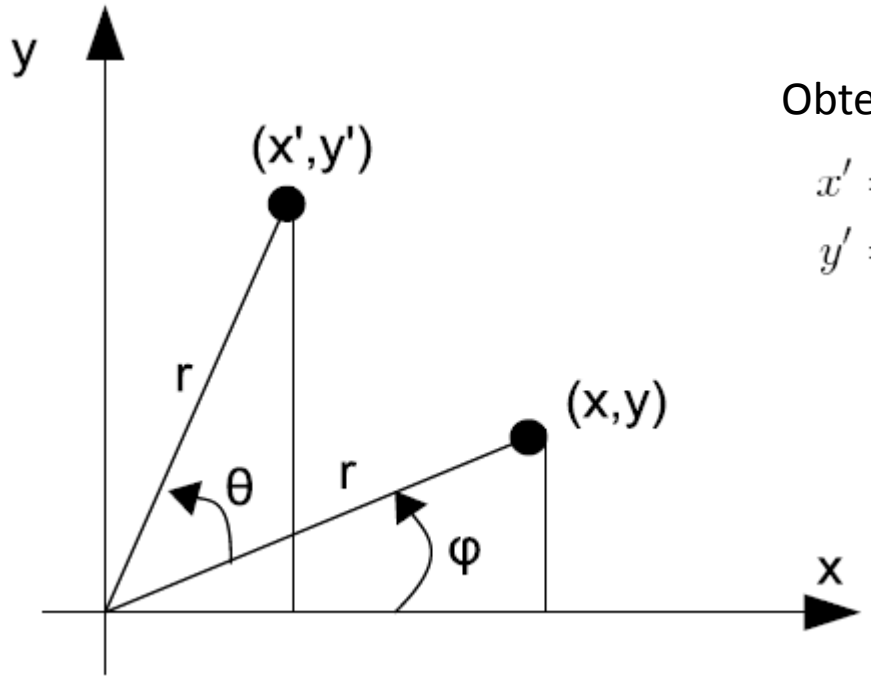
donde S es la matriz de escalamiento.

Transformaciones geométricas
transformaciones geométricas bidimensionales básicas
Traslación



El escalamiento simple tiene implícita una traslación. En el ejemplo se usó un factor de escalamiento de 2 tanto en x como en y

Transformaciones geométricas
transformaciones geométricas bidimensionales básicas
Rotación



Obtenemos:

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

De la figura:

$$x = r\ cos(\phi)$$
$$y = r\ sen(\phi)$$

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

Finalmente!

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

recordando las identidades trigonométricas:

$$cos(\alpha + \beta) = cos(\alpha)cos(\beta) - sen(\alpha)sen(\beta)$$
$$sen(\alpha + \beta) = cos(\alpha)sen(\beta) + sen(\alpha)cos(\beta)$$

Transformaciones geométricas
transformaciones geométricas bidimensionales básicas
Rotación

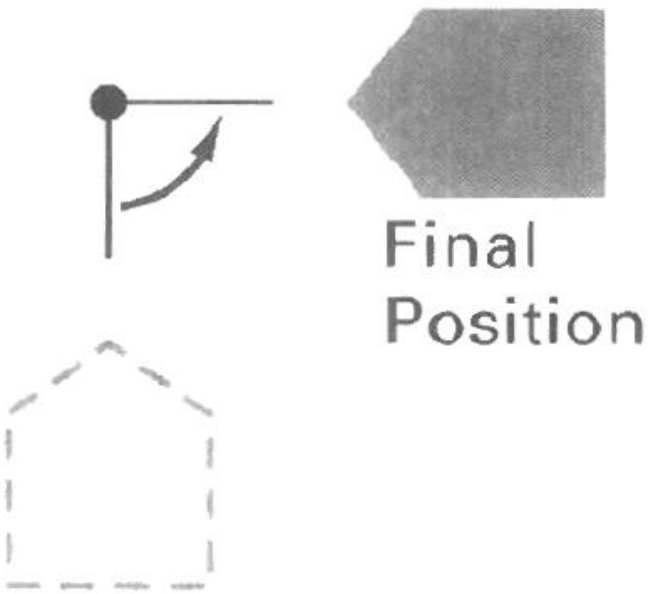
$$x' = x\ cos(\theta) - y\ sen(\theta)$$
$$y' = y\ cos(\theta) + x\ sen(\theta)$$

Se puede expresar matricialmente como:

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

y en forma compacta:

$$p' = Rp$$



La rotación también implica una traslación

Transformaciones geométricas
Coordenadas homogéneas

- Las coordenadas (x,y) tiene un número infinito de maneras de representarse en coordenadas homogéneas (xh,yh,h)
- La forma estándar h=1, es decir (x,y,1)
- Ventajas:

1. Todas las transformaciones se pueden hacer como productos matriciales, incluso la traslación
2. Facilita la composición de transformaciones. Muchas transformaciones sucesivas se convierten en una sola (¡gran ahorro!)

Transformaciones geométricas
Coordenadas homogéneas

Todas las transformaciones como productos matriciales:

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

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

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

Transformaciones geométricas
Transformaciones compuestas
Escalamiento respecto a un punto fijo

Traslación de manera que el punto fijo coincida con el origen

Escalamiento simple

Traslación de manera que el punto fijo regrese a su posición

Transformaciones geométricas
Transformaciones compuestas
Escalamiento respecto a un punto fijo

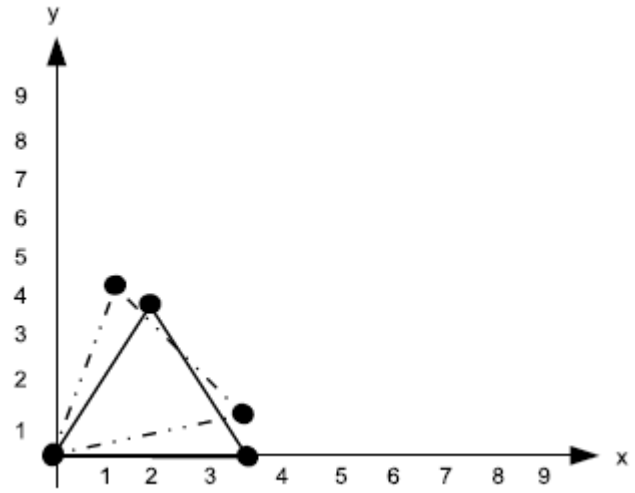
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & -S_x x_f \\ 0 & S_y & -S_y y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

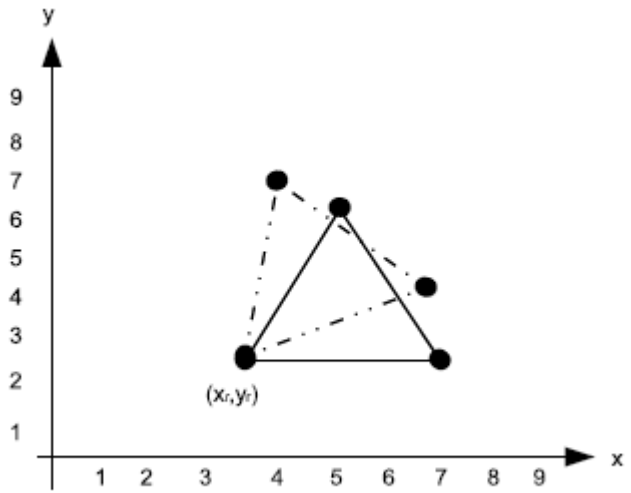
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & x_f(1-S_x) \\ 0 & S_y & y_f(1-S_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Ahora se aprecia la ventaja de trabajar en coordenadas homogéneas

Transformaciones geométricas
Transformaciones compuestas
Rotación respecto a un punto arbitrario



Trasladar de manera que el punto de rotación coincida con el origen



Rotar y luego Trasladar de manera que el punto de rotación regrese a su posición original

Transformaciones geométricas
Transformaciones compuestas
Rotación respecto a un punto arbitrario

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

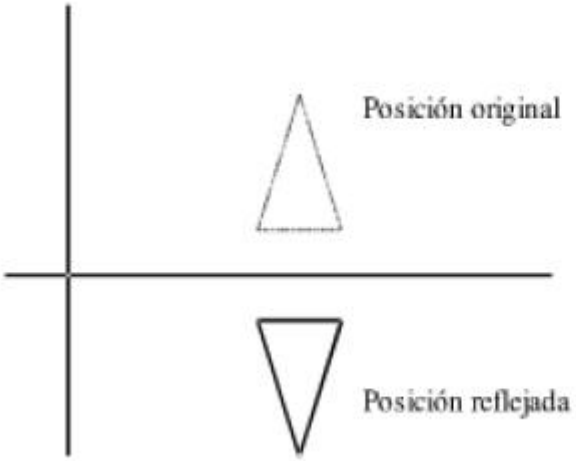
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sen\theta & y_r sen\theta - x_r cos\theta \\ sen\theta & cos\theta & -x_r sen\theta - y_r cos\theta \\ 0 & 0 & 1 \end{bmatrix}$$

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

De nuevo se aprecia la ventaja de trabajar en coordenadas homogéneas

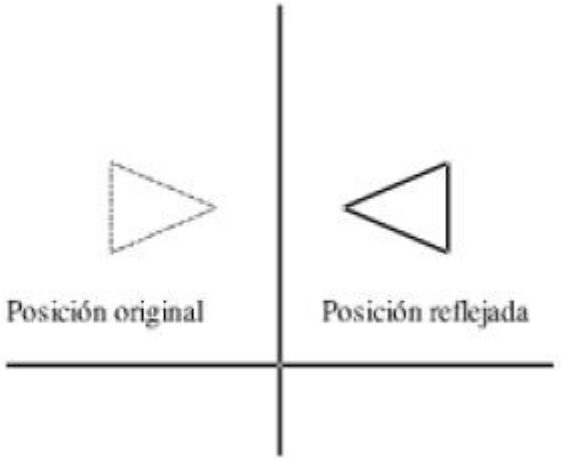
Transformaciones geométricas
Reflexiones

Reflexión respecto al eje x



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

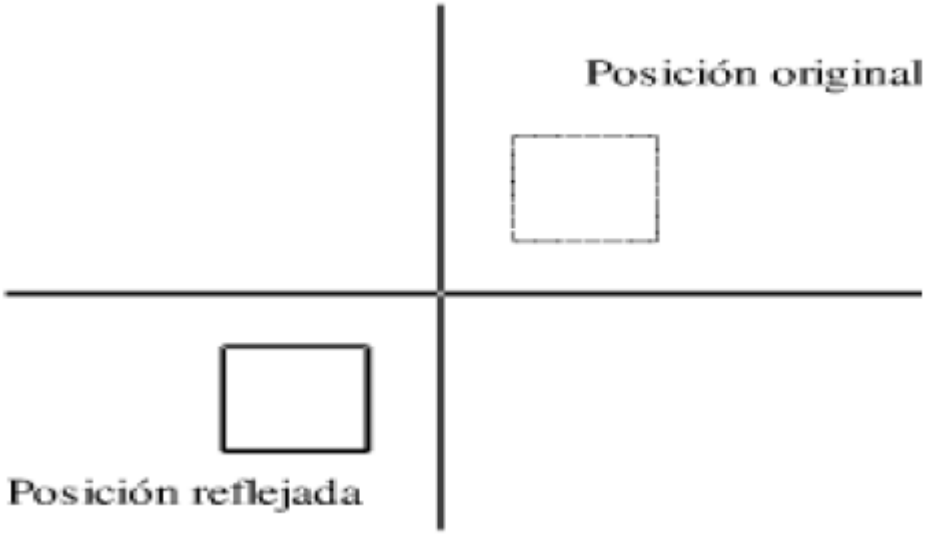
Reflexión respecto al eje y



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

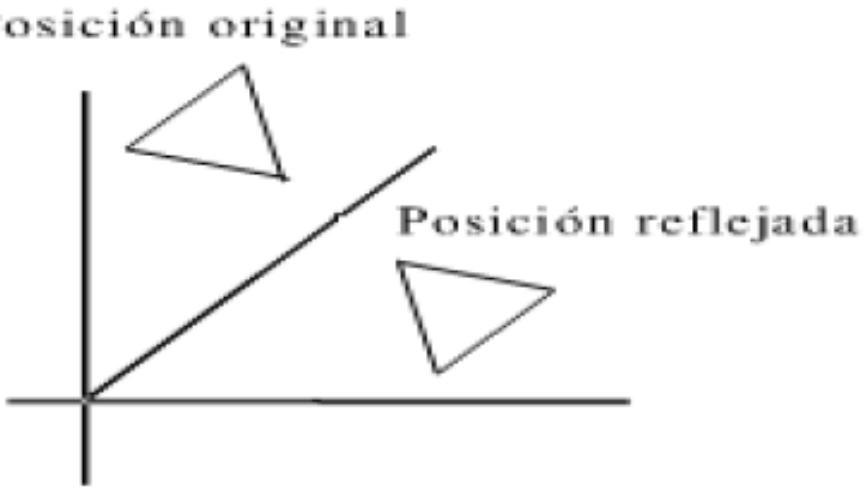
Transformaciones geométricas
Reflexiones

Reflexión respecto al origen



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

Reflexión respecto a la recta x=y



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

Transformaciones geométricas
Transformaciones geométricas en 3D simples
Escalamiento

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

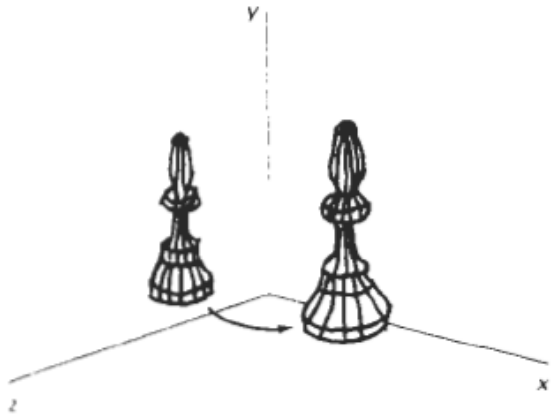
Transformaciones geométricas
Transformaciones geométricas en 3D simples
Traslación

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

Transformaciones geométricas en 3D simples
Rotación alrededor del eje X, Y o Z

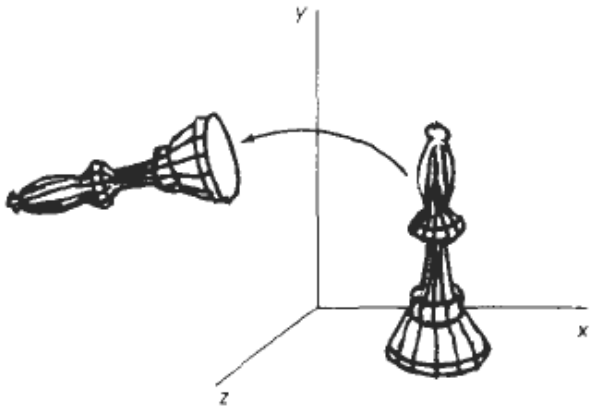
Rotación alrededor del eje X

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



Rotación alrededor del eje Y

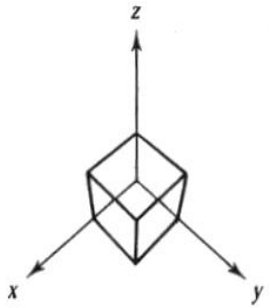
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & -sen\theta & 0 \\ 0 & 1 & 0 & 0 \\ sen\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Rotación alrededor del eje Z

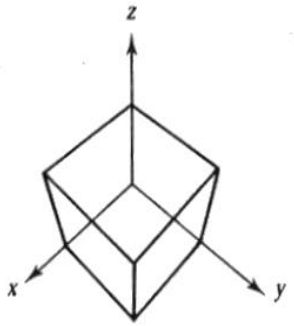
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta & 0 & 0 \\ sen\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Transformaciones geométricas en 3D simples
Ejemplos



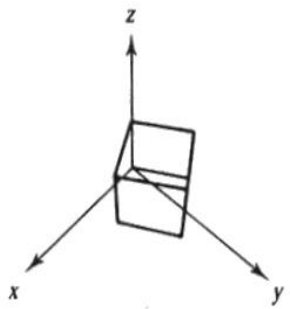
(a) Identity

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



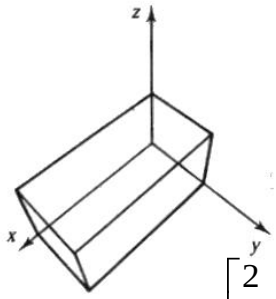
(b) Uniform scaling

$$\begin{bmatrix} 1.5 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



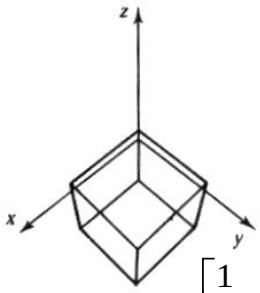
(c) Rotation (*z*-axis)

$$\begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



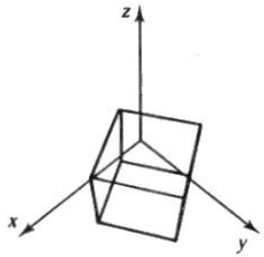
(d) *X* scaling

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(e) Translation

$$\begin{bmatrix} 1 & 0 & 0 & 500 \\ 0 & 1 & 0 & 500 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



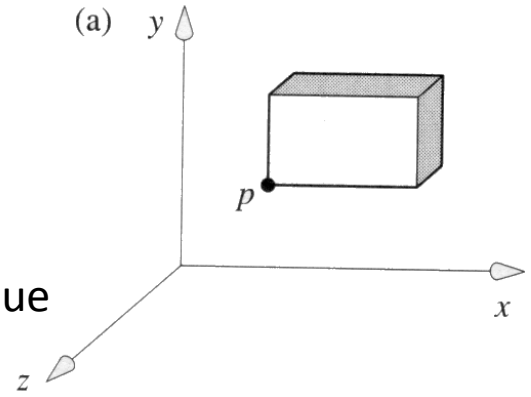
(f) Translation and rotation

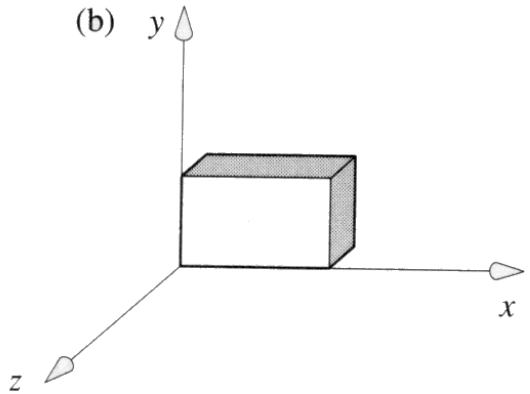
$$\begin{bmatrix} 0.866 & -0.5 & 0 & 430.7 \\ 0.5 & 0.866 & 0 & 60.3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

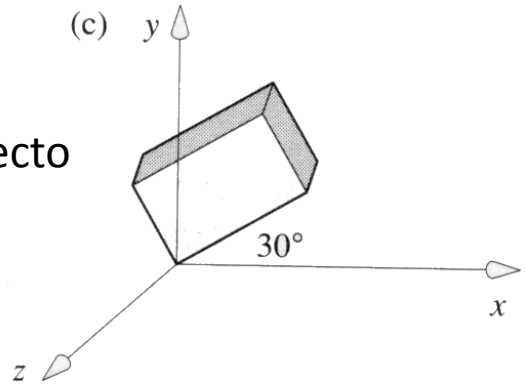
Transformaciones geométricas
Rotación respecto a un eje arbitrario

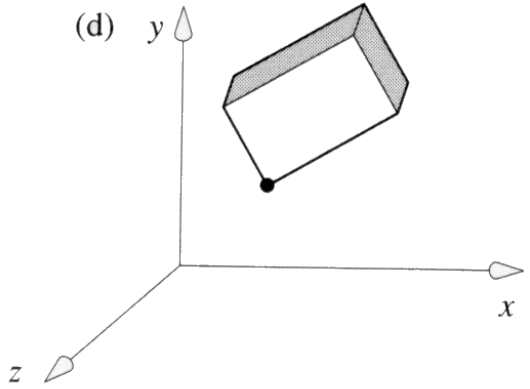
Si el eje es paralelo a uno
De los ejes del sistema de
Coordenadas:

1) Trasladar de manera que
Un punto que pasa por el
Eje de rotación coincida
Con el origen
2) Rotar alrededor del eje
3) Traslación inversa respecto
Al paso 1









Rotación respecto a un eje arbitrario
Determinación de la matriz de transformación por composición de matrices

1. Trasladar el punto de manera que el eje de rotación pase por el origen
2. Rotar el punto alrededor del eje x un ángulo α tal que el eje de rotación resida en el plano xz
3. Rotar el punto alrededor del eje y un ángulo β tal que el eje de rotación coincida con el eje z
4. Rotar el punto alrededor del eje z un ángulo θ
5. Rotar el punto alrededor del eje y un ángulo $-\beta$
6. Rotar el punto alrededor del eje x un ángulo $-\alpha$
7. Trasladar el punto de manera que el eje de rotación regrese a su lugar

Rotación respecto a un eje arbitrario
Determinación de la matriz de transformación por composición de matrices

$$u' \cdot u_z = |u'||u_z|cos(\alpha) = (0, b, c) \cdot (0, 0, 1) = c$$

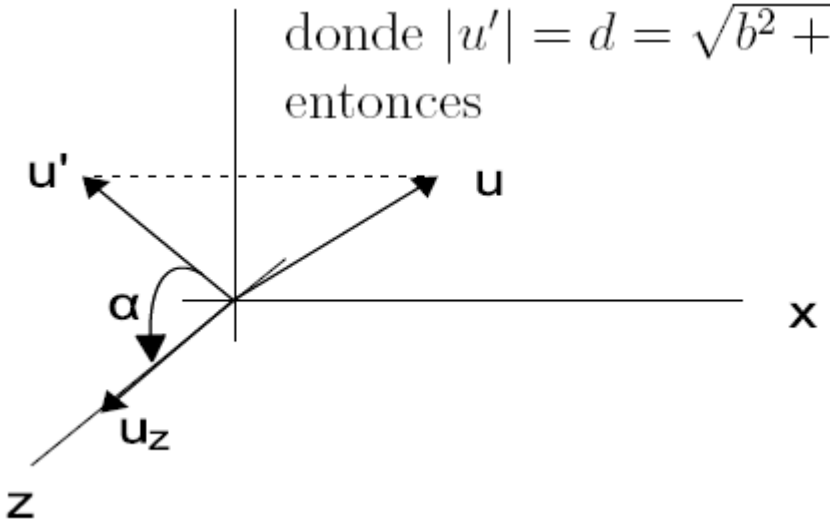
donde $|u'| = d = \sqrt{b^2 + c^2}$ y $|u_z| = 1$
entonces

$$cos(\alpha) = \frac{c}{d}$$

$$|u' \times u_z| = |u'||u_z|sen(\alpha) = |(0, b, c) \times (0, 0, 1)| = |(b, 0, 0)| = b$$

donde $|u'| = d = \sqrt{b^2 + c^2}$ y $|u_z| = 1$
entonces

$$sen(\alpha) = \frac{b}{d}$$



Rotación respecto a un eje arbitrario
Determinación de la matriz de transformación por composición de matrices

Finalmente:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sen(\alpha) & 0 \\ 0 & sen(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación respecto a un eje arbitrario
Determinación de la matriz de transformación por composición de matrices

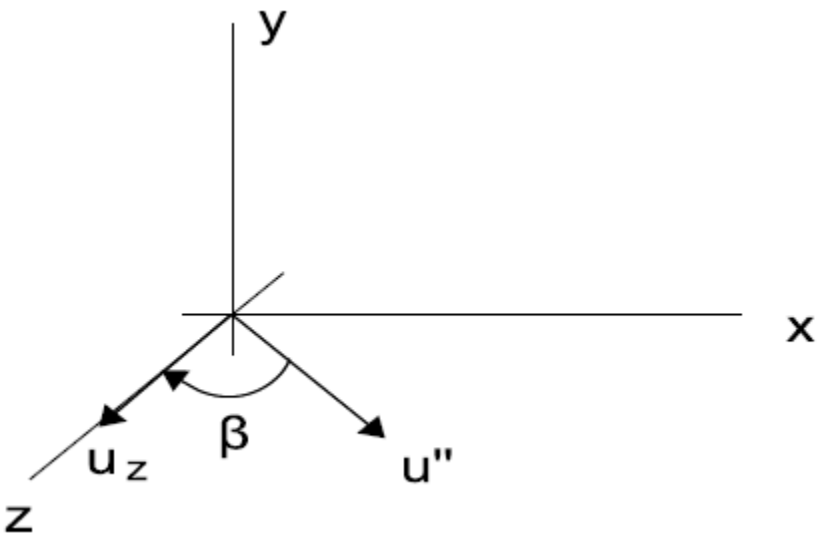
$$u'' \cdot u_z = |u'||u_z| cos(\beta) = (a, 0, d) \cdot (0, 0, 1) = d$$

donde $|u''| = \sqrt{a^2 + d^2} = \sqrt{a^2 + b^2 + c^2} = 1$

entonces $cos(\beta) = d$

$$|u'' \times u_z| = |u''||u_z| sen(\beta) = |(a, 0, d) \times (0, 0, 1)| = |(0, 0, -a)| = a$$

entonces



$$sen(\beta) = a$$

Rotación respecto a un eje arbitrario
Determinación de la matriz de transformación por composición de matrices

$$R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & -sen(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ sen(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Además:

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

Para rotar un punto p alrededor de un eje arbitrario usamos:

$$p' = T^{-1} R_x^{-1}(\alpha) R_y^{-1}(\beta) R_z(\theta) R_y(\beta) R_x(\alpha) T p$$

Rotación alrededor de un eje arbitrario búsqueda de un conjunto de vectores ortogonales

$$\begin{bmatrix} u'_{x,1} & u'_{x,2} & u'_{x,3} & 0 \\ u'_{y,1} & u'_{y,2} & u'_{y,3} & 0 \\ u'_{z,1} & u'_{z,2} & u'_{z,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde $u'_x = (u'_{x,1}, u'_{x,2}, u'_{x,3})$, $u'_y = (u'_{y,1}, u'_{y,2}, u'_{y,3})$ y $u'_z = (u'_{z,1}, u'_{z,2}, u'_{z,3})$

$$u'_z = u$$

$$u'_y = \frac{u'_z \times u_x}{|u'_z \times u_x|}$$

$$u'_x = u'_z \times u'_y$$

u es el vector alrededor del cual se desea rotar

Rotación alrededor de un eje arbitrario usando cuaterniones

Un cuaternio es un número con una parte real y tres partes imaginarias

$$(q = s + ia + jb + kc)$$

Se cumple que $$i^2 = j^2 = k^2 = -1$$

Además

$$\begin{aligned} ij &= k \\ ik &= -j \\ ji &= -k \\ jk &= i \\ ki &= j \\ kj &= -i \end{aligned}$$

Rotación alrededor de un eje arbitrario usando cuaterniones

Un cuaternio es también un par formado por un escalar y un vector de 3 componentes

$$\text{Sean los cuaterniones } q_1 = (s_1, v_1) \text{ y } q_{2=(s_2,v_2)}$$

El producto se calcula como:

$$q_1 q_2 = (s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + x_2 v_1 + v_1 \times v_2)$$

Rotación alrededor de un eje arbitrario usando cuaterniones

Para rotar un punto p alrededor de un vector u un cierto ángulo teta
Realizamos el producto de cuaternios:

$$p' = qpq^{-1}$$

$$\text{donde } p' = (0, P') \text{ y } P' \text{ es el punto rotado}$$

$$p = (0, P) \text{ y } P \text{ es el punto antes de ser rotado y } q = (cos(\theta/2), usen(\theta/2)$$

Como q es un cuaternio unitario se cumple que su inverso multiplicativo es:

$$q^{-1} = (cos(\theta/2), -usen(\theta/2)$$

Transformaciones geométricas con OpenGL

- Para traslación usamos

glTranslatef(tx,ty,tz)

- Para el escalamiento

glScalef(Sx,Sy,Sz)

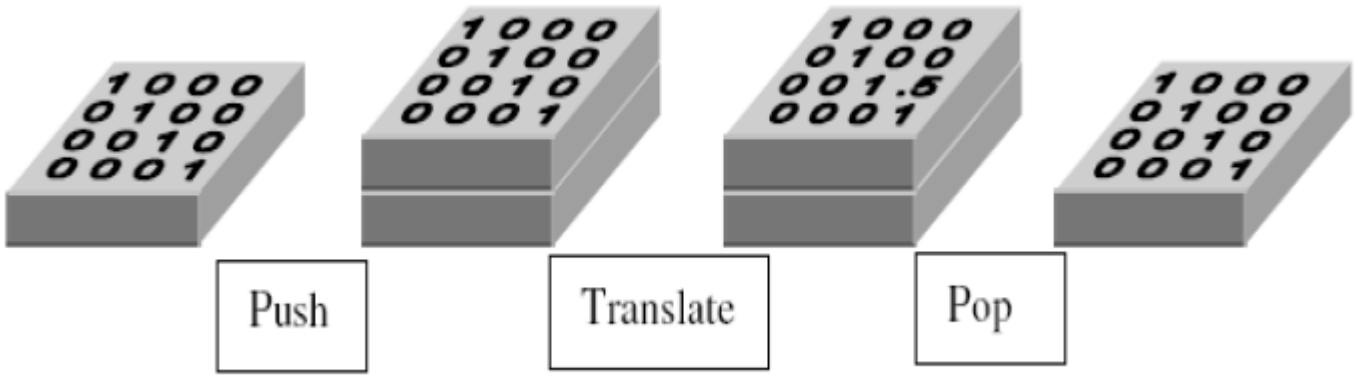
- Para la rotación

glRotatef(teta,x,y,z)

donde teta es el ángulo en grados que deseamos rotar alrededor de un vector especificado por x,y y z, por ejemplo para rotar 30 grados alrededor del eje y usariamos:

glRotatef(30,0,1,0)

Transformaciones geométricas
Manejo de Pilas de Matrices con OpenGL



La función glPushMatrix() realiza una copia de la matriz superior y la pone encima de la pila, de tal forma que las dos matrices superiores
son iguales.
glPopMatrix() desecha la matriz que está en el tope de la pila.

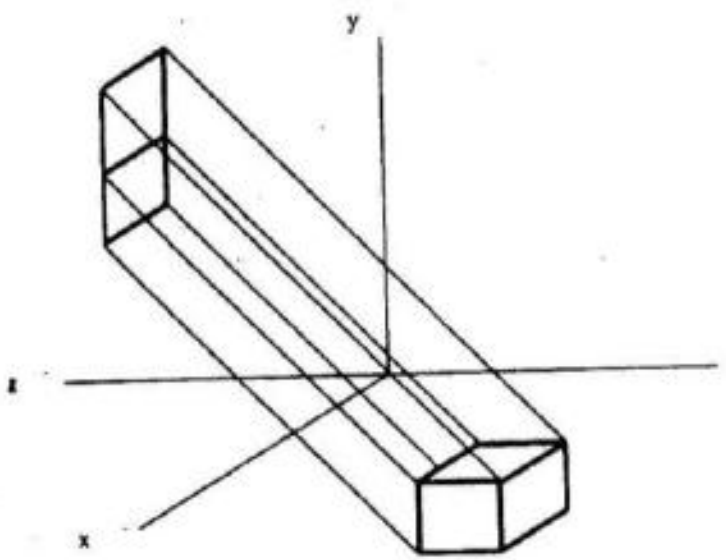
Visualización 3D

- Proyección en paralelo
- Proyección en perspectiva
- Pipeline de visualización tridimensional
- Volumen de visión
- Funciones de visualización 3D en OpenGL

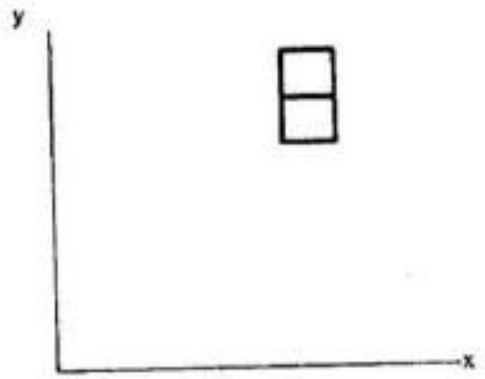
Visualización 3D
proyección en paralelo

$$x = x_1 + x_p u$$
$$y = y_1 + y_p u$$
$$z = z_1 + z_p u$$

En z=0: $u = -\frac{z_1}{z_p}$



$$x_2 = x_1 - z_1 x_p / z_p$$
$$y_2 = y_1 - z_1 y_p / z_p$$



Visualización 3D
proyección en paralelo

En forma matricial, en coordenadas homogéneas

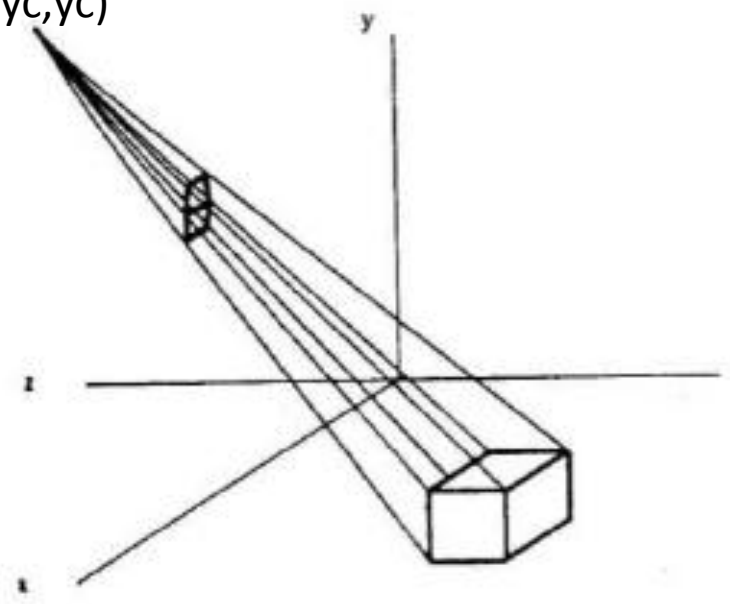
$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -x_p/z_p & 0 \\ 0 & 1 & -y_p/z_p & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

Visualización 3D
proyección en perspectiva

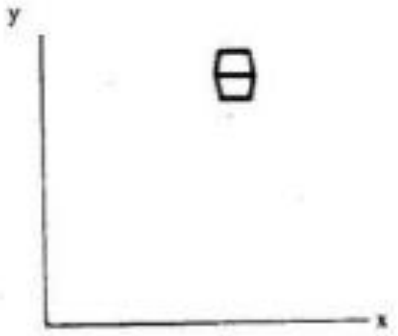
$$x = x_c + (x_1 - x_c)u$$
$$y = y_c + (y_1 - y_c)u$$
$$z = z_c + (z_1 - z_c)u$$

En z=0: $$u = -\frac{z_c}{z_1 - z_c}$$

(xc,yc,yc)



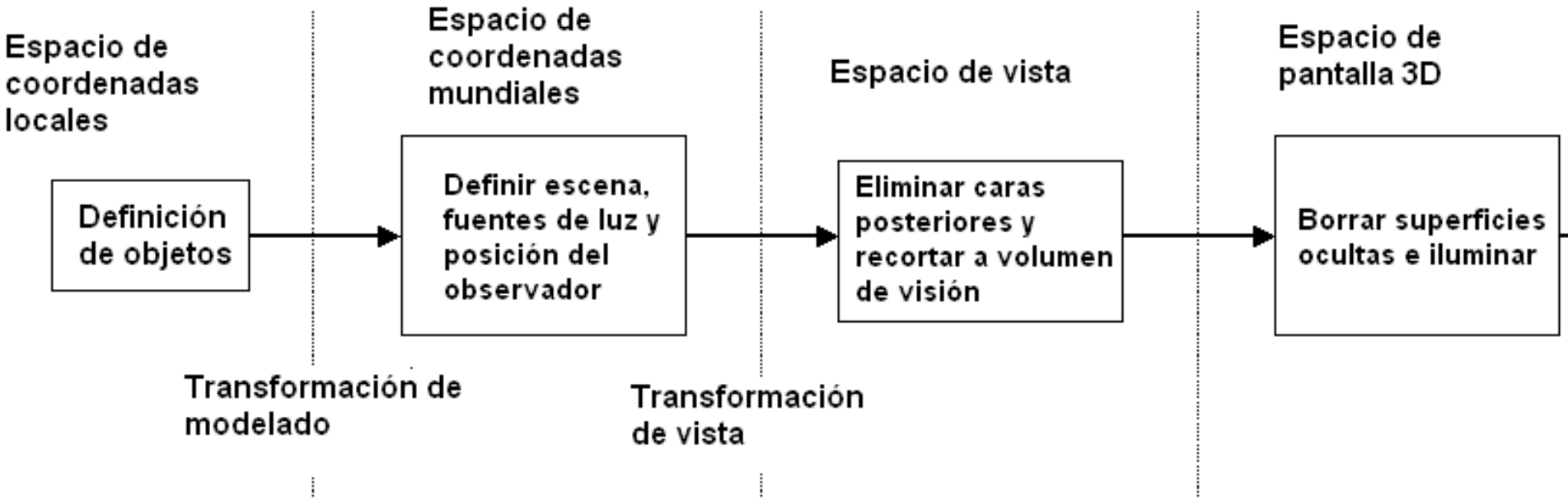
$$x_2 = x_c - z_c \frac{x_1 - x_c}{z_1 - z_c}$$
$$y_2 = y_c - z_c \frac{y_1 - y_c}{z_1 - z_c}$$



Visualización 3D
proyección en perspectiva

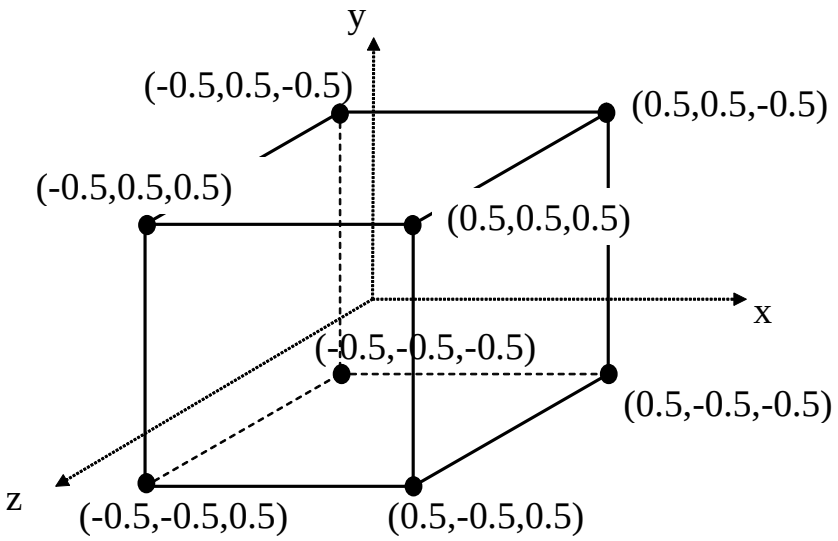
$$\begin{bmatrix} x_2 w_2 \\ y_2 w_2 \\ z_2 w_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} -z_c & 0 & x_c & 0 \\ 0 & -z_c & y_c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -z_c \end{bmatrix} \begin{bmatrix} x_1 w_1 \\ y_1 w_1 \\ z_1 w_1 \\ w_1 \end{bmatrix}$$

Visualización 3D
Pipeline de visualización tridimensional



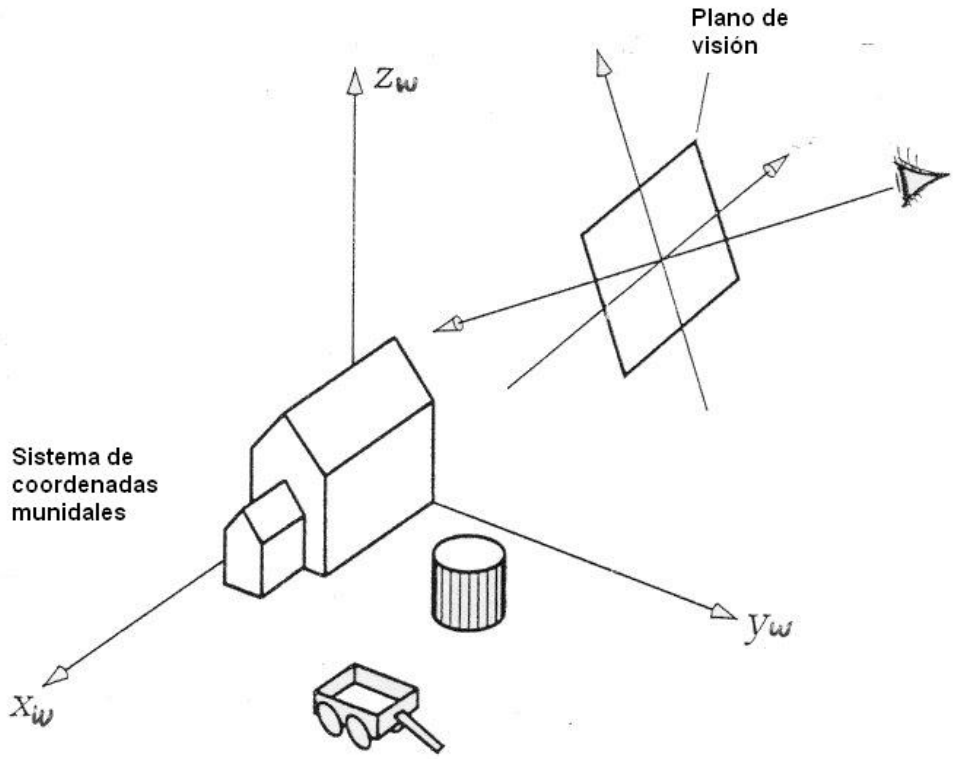
Pipeline de visualización 3D
Coordenadas locales

- También se llaman “coordenadas de modelado”
- Un objeto se define en coordenadas relativas a un punto del mismo objeto, por ejemplo el centro del objeto
- Ej podemos definir un cubo como

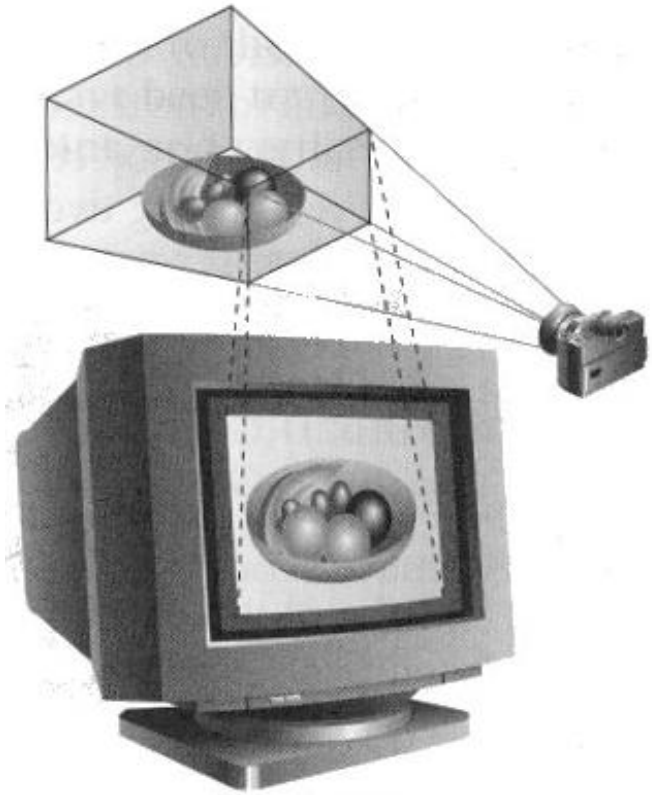


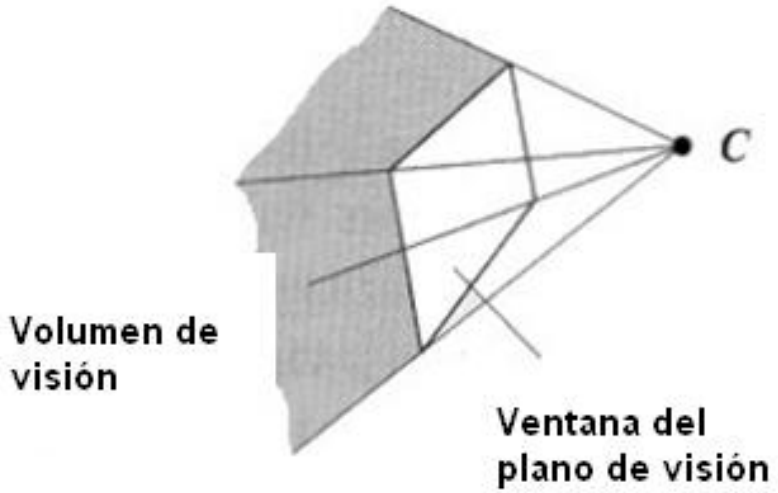
Pipeline de visualización 3D
Coordenadas mundiales

- Al definir una escena, transformamos las coordenadas de todos los objetos que la componen a un referente único, por ejemplo una de las esquinas de la habitación.

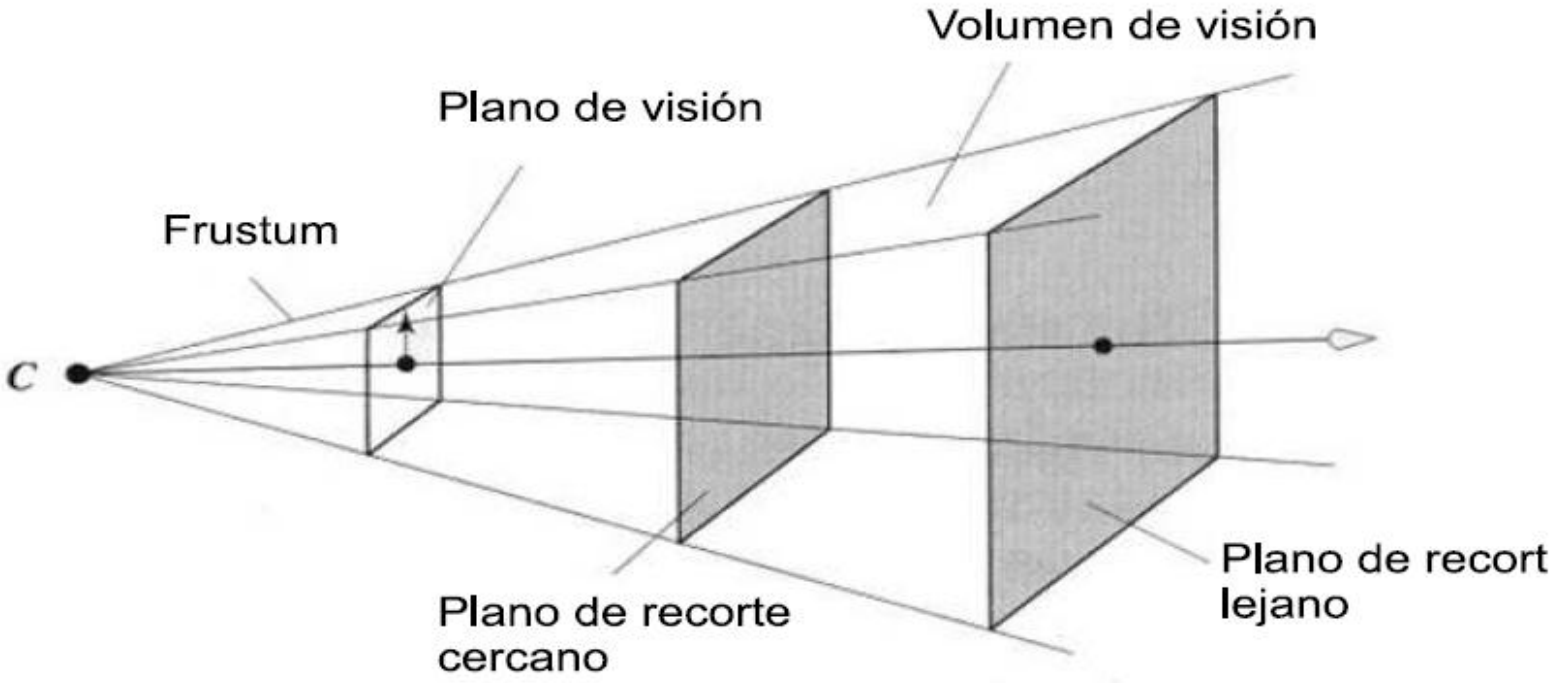


Visualización 3D
Volumen de visión





Visualización 3D
Volumen de visión



Visualización 3D
Funciones de visualización 3D en OpenGL

- Para definir el volumen de visión usamos:

glFrustum(left, right, bottom, top, near, far)

- Si el volumen de visión es simétrico podemos usar:

gluPerspective(fovy, aspect, near, far)

- donde fovy es el ángulo del campo de visión en el plano yz el cual debe estar en el rango de 0 a 180 grados, aspect es la relación ancho/alto del frustum, se recomienda hacer que coincida con la relación ancho/alto de la ventana de despliegue, near y far deben ser valores positivos, por ejemplo:

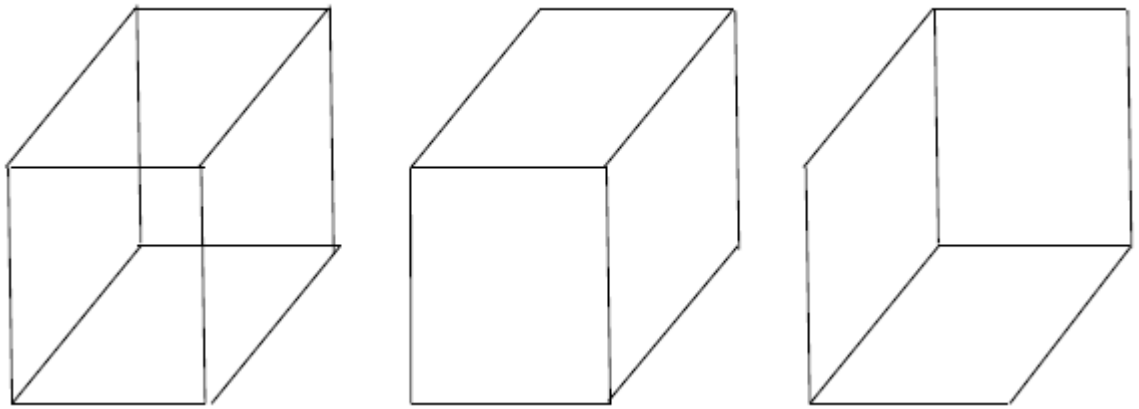
```
JGlu.gluPerspective(60.0, 1.0, 1.0, 20.0);
```

Supresión de líneas y superficies ocultas

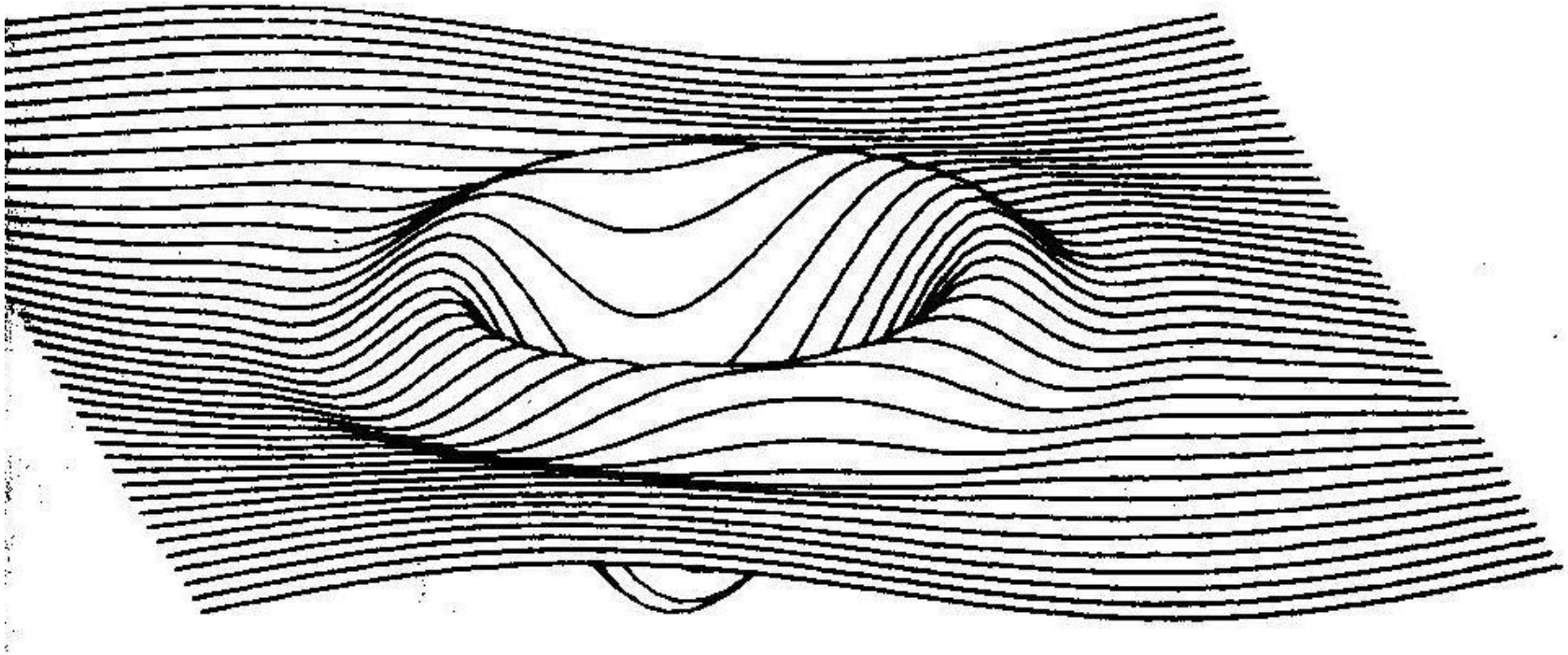
- Supresión de segmentos de línea ocultos
- Algoritmo del horizonte flotante
- Determinación de la ecuación de un plano
- Determinación del vector Normal a un plano
- Detección de caras posteriores. Algoritmo de Roberts
- Algoritmo del Buffer Z o buffer de profundidad
- Algoritmo del buffer A para superficies transparentes
- Algoritmo del buffer Z por línea de rastreo
- Método de proyección de rayos
- Método del árbol BSP
- Funciones OpenGL para suprimir superficies ocultas

Supresión de líneas y superficies ocultas

Al suprimir las líneas ocultas se elimina la ambigüedad



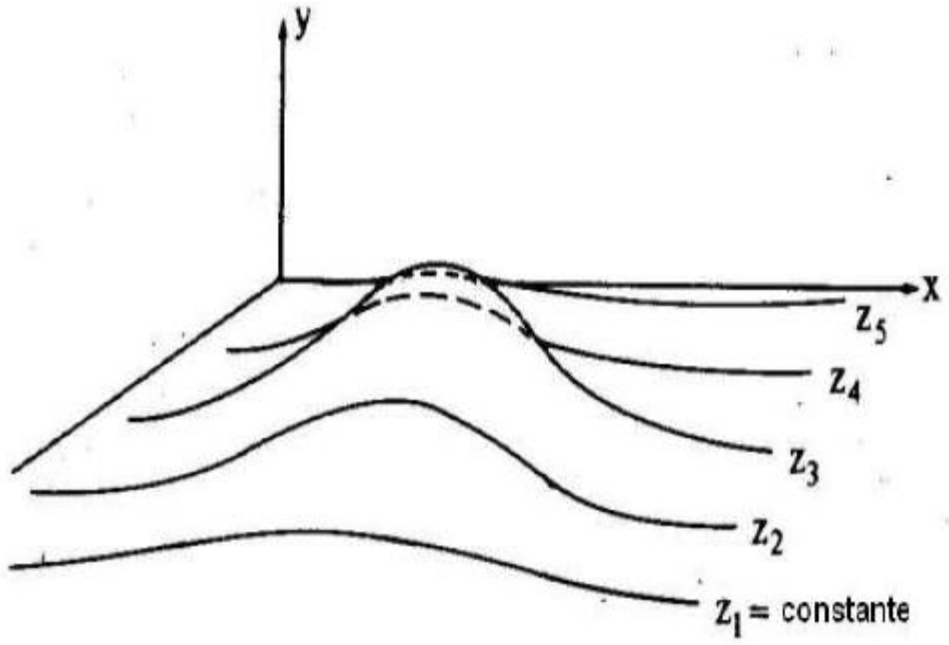
Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante



$$y = (1/5)sen x \cos z - (3/2)\cos(7a/4)\exp(-a)$$

$$a = (x-\pi)^2 + (z-\pi)^2$$

Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante

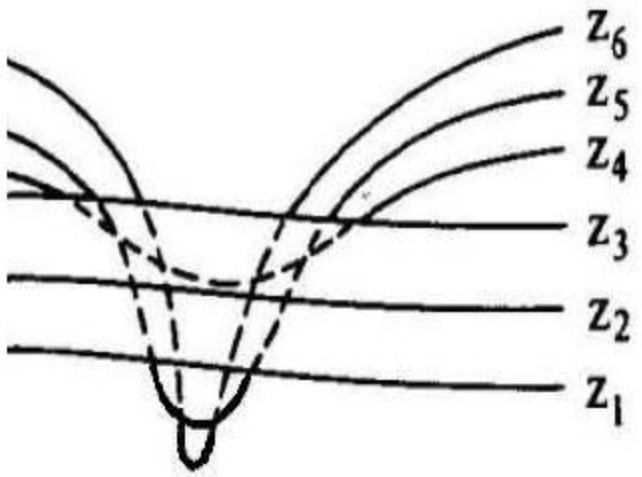


Si para cada valor dado de *x, el valor y de la curva en el plano actual es* mayor que el valor *y para todas las curvas anteriores, entonces la curva es* visible para ese valor específico de *x, en caso contrario será invisible.*

Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante

Para implementar este algoritmo simplemente hay que mantener un arreglo del mismo tamaño que la resolución de la imagen en el eje *x. Los valores* de este arreglo representan el horizonte y este horizonte “flota" a medida que cada curva es dibujada.

Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante

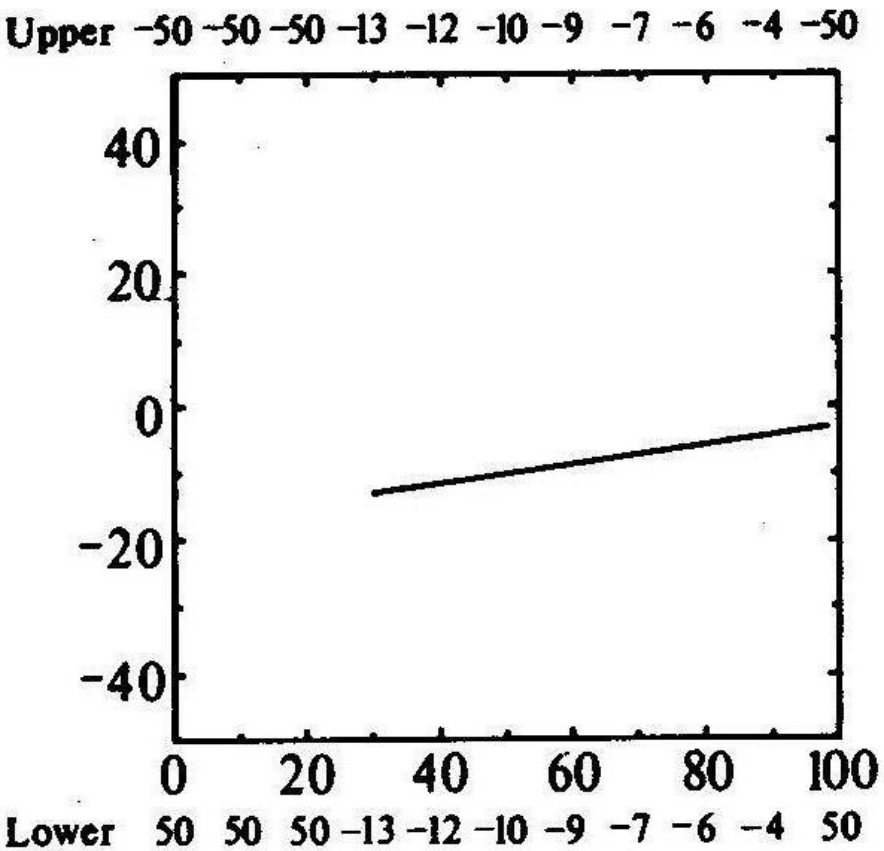


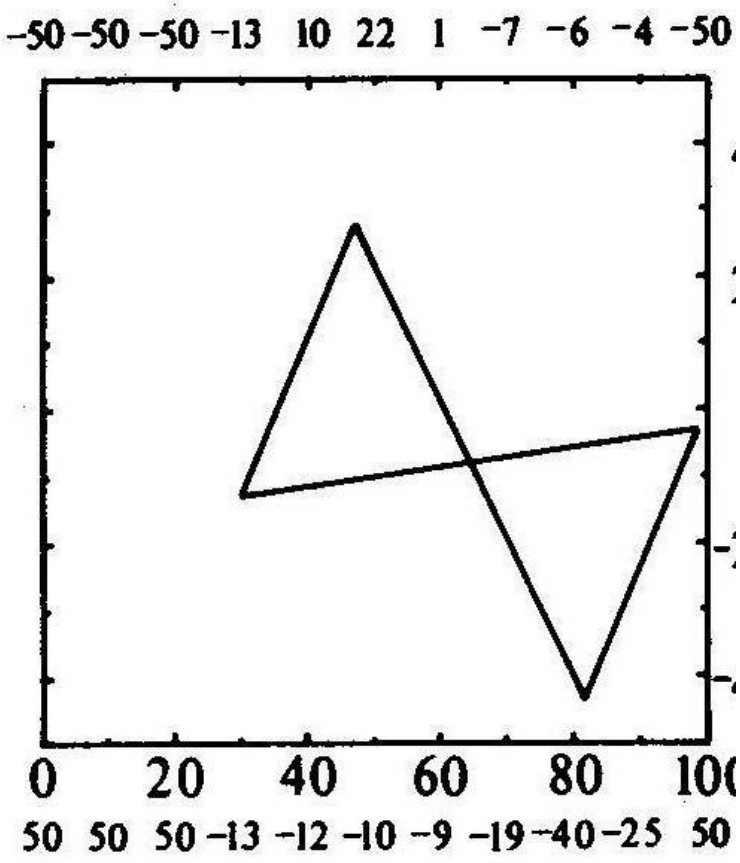
Si para algún valor dado de *x, el valor y correspondiente de la curva* en el plano actual es superior al máximo valor o inferior al mínimo valor entre todas las curvas anteriores, entonces la curva actual en dicho valor *x* es visible, si no se cumple ninguna de las dos cosas, entonces es invisible.

Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante

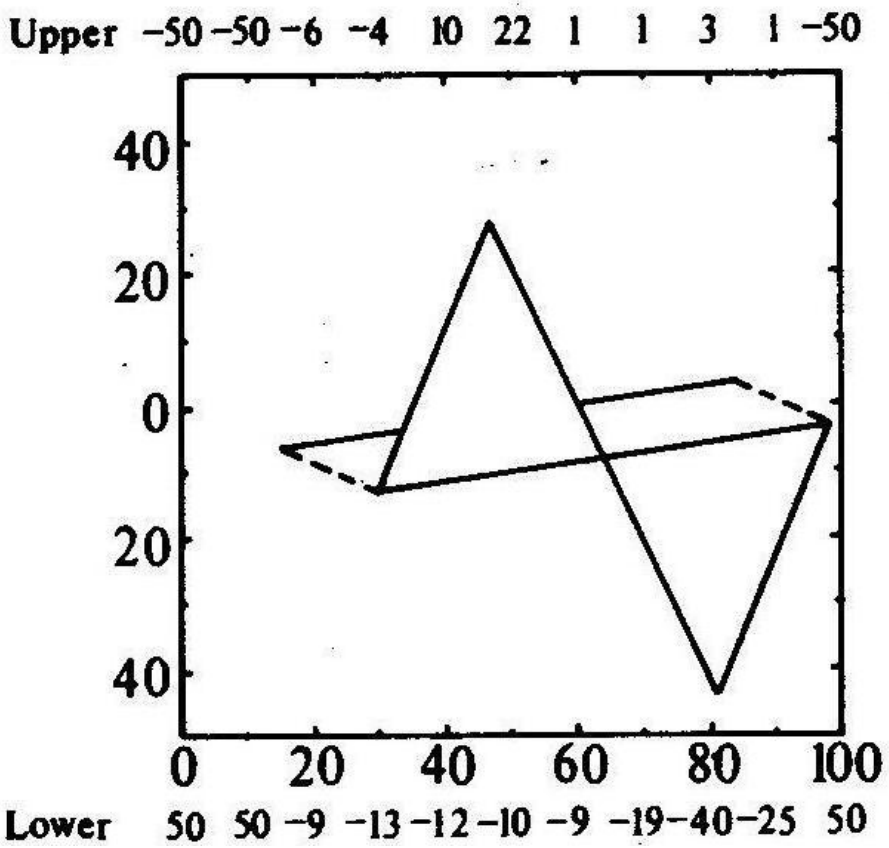
Para implementar el algoritmo del horizonte flotante se requieren entonces dos arreglos uno para almacenar los valores máximos y el otro para almacenar los mínimos, a estos arreglos se les denominan horizontes flotantes superior e inferior respectívamente.

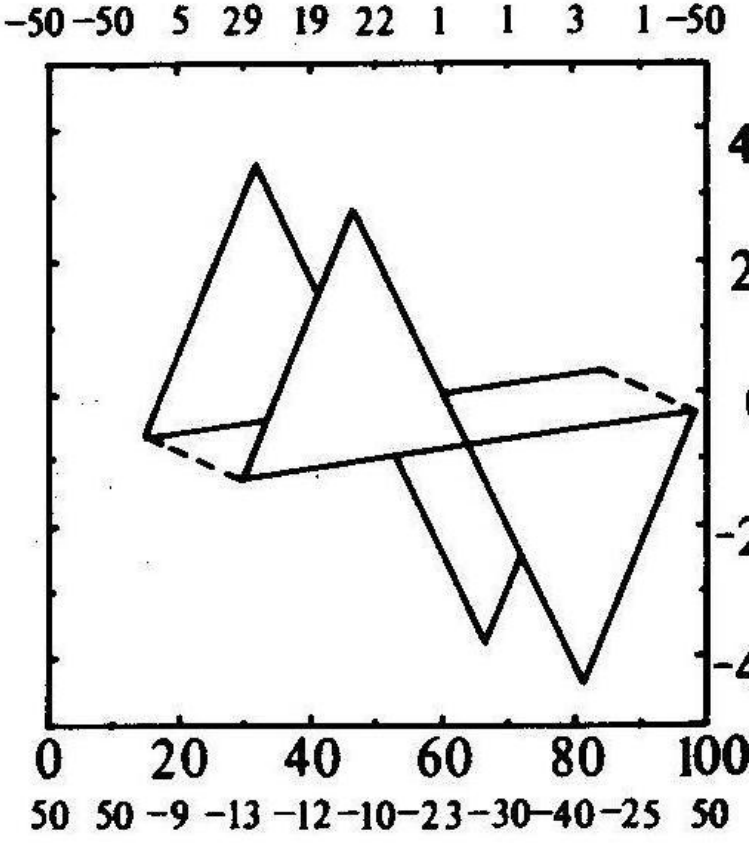
Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante



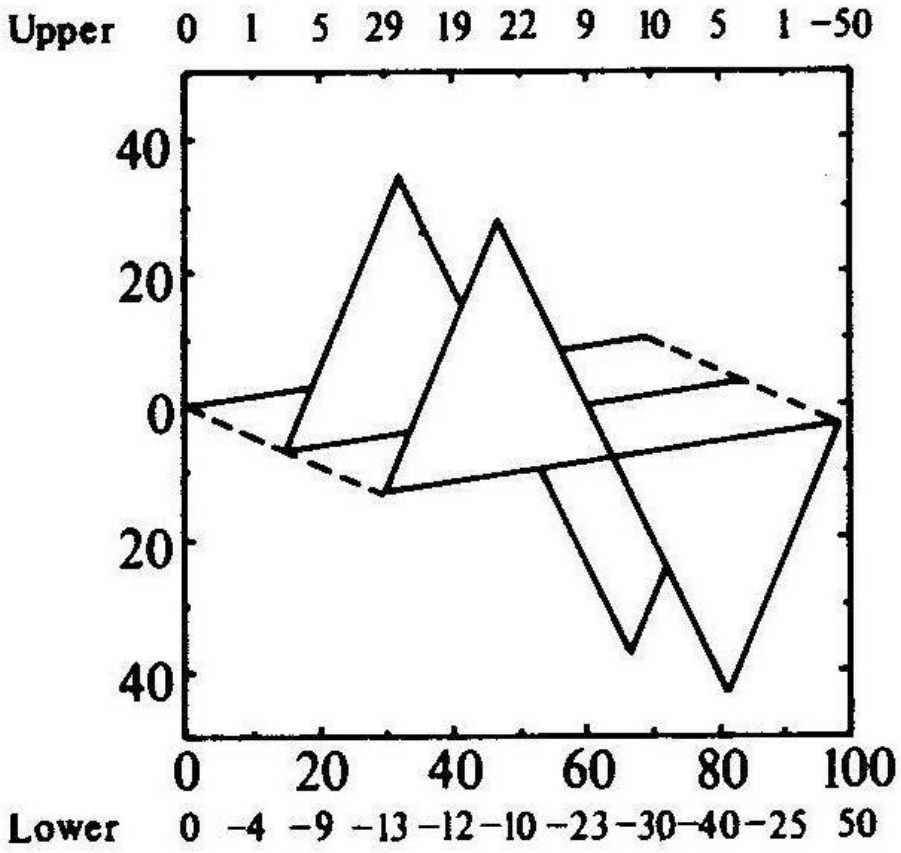


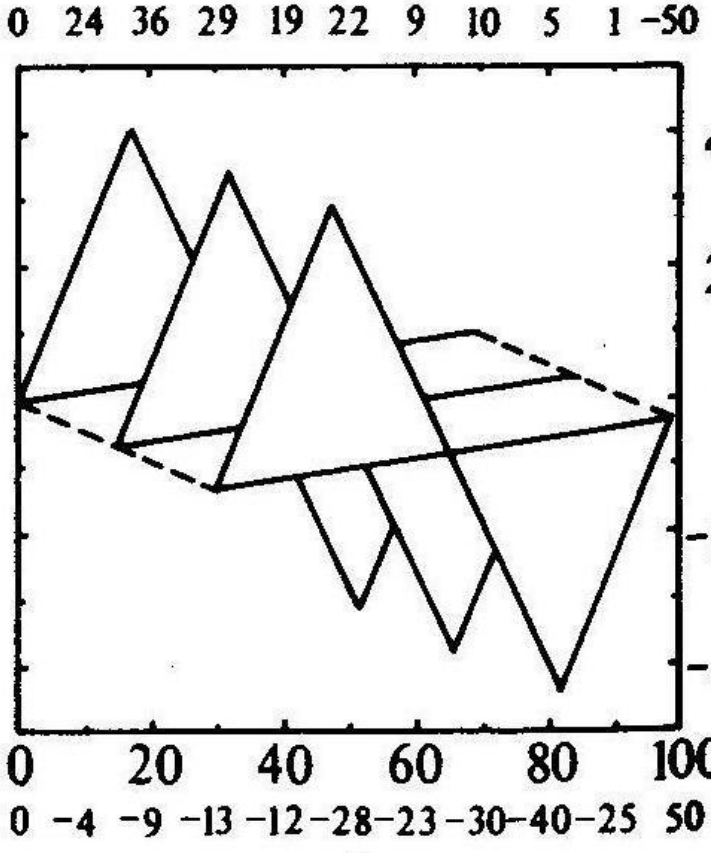
Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante





Supresión de líneas y superficies ocultas
Supresión de segmentos de línea ocultos
Algoritmo del horizonte flotante





Supresión de líneas y superficies ocultas
Determinación de la ecuación de un plano

La ecuación de un plano es: $ax + by + cz + d = 0$

O en su forma normalizada (d=1) $ax + by + cz = -1$

si tenemos 3 vértices (x_1, y_1, z_1), (x_2, y_2, z_2) y (x_3, y_3, z_3)

$$ax_1 + by_1 + cz_1 = -1$$
$$ax_2 + by_2 + cz_2 = -1$$
$$ax_3 + by_3 + cz_3 = -1$$

en forma matricial tenemos:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

Supresión de líneas y superficies ocultas
Determinación del vector normal a un plano

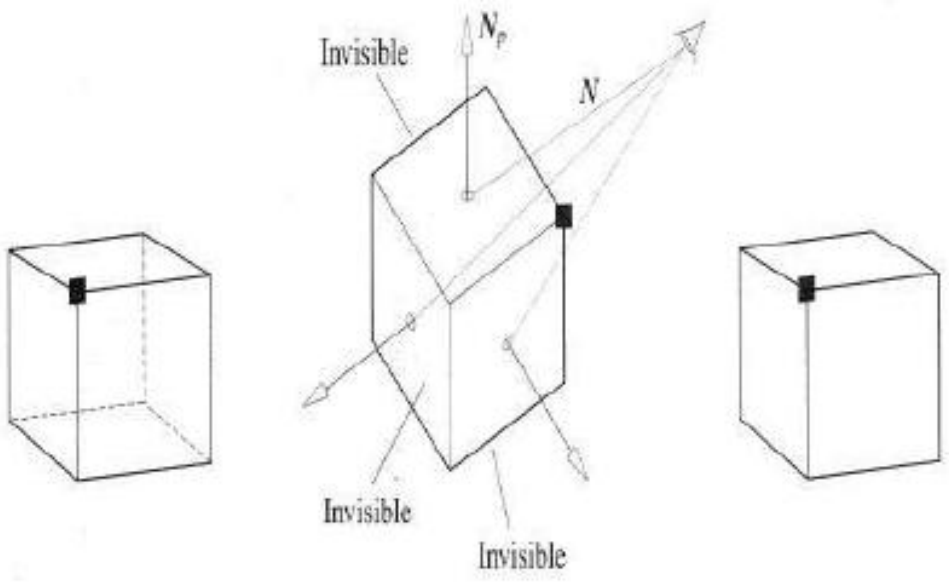
Si tenemos la ecuación del plano: $ax + by + cz + d = 0$

el vector que es normal a ese plano es:

$$N = (a, b, c)$$

también podemos determinar el vector normal al plano haciendo el producto cruz o producto vectorial de dos vectores que residan en ese plano, así es que si tenemos los vértices (x_1, y_1, z_1), (x_2, y_2, z_2) y (x_3, y_3, z_3), entonces podemos hacer el producto cruz del vector $(x_1 - x_2, y_1 - y_2, z_1 - z_2)$ y $(x_1 - x_3, y_1 - y_3, z_1 - z_3)$. El vector que resulte será un vector normal al plano al que pertenecen esos tres puntos no colineales.

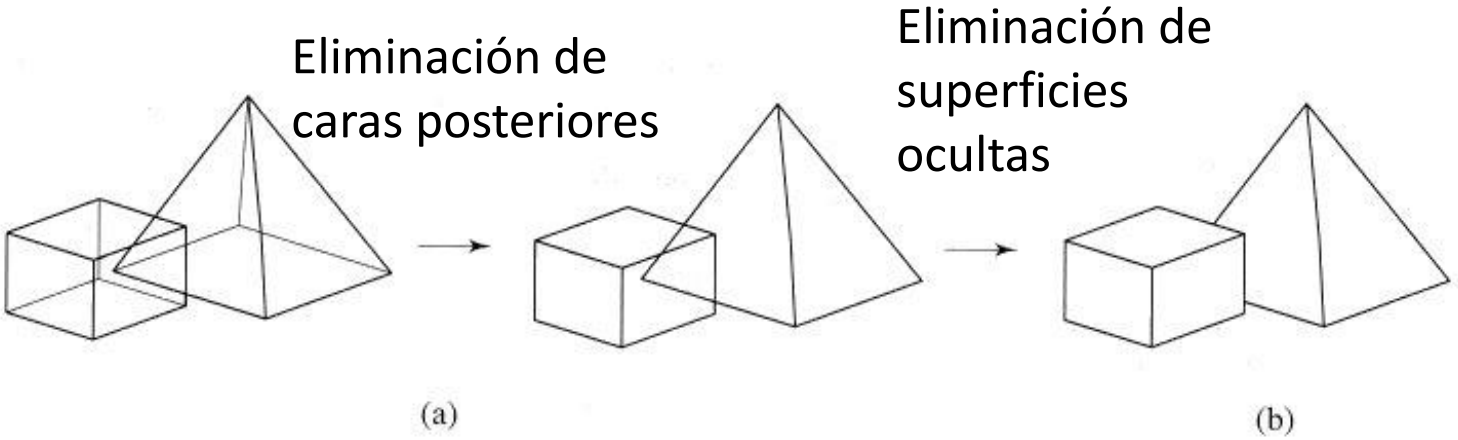
Supresión de líneas y superficies ocultas
Detección de caras posteriores
Algoritmo de Roberts



La cara es posterior si no se cumple: $N_p \cdot N < 0$

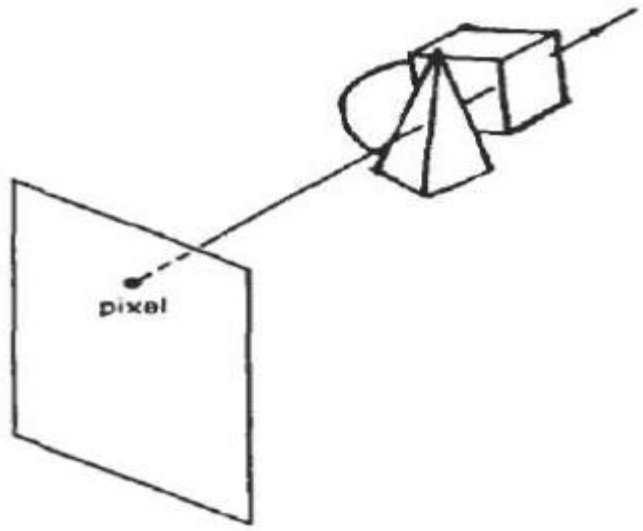
Las líneas compartidas entre dos caras que son posteriores deben suprimirse

Supresión de Líneas y superficies ocultas
Algoritmo del buffer Z o buffer de profundidad



Supresión de Líneas y superficies ocultas
Algoritmo del buffer Z o buffer de profundidad

Las partes visibles de una escena se pueden descubrir haciendo algo similar al algoritmo del horizonte flotante, es decir, podemos mantener un arreglo con las distancias desde el observador hasta la superficie mas cercana por cada pixel del plano de visión



A diferencia del algoritmo del horizonte flotante, el arreglo que necesitamos es un arreglo bidimensional del tamaño de la resolución de la imagen deseada. La "profundidad" se mide la coordenada z, por esta razón, a este arreglo le llamamos **buffer de profundidad o buffer Z.**

Supresión de Líneas y superficies ocultas
Algoritmo del buffer Z o buffer de profundidad

Por cada objeto que forma parte de una escena, por cada cara que forma parte del objeto y por cada punto (x,y,z) de la cara , calcular su profundidad z y compararla con la almacenada en el buffer Z para el correspondiente (x,y), si la profundidad del punto es menor, reemplaza a la del buffer Z.

```
IF zs < zbuffer[x,y] THEN
    write intensity to colorbuffer[x,y];
    write zs to zbuffer[x,y];
END;
```

Los valores de profundidad de una cara se pueden obtener a partir de la ecuación del plano correspondiente a esa cara mediante

$$z = \frac{-ax - by - d}{c}$$

Supresión de Líneas y superficies ocultas
Algoritmo del buffer Z o buffer de profundidad

- La principal ventaja de este método es su simplicidad, las desventajas son:
- Demasiados cálculos innecesarios puesto que en demasiadas ocasiones los pixels sobreescriben pixels anteriores
- Posibles errores de cuantización
- No soporta objetos transparentes

Supresión de Líneas y superficies ocultas
Algoritmo del buffer A para superficies transparentes

- El algoritmo del buffer A es una modificación del algoritmo del buffer Z para soportar objetos translúcidos, la única razón por la que recibe ese nombre es por que la A está en el extremo opuesto a la Z
- En el buffer A, en lugar de almacenar simplemente un valor de profundidad se almacena un apuntador a una lista ligada. Si el primer elemento de la lista es un objeto opaco, entonces la lista termina ahí
- si se trata de un objeto translúcido, entonces la lista continúa con el siguiente objeto mas cercano en la misma dirección, el cual también pudiera ser otro objeto translúcido, la lista continúa de manera que el último objeto es opaco
- Para saber el color de un pixel se debe recorrer la lista asociada a ese pixel

Supresión de Líneas y superficies ocultas
Algoritmo del buffer Z por línea de rastreo

Para acelerar el método del buffer Z, se puede proceder por línea de rastreo, las posiciones horizontales adyacentes a lo largo de la línea differen tan solo una unidad, si se ha determinado que la profundidad en la posición (x,y) es z, entonces la profundidad de la siguiente posición (x+1,y) en la misma línea de rastreo será:

$$z' = \frac{-a(x+1) - by - d}{c} = z - \frac{a}{c}$$

El valor *a/c permanece constante mientras no nos cambiemos de polígono*

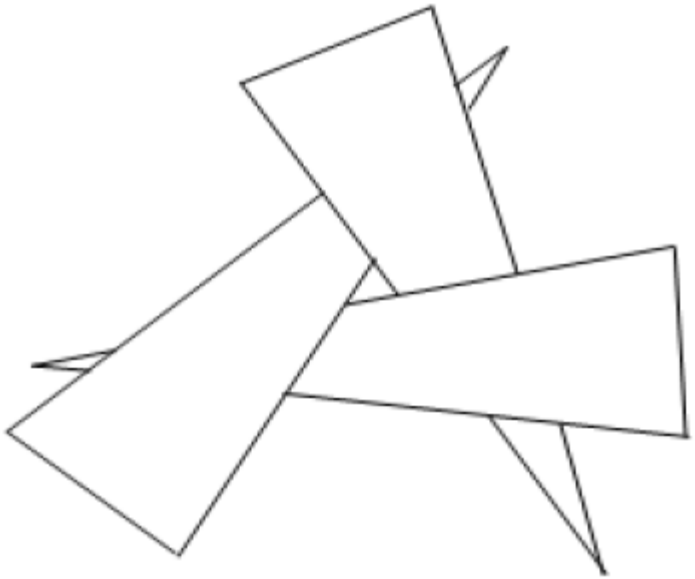
Supresión de Líneas y superficies ocultas
Método de proyección de rayos

- Consiste en trazar una línea perpendicular al plano de visión, determinar a cuales polígonos intersecta esta línea y elegir la intersección mas cercana al plano de visión, esto se hace por cada pixel del plano de visión.
- En el método del buffer de profundidad se sigue el órden de los polígonos de la escena para determinar el mas cercano para cada pixel
- En este método se sigue el orden de los pixels y se sigue un rayo por cada pixel, este método ahorra cálculos.

Supresión de Líneas y superficies ocultas
Método del árbol BSP

El algoritmo del pintor es probablemente la mas simple de las técnicas para resolver el problema de HSR, se trata de ordenar los polígonos en orden ascendente de profundidad y desplegarlos completos comenzando por los de mayor profundidad, los de menor profundidad cubrirán a los de mayor profundidad de la misma manera que en un lienzo al pintar un árbol, este cubre la parte del paisaje que queda detrás de el.

Ejemplo en el que el Algoritmo del pintor falla



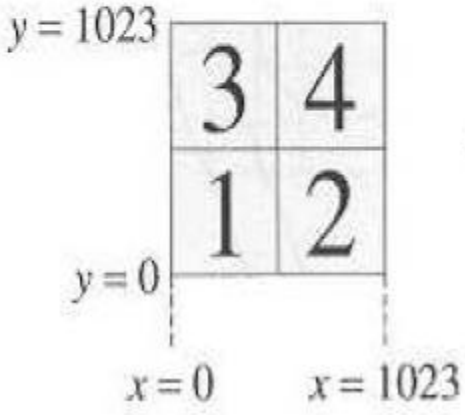
Supresión de Líneas y superficies ocultas Método del árbol BSP

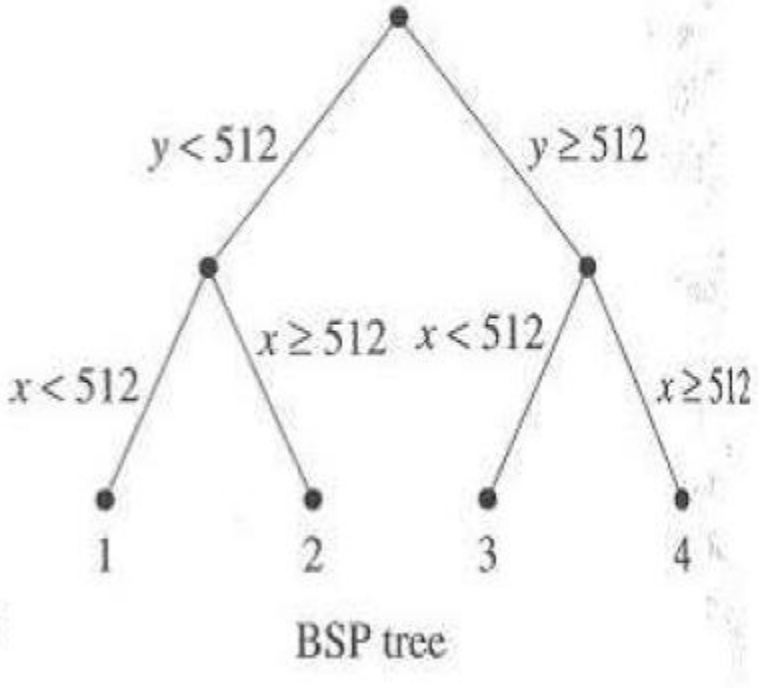
- Para solucionar el problema de los polígonos traslapados, se puede dividir el espacio sucesivamente hasta que las diferentes regiones en las que el espacio se haya dividido sean lo suficientemente pequeñas para que en su interior ningún polígono se traslape en profundidad con ningún otro polígono.
- Un árbol BSP (por Binary Space Partitioning) sirve para representar la manera en que el espacio ha sido dividido.

Supresión de Líneas y superficies ocultas
Método del árbol BSP

El espacio es particionado estableciendo planos que dividen a los polígonos en aquellos que quedan de un lado y aquellos que quedan del otro lado

Un árbol BSP tiene tantos niveles como planos divisorios, cada plano divisorio divide el espacio en dos (de ahí lo de binario).





Supresión de Líneas y superficies ocultas
Funciones OpenGL para suprimir superficies ocultas

- La eliminación de caras posteriores se lleva a efecto usando:

glEnable(GL_CULL_FACE);

glCullFace(modo);

donde modo puede tomar el valor de GL_FRONT, GL_BACK o GL_FRONT_AND_BACK.

- Utilizamos GL_BACK para eliminar caras posteriores, si usamos GL_FRONT se eliminan las caras frontales.
- Si utilizamos GL_FRONT_AND_BACK se eliminan tanto las caras frontales como las posteriores, quedando solamente los puntos y líneas que no forman polígonos.
- La eliminación de caras se desactiva mediante:

glDisable(GL_CULL_FACE);

Supresión de Líneas y superficies ocultas
Funciones OpenGL para suprimir superficies ocultas

- Para usar las rutinas de detección de visibilidad mediante buffer de profundidad debemos solicitar que configure un buffer de profundidad y un buffer de refresco (el buffer del color), esto lo hacemos de la siguiente manera:

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);

- Para inicializar el buffer de profundidad y el buffer del color hacemos:

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

Como los valores de profundidad en el sistema de coordenadas de pantalla 3D están normalizados y quedan en el rango de cero a uno, esta instrucción asigna puros valores de 1.0 al buffer de profundidad.

- La detección de la visibilidad se activa mediante:

glEnable(GL_DEPTH_TEST);

- Y se desactiva mediante:

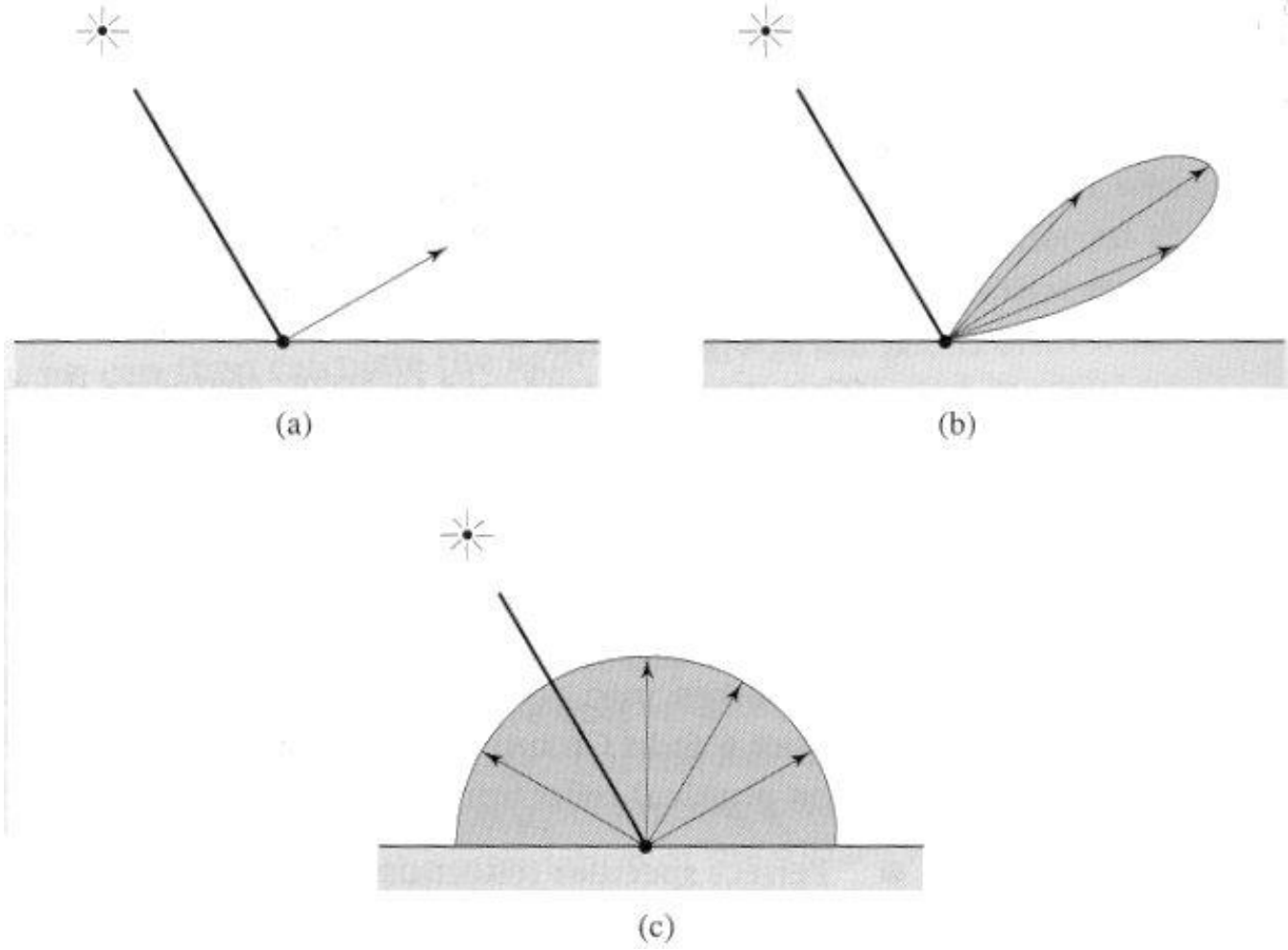
glDisable(GL_DEPTH_TEST);

Iluminación

- Reflexión difusa
- Ecuación de Lambert
- Reflexión especular
- Modelado de la iluminación
- Determinación del vector Normal a un vértice
- Sombreado de Gourad
- Sombreado de Phong
- Funciones de OpenGL para manejo de Iluminación
- Radiosidad
- Funciones OpenGL para Iluminación

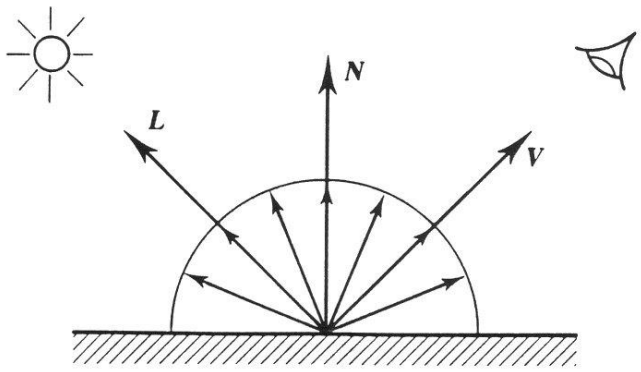
Reflexión difusa

(a)Reflexión especular perfecta
(b)Reflexión especular imperfecta
(c)Reflexión difusa



Reflexión difusa

Cuando la luz incide en un objeto opaco, la luz reflejada en su superficie lo hará en todas direcciones de manera uniforme

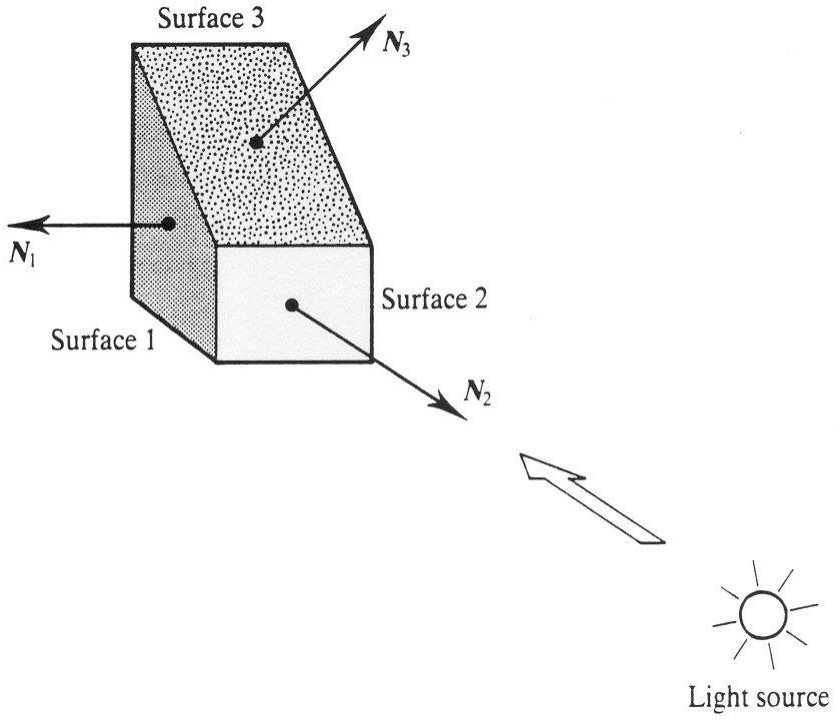


L es el vector que indica la dirección de la fuente de luz
N es el vector Normal (perpendicular) a la superficie
V es el vector que indica la posición del observador

Reflexión difusa

La brillantez de una superficie donde la luz es reflejada de forma difusa no depende de la posición del observador sino de la inclinación de dicha superficie respecto a la fuente de luz

A la reflexión difusa también se le conoce como reflexión Lambertiana



Ecuación de Lambert

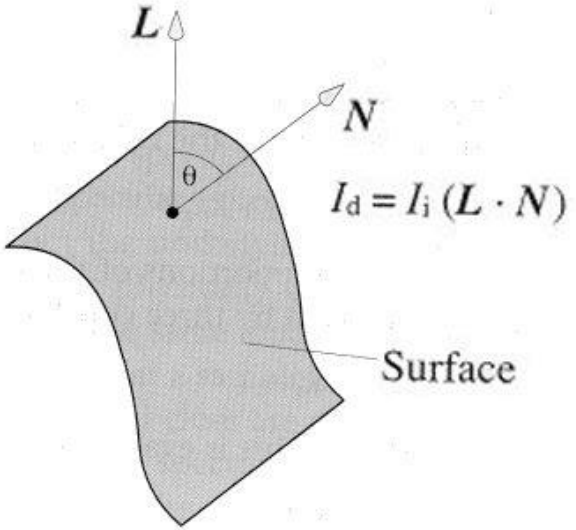
$$I_d = I_i cos\theta = I_i(L \cdot N) \quad 0 \leq \theta \leq 2\pi$$

donde I_i es la intensidad de la luz incidente, θ es el ángulo entre el vector L que apunta hacia la fuente de luz desde la superficie en cuestión y el vector

N

Para múltiples fuentes de luz se convierte en:

$$I_d = \sum_n I_{i,n} (L_n \cdot N)$$



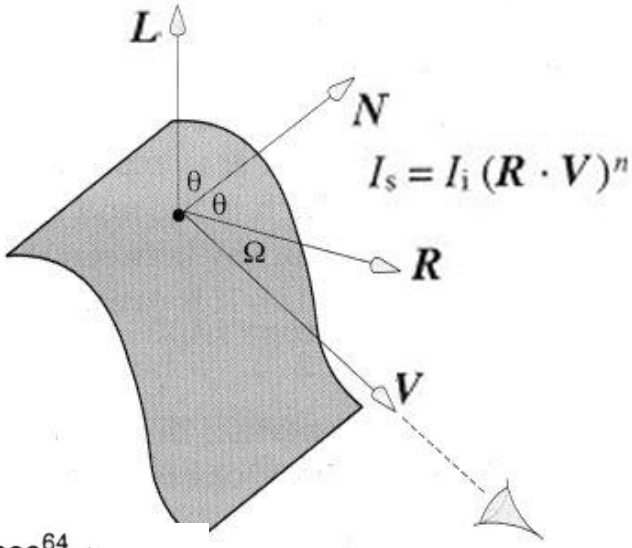
Reflexión especular

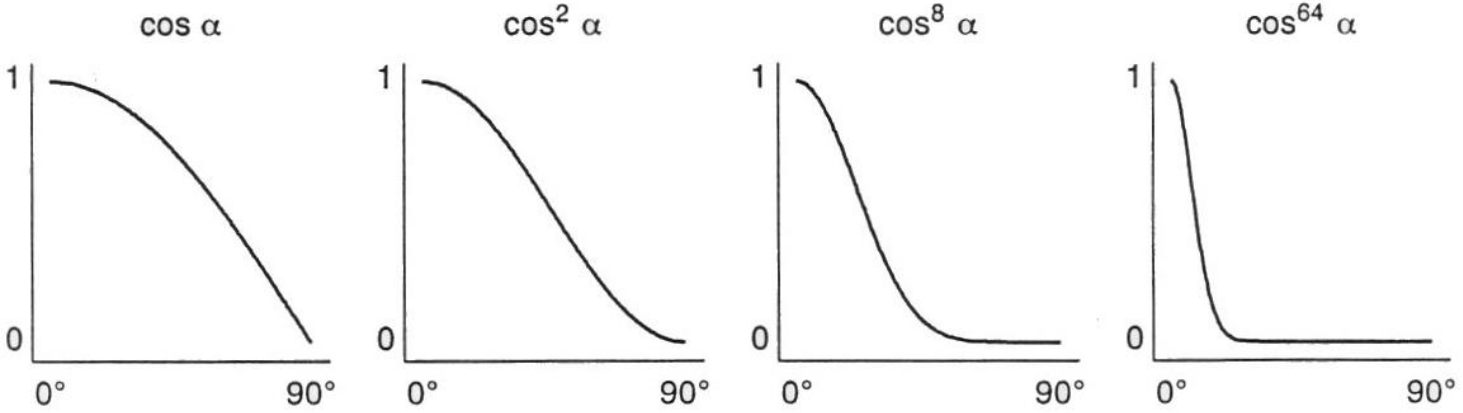
$$I_s = I_i \cos^n \Omega = I_i (\mathbf{R}.\mathbf{V})^n$$

donde I_i es la intensidad de luz incidente

Ω es el ángulo entre la dirección del observador $\mathbf{V}$ y la dirección espejo **R**.

n es un índice que indica el grado de perfección de la superficie





Modelado de la iluminación

- Modelo local.

Es aquel que solo toma en cuenta la luz que llega directamente de la fuente de luz

Ej Modelo de Iluminación de Phong

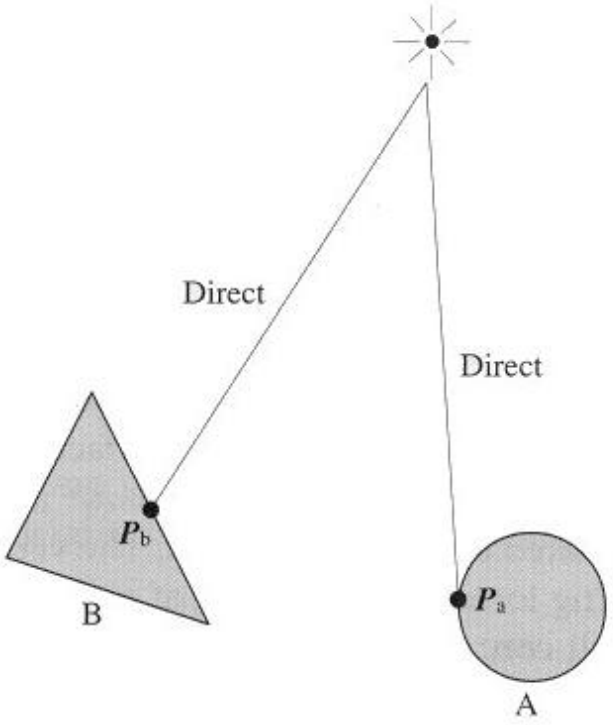
- Modelo Global.

Es el que considera no solo la luz que llega de manera directa sino aquella que llega de manera indirecta (reflejada de otros objetos)

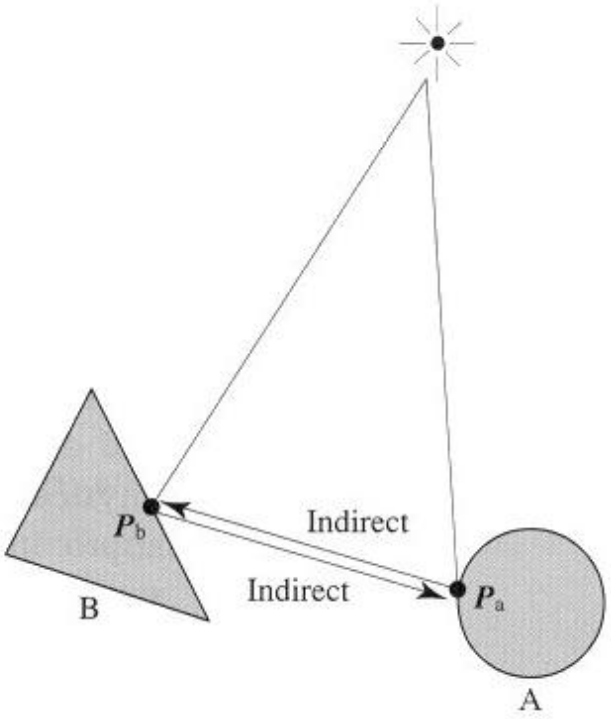
Ej Iluminación por proyección de rayos

Ej Radiosidad

Modelado de la iluminación



Iluminación directa solamente
(Modelo de iluminación local

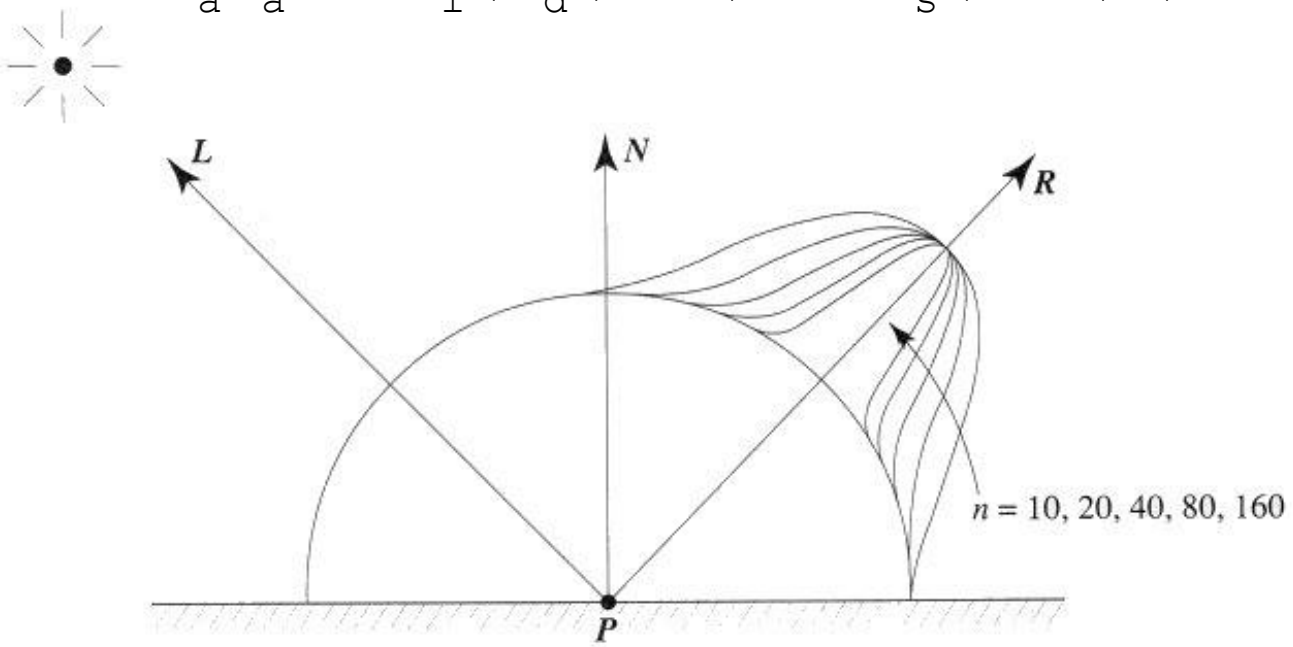


Iluminación directa e indirecta
(Modelo de iluminación global)

Modelado de la iluminación

El modelo de iluminación de Phong (1975) toma en cuenta que en una superficie ocurre en cierto grado la reflexión difusa, en cierto grado la reflexión especular y un término denominado “luz ambiental” donde se trivializa la iluminación indirecta

$$I = k_a I_a + I_i (k_d (\mathbf{L}.\mathbf{N}) + k_s (\mathbf{R}.\mathbf{V})^n)$$



Modelado de la iluminación

El modelo de iluminación de Phong (1975) toma en cuenta que en una superficie ocurre en cierto grado la reflexión difusa, en cierto grado la reflexión especular y un término denominado “luz ambiental” donde se trivializa la iluminación indirecta

$$I = k_a I_a + I_i (k_d (\mathbf{L}.\mathbf{N}) + k_s (\mathbf{R}.\mathbf{V})^n)$$

Modelado de la iluminación

Un coeficiente de reflexión es la proporción de luz incidente que es reflejada por una superficie.

K_a es el coeficiente de reflexión de la superficie para la luz ambiental; k_d es el coeficiente de reflexión difusa y k_s es el coeficiente de reflexión especular

$$I = k_a I_a + I_i (k_d (\mathbf{L}.\mathbf{N}) + k_s (\mathbf{R}.\mathbf{V})^n)$$

Al modelo de iluminación de Phong se le puede incluir un factor de atenuación de la luz que dependa de la distancia a la fuente de luz d

- $I = k_a I_a + f_{att} I_i (k_d (\mathbf{L}.\mathbf{N}) + k_s (\mathbf{R}.\mathbf{V})^n)$ donde $f_{att} = 1/(d^2)$

Modelado de la iluminación

1.El costo de la iluminación de Phong se puede reducir si se hacen las sig. consideraciones:

- La fuente de luz está muy lejos (en infinito), entonces **L** es constante
- El observador está muy lejos, entonces **V** es constante

Para transformaciones geométricas consideramos un observador muy lejano y para cálculo de iluminación consideramos una fuente de luz muy lejana.

El vector H

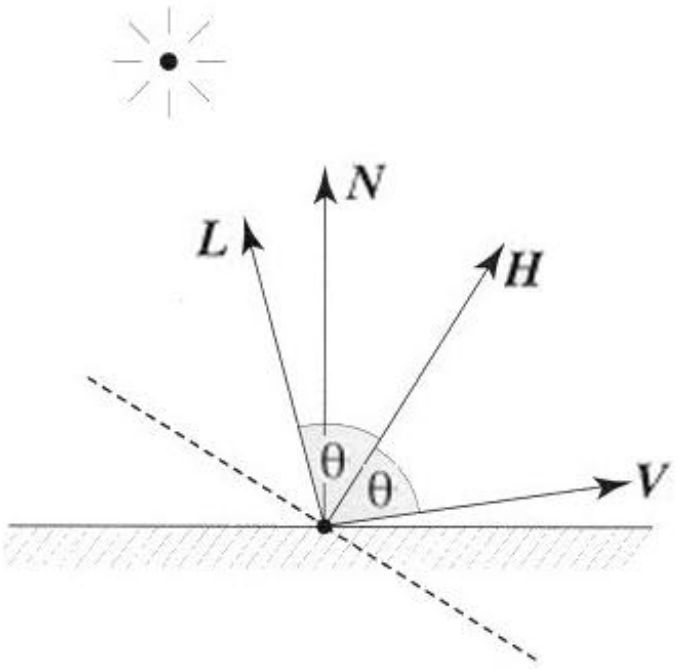
- El vector **R** es muy costoso de detrerminar, entonces definimos el vector **H**:

$$\mathbf{H} = (\mathbf{L}+\mathbf{V})/2$$

Ahora,

$$I = k_a I_a + I_i(k_d(\mathbf{L}.\mathbf{N}) + k_s(\mathbf{N}.\mathbf{H})^n)$$

- El término (**N**.**H**) varía de la misma forma que (**R**.**V**)
- Estas simplificaciones implcan que $I = f(\mathbf{N})$



Modelado de la iluminación

Iluminación de Phong para diferentes valores de Ks y Kd variando desde 0.0 a 1.0 con incrementos de 0.2 (Ka=0.7 y n=10.0 en todas las imágenes)

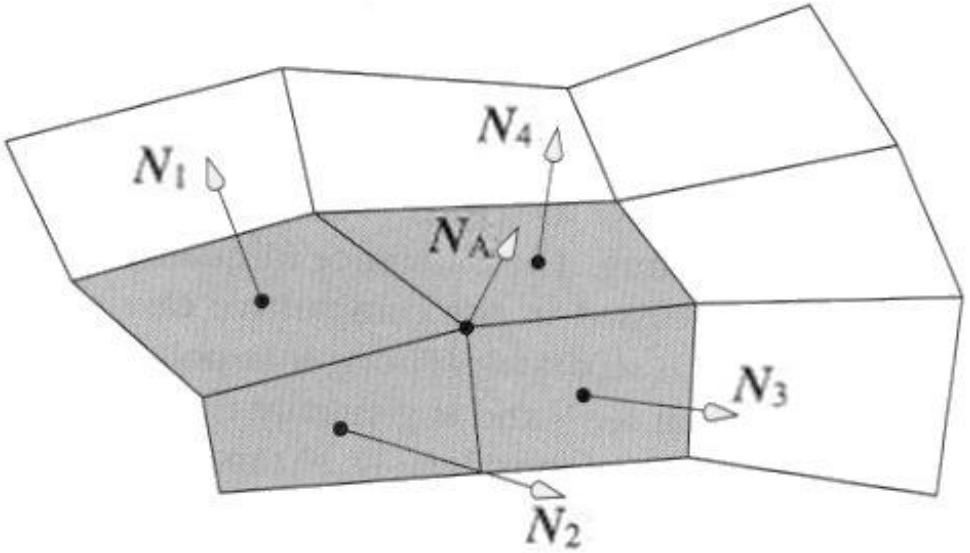
Modelado de la iluminación

Iluminación para diferentes valores de Ks y de n (Ka=0.7, Kd=1.0 para todas las imágenes)

Determinación del vector Normal a un vértice

La normal a un vértice se define como el promedio de las normales a los polígonos de los que forma parte el vértice en cuestión

$$N_v = \frac{\sum_{k=i}^{n} N_k}{|\sum_{k=i}^{n} N_k|}$$



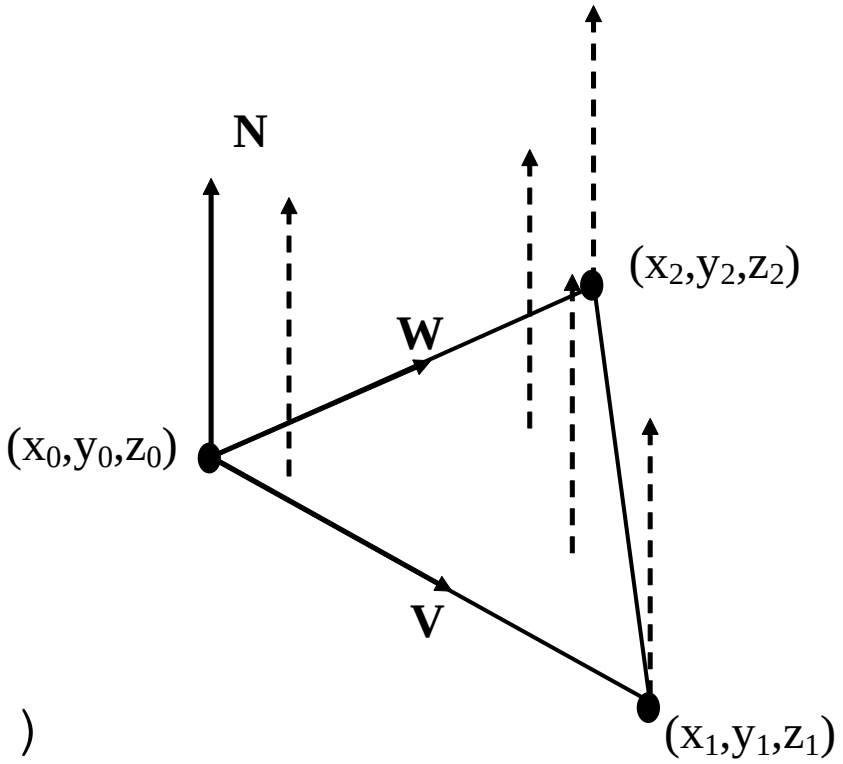
Ejemplo:

$$N_A = (N_1 + N_2 + N_3 + N_4)/4$$

Determinación del vector Normal a un vértice

Por supuesto, primero debemos determinar la normal en cada plano

$$\mathbf{V} = (x_1, y_1, z_1) - (x_0, y_0, z_0)$$
$$\mathbf{W} = (x_2, y_2, z_2) - (x_0, y_0, z_0)$$
$$\mathbf{N} = \mathbf{V} \times \mathbf{W}$$
$$= (v_x, v_y, v_z) \times (w_x, w_y, w_z)$$
$$= ((v_y w_z - v_z w_y),$$
$$(v_z w_x - v_x w_z),$$
$$(v_x w_y - v_y w_x))$$



- Cada **N** debe ser normalizada:

$$\mathbf{N} = (N_x/|\mathbf{N}|,\ N_y/|\mathbf{N}|,\ N_z/|\mathbf{N}|)$$

donde

$$|\mathbf{N}| = \mathrm{sqrt}(N_x^2 + N_y^2 + N_z^2)$$

Sombreado de Gourad

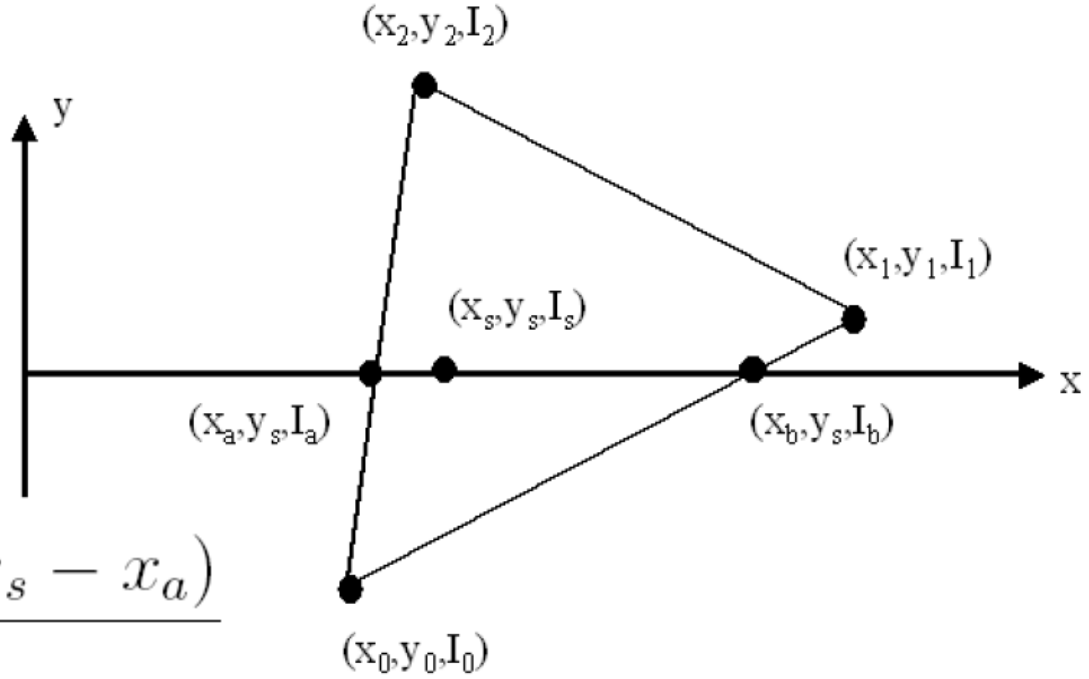
- El proceso de sombreado de Gourad de cada polígono de que está hecho el objeto consiste en:

1. Determinar el vector normal en cada uno de los vértices del polígono
2. Aplicar el modelo de iluminación de Phong o alguna variante para determinar la intensidad en cada vértice
3. Interpolar linealmente las intensidades de los vértices para determinar

Sombreado de Gourad

$$I_a = \frac{I_2(y_s - y_0) + I_0(y_2 - y_s)}{y_2 - y_0}$$

$$I_b = \frac{I_1(y_s - y_0) + I_0(y_1 - y_s)}{y_1 - y_0}$$



$$I_s = \frac{I_a(x_b - x_s) + I_b(x_s - x_a)}{x_b - x_a}$$

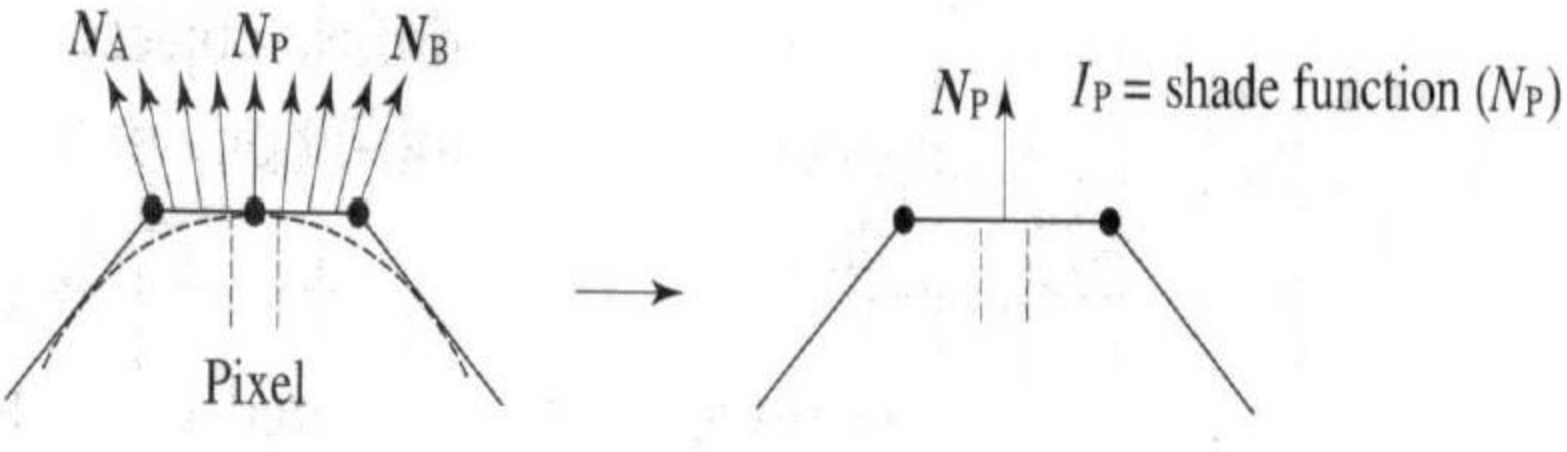
Sombreado de Gourad

- El sombreado de Gourad evita que se vean las divisiones entre los polígonos que forman un objeto
- Como la reflexión especular solo es calculada en los vértices, la interpolación de Gourad no maneja de manera adecuada la reflexión especular

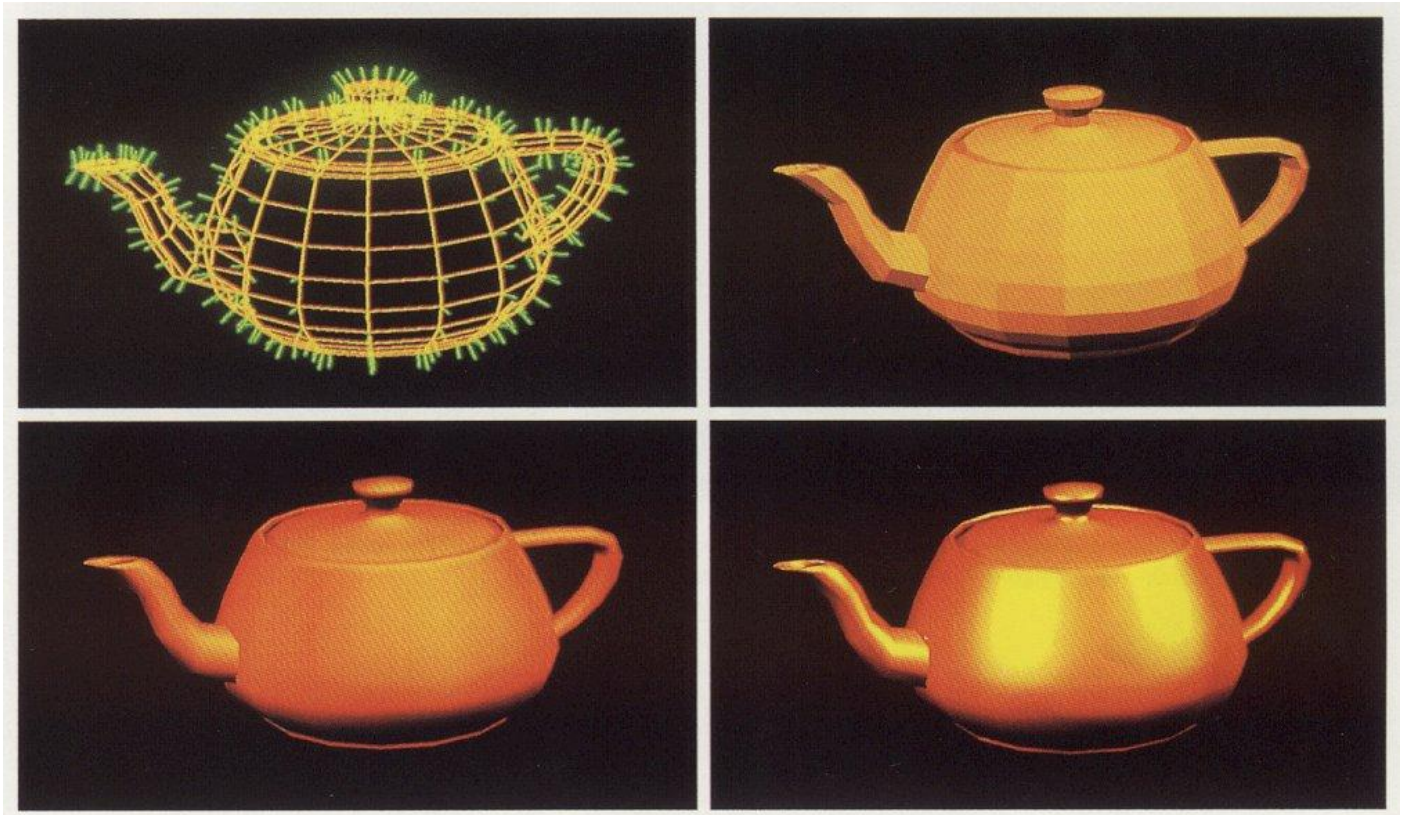
Sombreado de Phong

- Las normales a los vértices se interpolan a través de los polígonos
- La intensidad de luz se calcula en cada pixel usando la normal interpolada
- La luz especular en cada pixel se determina de manera adecuada
- El costo de la interpolación de Phong es mucho mayor que el de Gourad porque en el de Gourad la intensidad de luz solo se calcula en los vértices

Sombreado de Phong



Sombreado de Phong



2 The Utah teapot and shading options. Note the visibility of piecewise linearities along silhouette edges since this representation contains only 512 polygons. (*top left*) Wireframe plus vertex normals. (*top right*) Constant shaded polygons. (*bottom left*) Gouraud shading. (*bottom right*) Phong shading.

Funciones de OpenGL para manejo de Iluminación

- Para definir una fuente de luz, debemos establecer sus parámetros.
- La siguientes dos líneas establecen una fuente de luz en la posición (-2,2,2), el cuarto elemento del arreglo light_pos indica si los tres elementos anteriores del arreglo se deben interpretar como una posición (uno) o como una dirección (cero) lo cual implica que la luz se considera local o muy lejana respectivamente.
- GLFloat light_pos[]=(-2.0,2.0,2.0,1.0);
- glLightfv(GL_LIGHT0,GL_POSITION,light_pos);

Funciones de OpenGL para manejo de Iluminación

- Las siguientes dos líneas sirven para especificar la luz ambiental como blanca (primeros tres valores del arreglo light_Ka)
- GLFloat light_Ka[]=(1.0,1.0,1.0,1.0);
- glLightfv(GL_LIGHT0,GL_AMBIENT,light_Ka);
- Las siguientes dos líneas sirven para especificar la luz difusa como blanca (primeros tres valores del arreglo light_Kd)
- GLFloat light_Kd[]=(1.0,1.0,1.0,1.0);
- glLightfv(GL_LIGHT0,GL_DIFFUSE,light_Kd);

Funciones de OpenGL para manejo de Iluminación

- Las siguientes dos líneas sirven para especificar la luz especular como blanca (primeros tres valores del arreglo light_Ks)
- GLFloat light_Ks[]=(1.0,1.0,1.0,1.0);
- glLightfv(GL_LIGHT0,GL_SPECULAR,light_Ks);

Funciones de OpenGL para manejo de Iluminación

- Los coeficientes de reflexión dependen del material del que están hechos los objetos.
- Las siguientes dos líneas sirven para especificar una superficie que al incidir luz ambiental blanca la reflejará roja
- GLFloat material_Ka[]=(1.0,0.0,0.0,1.0);
- glMaterialfv(GL_FRONT,GL_AMBIENT,material_Ka);
- Las siguientes dos líneas sirven para especificar los coeficientes de reflexión difusa para cada uno de los tres colores básicos.
- GLFloat material_Ka[]=(1.0,1.0,1.0,1.0);
- glMaterialfv(GL_FRONT,GL_DIFFUSE,material_Kd);

Funciones de OpenGL para manejo de Iluminación

- Las siguientes dos líneas sirven para especificar los coe¯cientes de reflexión especular para cada uno de los tres colores básicos.
- GLFloat material_Ka[]=(1.0,1.0,1.0,1.0);
- glMaterialfv(GL_FRONT,GL_SPECULAR,material_Kd);
- Las siguientes dos líneas sirven para especificar que un objeto tiene luz propia y el color de esta.
- GLFloat material_Ke[]=(1.0,1.0,1.0,1.0);
- glMaterialfv(GL_FRONT,GL_EMISSION,material_Ke);

Funciones de OpenGL para manejo de Iluminación

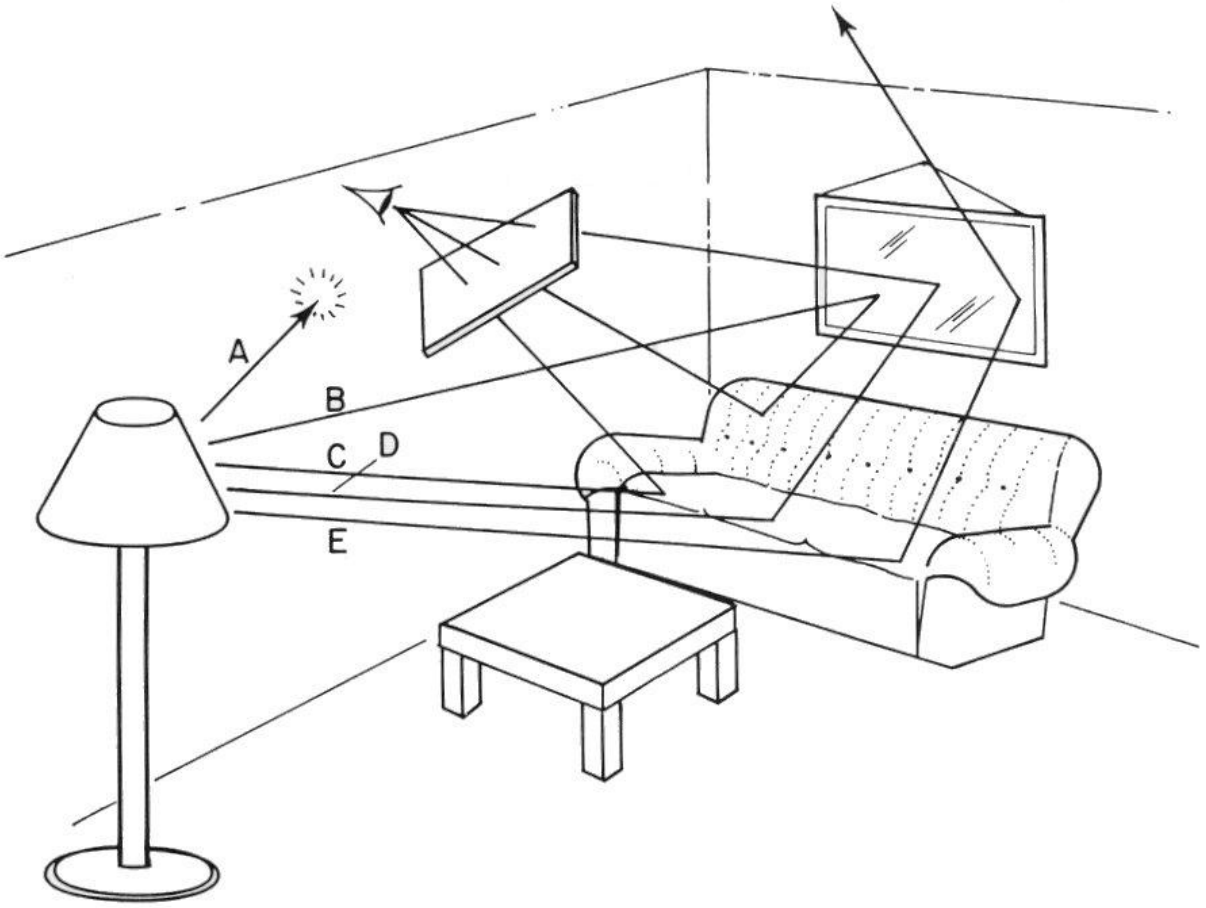
- La siguiente línea sirve para especificar que tan perfecta es la reflexión especular, es decir, el exponente *n del modelo de iluminación de Phong*
- Por ejemplo, para asignar el valor de 10.
- glMaterialfv(GL_FRONT,GL_SHININESS,10);

Iluminación por proyección de rayos

Es un método de iluminación global
(toma en cuenta la iluminación indirecta)

Este método modela
la reflexión especular
pero no la difusa

El método es
dependiente de la
posición del observador



Iluminación por proyección de rayos

- La proyección recursiva de rayos es un método que resuelve simultáneamente los siguientes problemas:
- Detección de superficies visibles
- Sombreado por iluminación directa
- Agregar efectos de reflexión especular global
- Determinación de la geometría de las sombras

Iluminación por proyección de rayos

$$I(P) = I_{local}(P) + K_{rg}I(P_r) + K_{tg}I(P_t)$$

El primer término es la contribución local proviene directamente de la fuente de luz y se calcula con el Modelo de iluminación de Phong

El segundo es la cantidad de luz reflejada que proviene de la dirección de reflexión de otro punto Pr (iluminación indirecta)

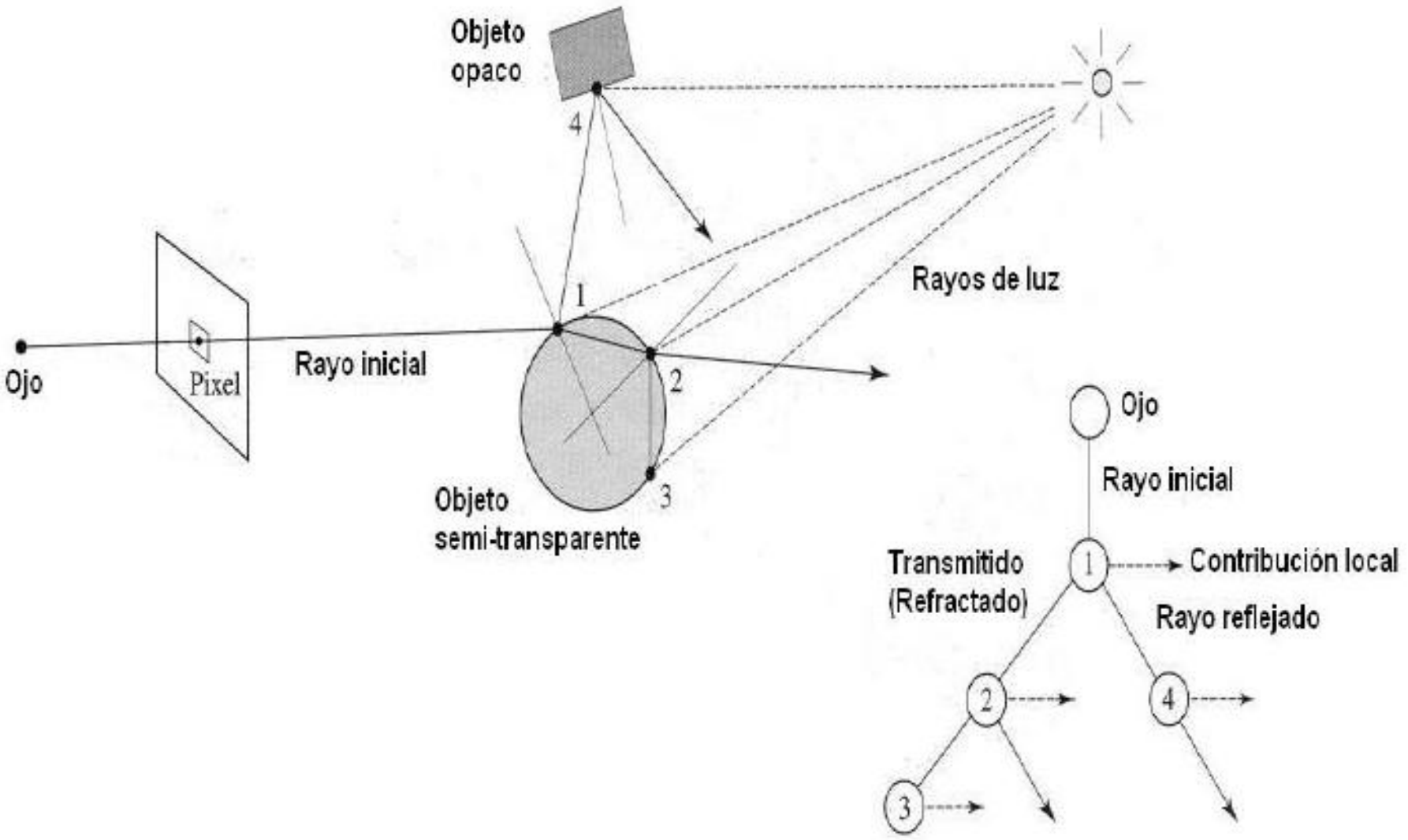
El tercero es la cantidad de luz reflejada que proviene del punto Pt en la dirección de refracción (iluminación indirecta)

Este es un modelo recursivo pues I(Pr) e I(Pt) también se determinan usando la misma fórmula

Iluminación por proyección de rayos

- Para implementar este método conviene construir un árbol donde se van guardando las diferentes contribuciones de luz
- Las hojas de este árbol son las primeras en tener completa su información
- La raíz es la última en completarla, entonces se puede decir cual es la intensidad de luz que corresponde a un pixel.
- La altura del árbol es un parámetro que se interpreta como el número de intersecciones que se desea encontrar cuando se le da seguimiento al rayo,
- Un árbol de mayor altura corresponde con una mejor renderización pero evidentemente mas costosa

Iluminación por proyección de rayos



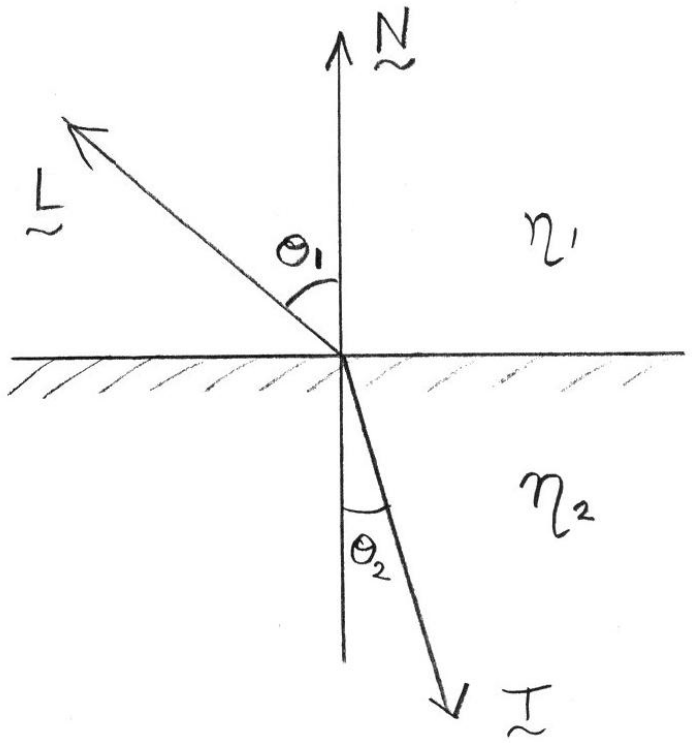
Iluminación por proyección de rayos

Para encontrar la dirección de refracción *T usamos la ley de Snell*

$$\mathbf{T} = -\frac{\eta_1}{\eta_2}\mathbf{L} - \left(\cos\theta_2 - \frac{\eta_1}{\eta_2}\cos\theta_1\right)\mathbf{N}$$

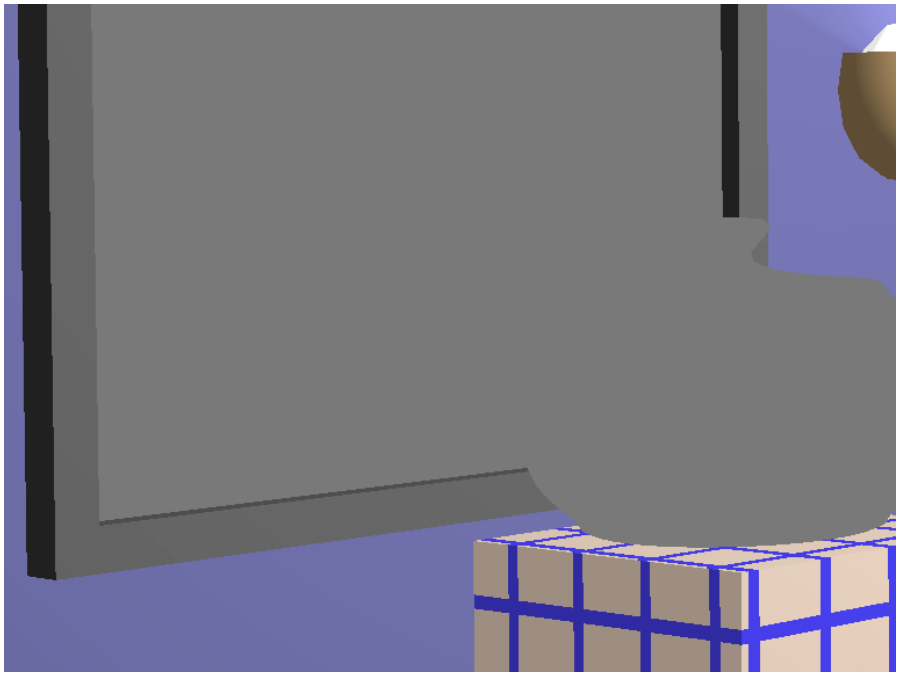
Indices de refracción de algunos materiales

Azúcar	1.56
Diamante	2.417
Mica	1.56-1.6
Benceno	1.504
Glicerina	1.47
Agua	1.333
Alcohol etílico	1.362
Aceite de oliva	1.46

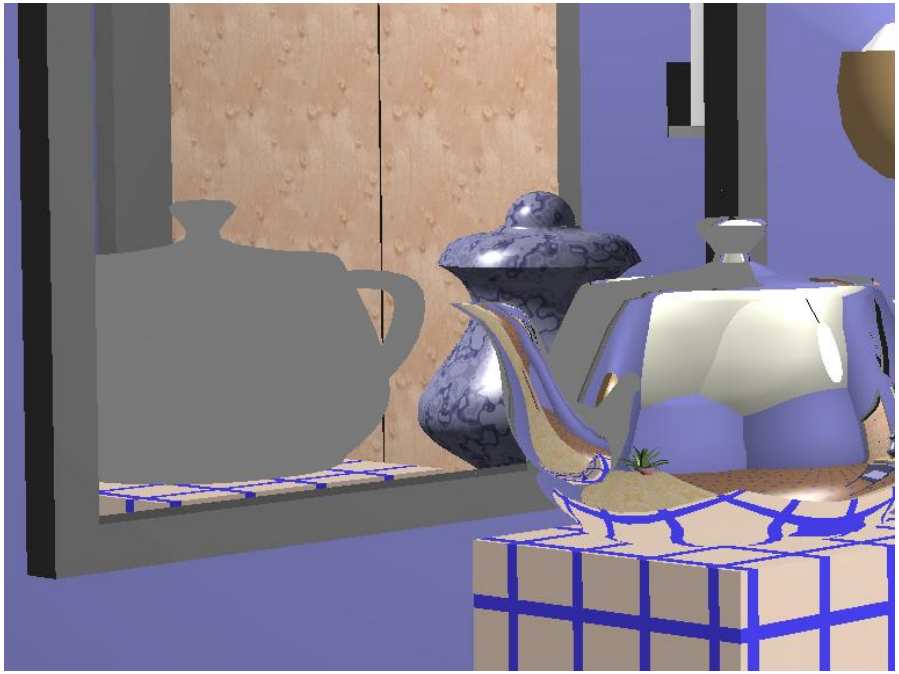


Iluminación por proyección de rayos
Ejemplo

Iluminación por proyección de rayos
Ejemplo

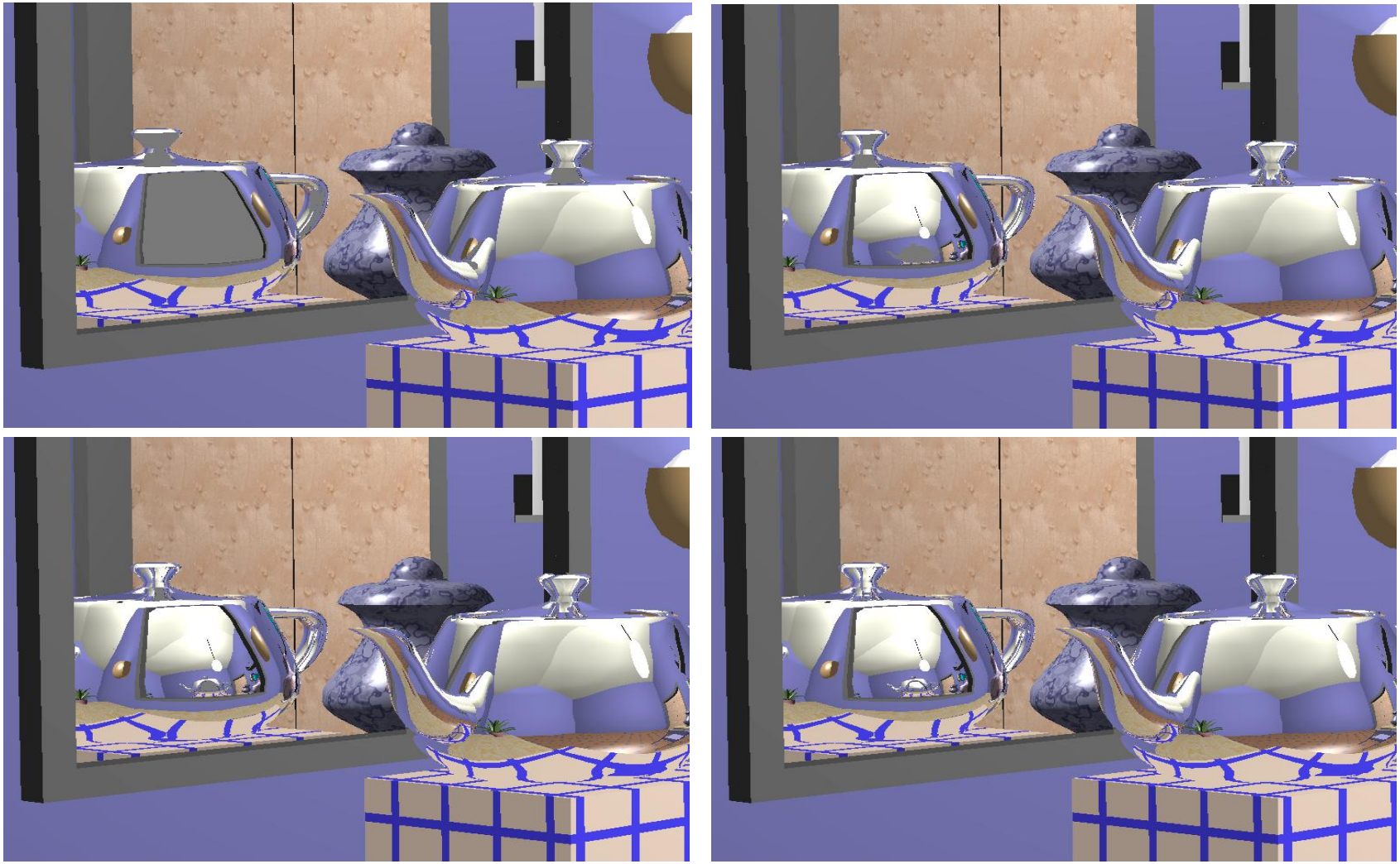


Profundidad cero

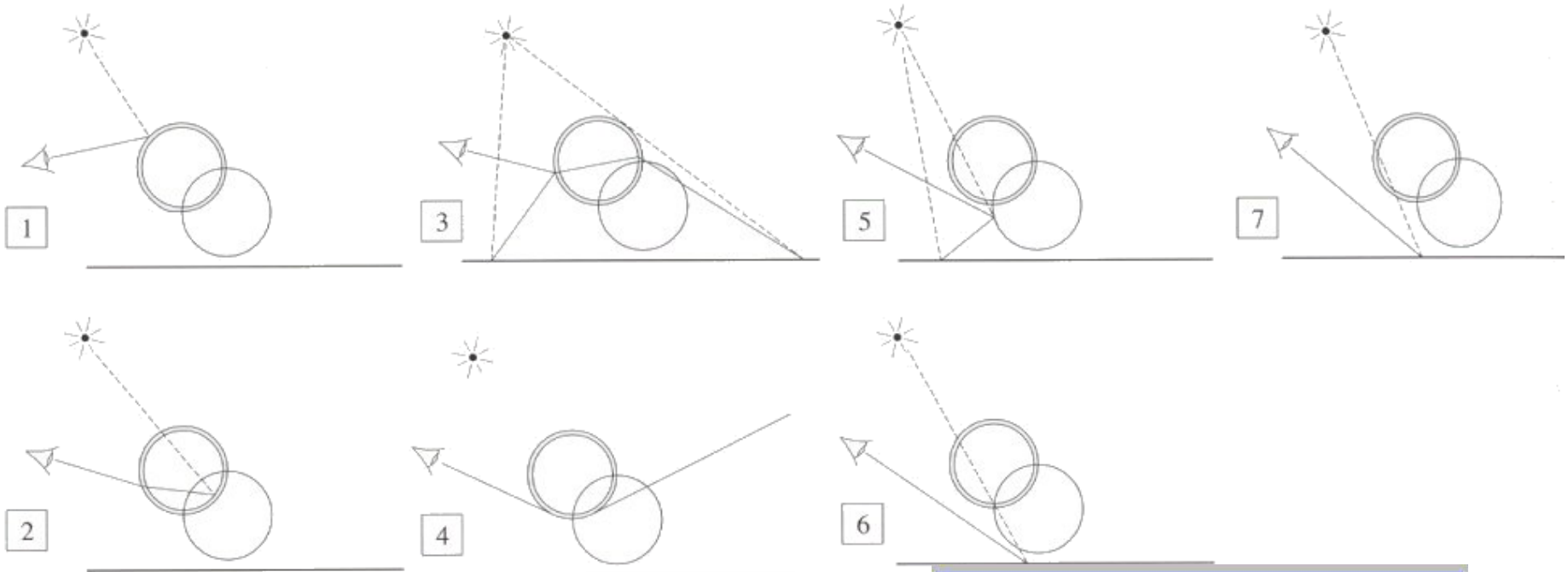


Profundidad uno

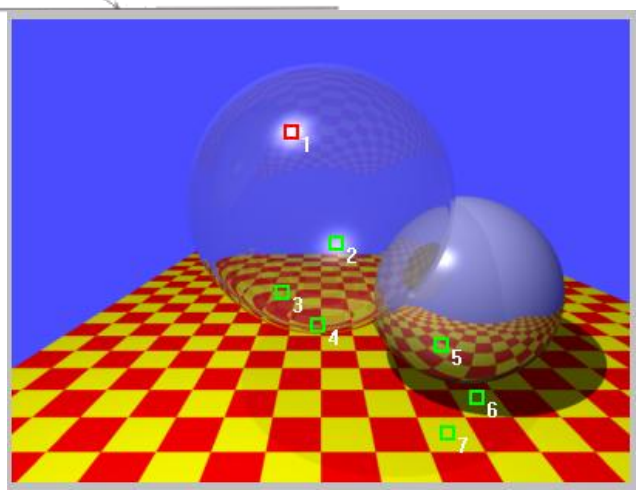
Iluminación por proyección de rayos
Ejemplo



Iluminación por proyección de rayos



El método también es útil para determinar la forma de las sombras



Radiosidad

- Modelo global (Luz directa a indirecta)
- Modela la interacción de la reflexión difusa
- Método independiente de la posición del observador
- Cada superficie es dividida en parches
- Se basa en la ley de conservación de la energía
- Se asume que todas las superficies son difusores perfectos
- Cada parche refleja luz recibida por cada uno de los demás parches
- Las fuentes de luz se modelan como parches que emiten luz propia

Radiosidad

- La interacción entre parches depende su relación geométrica
- Complejidad O(n^2) por lo que se procura hacer parches grandes
- Las sombras se calculan correctamente
- Es un algoritmo que trabaja en el espacio de definición de objetos ya que es independiente del observador
- Requiere de una etapa posterior de interpolación similar a la de Gourad para que no se noten los parches

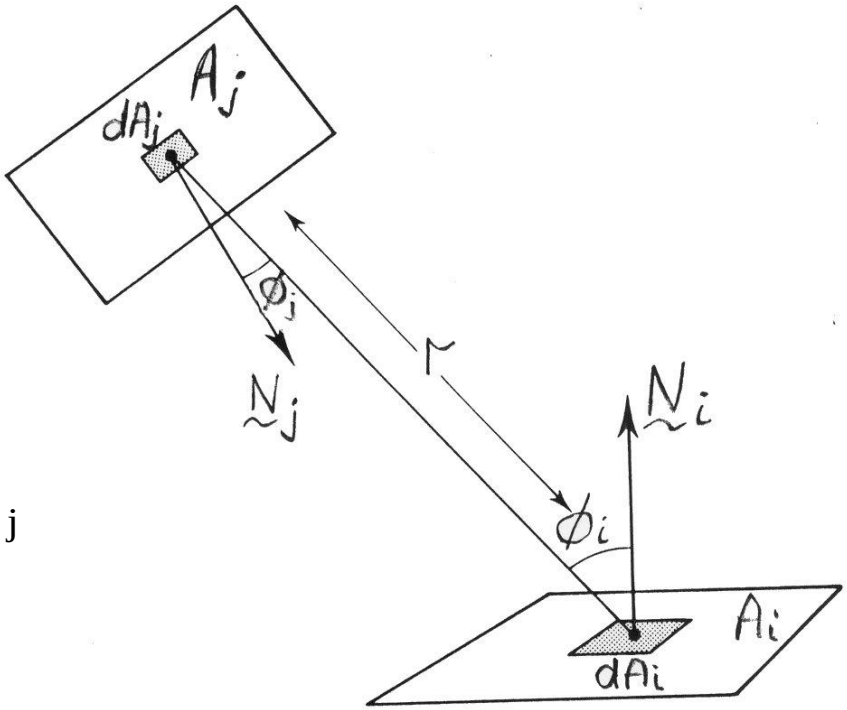
Radiosidad

Una vez que la escena se ha dividido en parches, se calcula la relación entre cada par de parches para calcular los “Factores de forma”

$$F_{AiAj} = F_{ij} = \frac{1}{A_i} \int_{Ai} \int_{Aj} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} dA_j dA_i$$

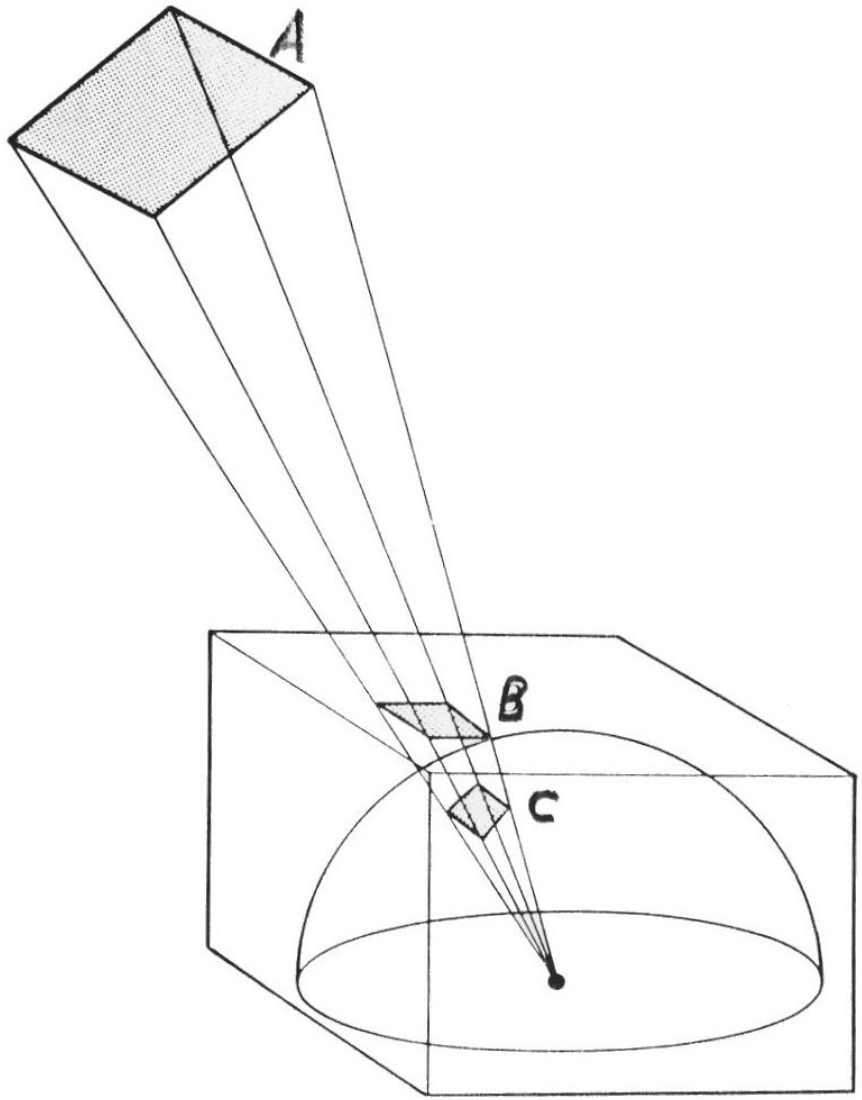
Si r es razonablemente grande

$$F_{ij} \approx F_{dAiAj} = \int_{Aj} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} dA_j$$



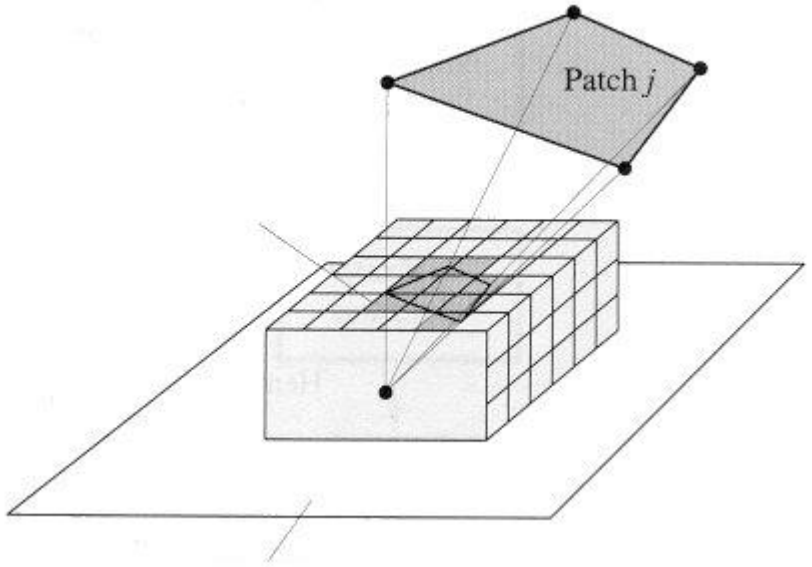
Radiosidad

La analogía de Nusselt dice que podemos considerar la proyección del parche j sobre la superficie de una semi-esfera centrada en el área elemental dAi como equivalente a considerar el parche mismo

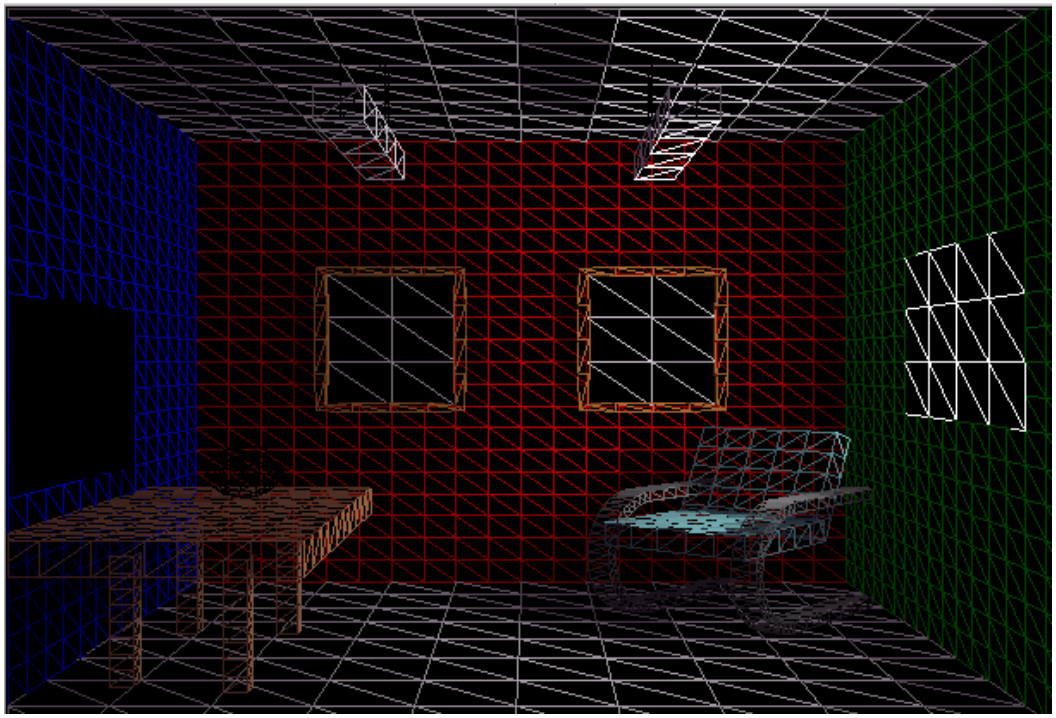


Radiosidad

Si usamos un semi-cubo es menos costoso calcular las proyecciones



Radiosidad
Ejemplo



Radiosidad
Ejemplo

Funciones OpenGL para Iluminación

- La Librería gráfica OpenGl no tiene implementada ninguna técnica de modelado global de iluminación
- Las funciones de iluminación de OpenGL que presentamos anteriormente son una modificación del modelo local de iluminación de Phong
- Recordar que un modelo local no contempla la luz indirecta
- Sin embargo, podemos implementar las técnicas de Radiosidad o de proyección de rayos y usar OpenGL para desplegar los resultados