ALGORITMOS DE ANTIALIASING

Ing. Gabriel Mañana Guichón[1]

Resumen

Se presentan los algoritmos desarrollados para el trazado de líneas suaves y aplicación de texto sobre imágenes, como ejemplo del trabajo de investigación que se lleva a cabo en el campo del filtrado de imágenes o antialiasing.

Introducción

Los elementos discretos que componen una imagen digital[2], llamados pixeles (de *pic*ture *el*ement), son generados por un proceso de muestreo de una imagen continua[3], en el dominio del espacio y en el dominio del color. Las muestras son tomadas en las posiciones determinadas por una retícula (la técnica se llama *point sampling*) definida en el espacio de los enteros (Figura 1).

Cuando se aplica una transformación espacial a una imagen (rotación, reducción, ampliación, etc.), los pixeles de la imagen resultante son generados con una nueva retícula de muestreo que por lo general no coincide con los números enteros (Figura 2). Dado que la imagen de entrada está definida solamente en posiciones enteras, se hace necesario aplicar un proceso de reconstrucción que interpole una función continua a partir de las muestras. Esta función puede ser entonces muestreada en posiciones arbitrarias, obteniéndose así los pixeles de la imagen final.

Este proceso de interpolación es llamado *reconstrucción de imágenes* y el proceso de reconstruc-

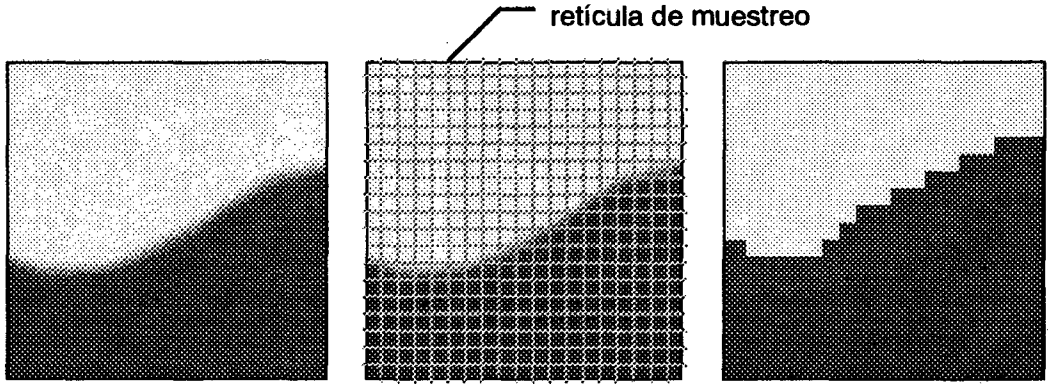


FIGURA 1. Retícula de muestreo utilizada en la digitalización de una imagen continua.

1 Departamento de Sistemas, Facultad de Ingeniería, Universidad Nacional de Colombia.

2 En este caso excluimos las imágenes sintéticas, i.e, . generadas por computador.

3 Transformaciones geométricas de imágenes digitales, L. F. Niño y G. Hernández.

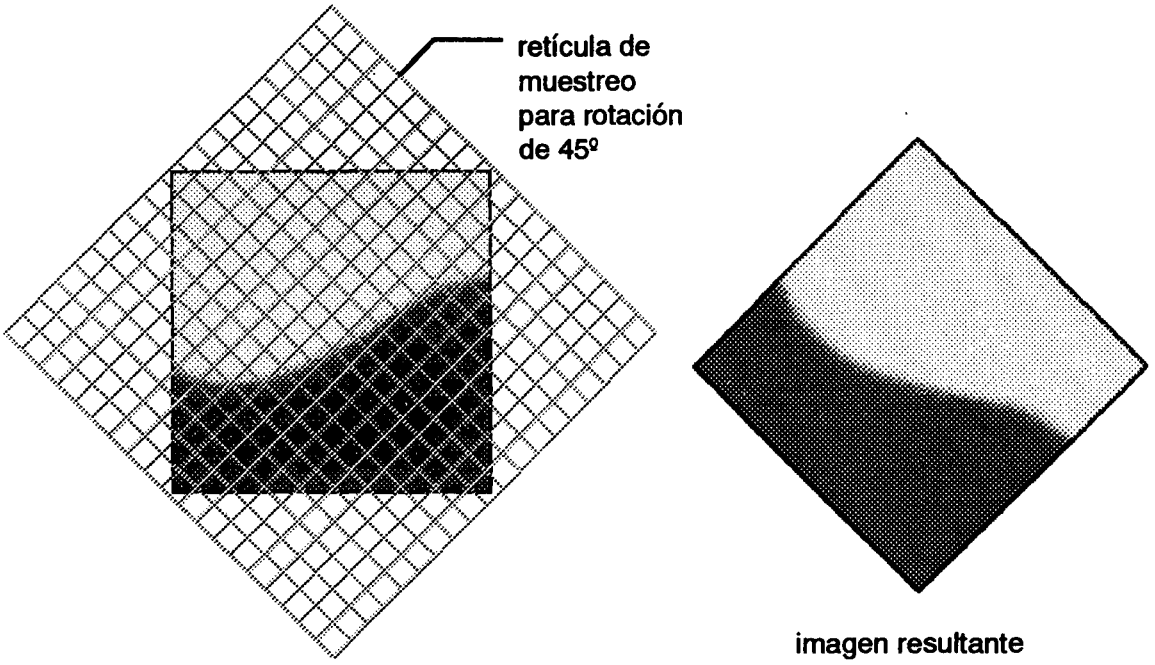


FIGURA 2. Ejemplo de una retícula de muestreo utilizada para una rotación.

ción seguido de un nuevo muestreo es llamado conjuntamente *remuestreo de imágenes.*

La siguiente figura (Figura 3) muestra el marco conceptual del procesamiento de imágenes digitales referenciado en este artículo. En un principio la imagen digital es obtenida por medio de un sistema electrónico de adquisición (básicamente puede estar compuesto de una cámara de video y un equipo conversor análogo-digital, donde hay una primera etapa de muestreo). En seguida pasa por una etapa de remuestreo compuesta de un proceso de reconstrucción de la señal continua y un nuevo proceso de muestreo en las posiciones deseadas. La posición exacta de las nuevas muestras estará determinada por la transformación espacial en particular que se aplique. Para evitar distorsiones en la imagen final se utilizan técnicas de filtrado digital (*antialiasing*) fundamentadas en la teoría del muestreo.

Teoría del muestreo

La teoría del muestreo es una herramienta fundamental en el estudio de sistemas digitales, específicamente en tareas tales como el procesamiento de imágenes digitales. Nos provee el fundamento matemático necesario para el análisis de señales discretas, dándonos una visión precisa de los problemas que surgen con el muestreo de señales continuas, así como una buena idea de las posibles soluciones. Para esto nos brinda una elegante formulación matemática que describe la relación existente entre una señal continua y sus muestras. En el caso que nos interesa esta formulación será utilizada en la solución de problemas relacionados con la reconstrucción de imágenes y la aparición de distorsiones o *aliasing*.

Por reconstrucción nos referimos al proceso de interpolación aplicado a las muestras y por *aliasing* a las distorsiones causadas por la presencia, en la señal continua, de altas frecuencias que no son reproducibles en la imagen resultante dadas las limitaciones impuestas por la resolución del dispositivo de salida (monitor, etc.).

Además de definir los límites teóricos del proceso de reconstrucción de una señal continua a partir de una entrada discreta, la teoría del

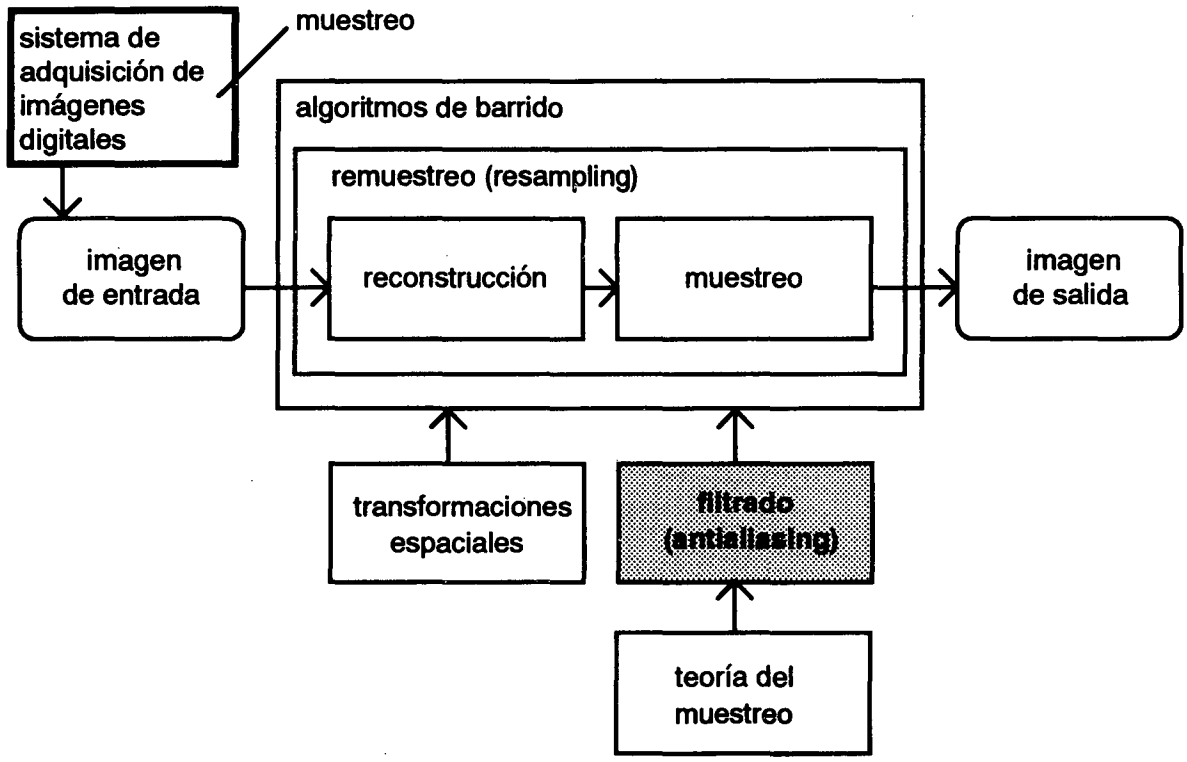


FIGURA 3. Marco conceptual del procesamiento digital de imágenes.

muestreo nos permite establecer la calidad de las diferentes técnicas de filtrado o *antialiasing*.

Aliasing

A través de las técnicas de reconstrucción de imágenes se resuelve uno de los primeros problemas que aparecen al trabajar en un dominio discreto: muestrear en posiciones arbitrarias una entrada originalmente discreta. Otro problema aparece sin embargo al evaluar la salida discreta. Este problema, que hace parte de la etapa de remuestreo, se describe a continuación.

La imagen resultante, como ya se ha visto, ha sido generada a partir de un muestreo puntual de la imagen de entrada reconstruida. Por muestreo puntual nos referimos a un proceso de muestreo "ideal" donde el valor de cada punto es tomado en forma independiente de sus vecinos. Esto es, cada punto en la imagen de entrada tiene influencia en un punto, y solo uno, de la imagen de salida.

Cuando se aplica una transformación espacial, con el muestreo puntual son descartados intervalos completos de la imagen de entrada. Si esta imagen de entrada varía en forma suave, la información perdida puede ser recuperada por medio de interpolación, esto es, reconstrucción. Sin embargo, si los intervalos descartados son lo suficientemente complejos (la imagen de entrada varía en forma abrupta en intensidad), el proceso de interpolación puede resultar deficiente y por ende la información descartada, irrecuperable. En este caso se dice que la señal o imagen de entrada ha sido submuestreada (*undersampled*) y cualquier intento de reconstrucción dará origen a distorsiones o *aliasing*. Estas distorsiones, causadas por la presencia de altas frecuencias no reproducibles, aparecen en forma de bordes escalonados o patrones de Moirè.

El filtrado empleado para contrarrestar estos efectos es llamado *antialiasing* y se fundamenta en los principios bien establecidos por la teoría del muestreo. Una técnica típica de filtrado se

basa en el desenfoque de la imagen de entrada *antes* de ser remuestreada. Esto permite que los puntos muestreados hayan sido influidos por los puntos vecinos que luego serán descartados. De esta forma las distorsiones pueden disminuirse pero no ser eliminadas. Una imagen de salida totalmente libre de distorsiones solo puede ser obtenida muestreando a una frecuencia lo suficientemente alta, como lo dicta la teoría del muestreo. Sin embargo, las limitaciones impuestas por la resolución del dispositivo de salida por lo general impiden esta alternativa y por lo tanto se recurre a "suavizar" la imagen de entrada antes de volverla a muestrear.

Si bien los principios de la teoría del muestreo ofrecen sendos fundamentos teóricos para afrontar el problema de las distorsiones, existen limitaciones prácticas en la implementación de los filtros ideales propuestos por esta teoría. De aquí que se hayan propuesto numerosos algoritmos para lograr soluciones aproximadas. En este artículo se presenta un algoritmo para el trazado de líneas suaves, basado en el conocido algoritmo de Bresenham y en una técnica de filtrado llamada EWA (*Elliptical Weighted Average*) propuesta por Greene y Heckbert en 1986, y un algoritmo para la reducción de los bordes escalonados en la aplicación de texto sobre imágenes, basado en una técnica llamada supermuestreo o posfiltrado. Antes, sin embargo, se hace una breve revisión de la teoría del muestreo y su aplicación en el filtrado de imágenes.

El problema básico enfrentado se conoce como *reconstrucción de señales* y puede ser planteado de la siguiente forma[4]:

- Dada una señal continua g(x) y su contraparte muestreada $g_s(x)$, ¿son suficientes las muestras que componen $g_s(x)$ para describir *exactamente* g(x)?
- Si este es el caso, ¿cómo puede ser reconstruida g(x) a partir de $g_s(x)$?

La solución se encuentra en el dominio de la frecuencia donde es utilizado el análisis espectral para examinar el espectro de la información muestreada. Las conclusiones derivadas del análisis de este problema serán útiles en la etapa de remuestreo y serán indicativas del filtrado necesario para evitar las distorsiones.

El importante *teorema del muestreo* nos permite relacionar la resolución de la retícula de muestreo con la naturaleza de la imagen, más específicamente con las frecuencias espaciales presentes en la imagen. En forma intuitiva se puede notar que cuanto más compleja (en términos de detalle) sea la imagen, más densa deberá ser la retícula para poder reproducir estos detalles. En términos formales el teorema del muestreo establece lo siguiente:

> Una señal continua univariada, limitada en frecuencia[5], puede ser reproducida completamente a partir de un conjunto de muestras tomadas a intervalos regulares. El intervalo entre las muestras debe ser menor a un medio del período de la componente de mayor frecuencia de la señal.

Esto último equivale a decir que la frecuencia mínima de muestreo debe ser mayor al doble de aquella de la componente de mayor frecuencia de la señal. Esta frecuencia mínima de muestreo, conocida como la *frecuencia de Nyquist*, corresponde a la distancia mínima entre las copias de los espectros de la señal. En la Figura 4 se pueden apreciar los efectos de muestrear una función sinusoidal a una frecuencia mayor que la frecuencia de Nyquist (a), a una frecuencia igual (b), a una frecuencia menor (c) y por último a una frecuencia bastante menor (d). En el primer caso el intervalo de muestreo es claramente menor a la mitad del período de la onda seno y por lo tanto no hay pérdida de información. En el segundo caso el

4 En esta parte se asume un conocimiento básico sobre la teoría de Fourier (análisis y síntesis).

5 Esto con el fin de evitar espectros de extensión infinita, imposibles de reproducir sin superposición.

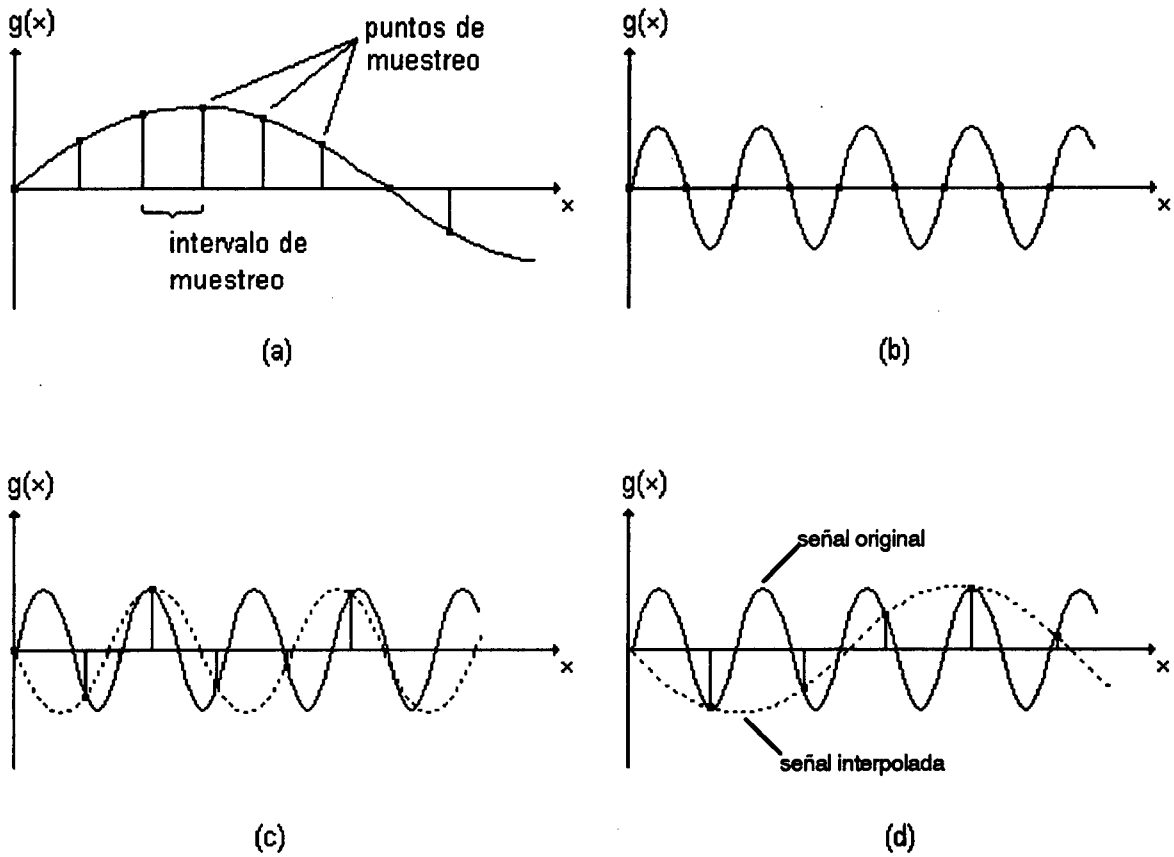


FIGURA 4. Representación en el dominio del espacio del muestreo de una onda seno.

intervalo de muestreo es igual a la mitad del período y de esta manera las muestras coinciden con el paso por cero de la onda seno. Es evidente que en este caso no se obtiene información alguna sobre la función muestreada (por los puntos de muestreo pasan infinitas ondas seno diferentes). Cuando el intervalo de muestreo es mayor que la mitad del período, casos (c) y (d), las muestras generan ondas seno de menor frecuencia que la onda original. Estas frecuencias menores son conocidas como *alias* de la onda original y de ahí la derivación del término *aliasing*.

Esta situación puede ser generalizada considerando estos casos en el dominio de la frecuencia para una función g(x) cualquiera. Esta función puede representar, por ejemplo, la variación (continua) de la intensidad en una línea de pixeles (*scanline*).

En la Figura 5 aparece la representación en el dominio de la frecuencia del muestreo y reconstrucción de una onda seno. En la parte (a) de la figura aparece el espectro de frecuencia de la señal de entrada g(x), donde se puede apreciar la distribución de energía en el ancho de banda, presentando una concentración en el rango de las bajas frecuencias.

En la parte (b) aparece el espectro de frecuencia de la función utilizada para el muestreo o filtro, consistente en una serie de líneas que se extienden (teóricamente) hasta infinito, separadas por f_s que es la frecuencia de muestreo. La función de muestreo puede ser la función impulso o función delta de Dirac, o la función delta de Kronecker si se está operando en un dominio discreto. Cuando una función impulso es aplicada a un filtro, un impulso modificado se pro-

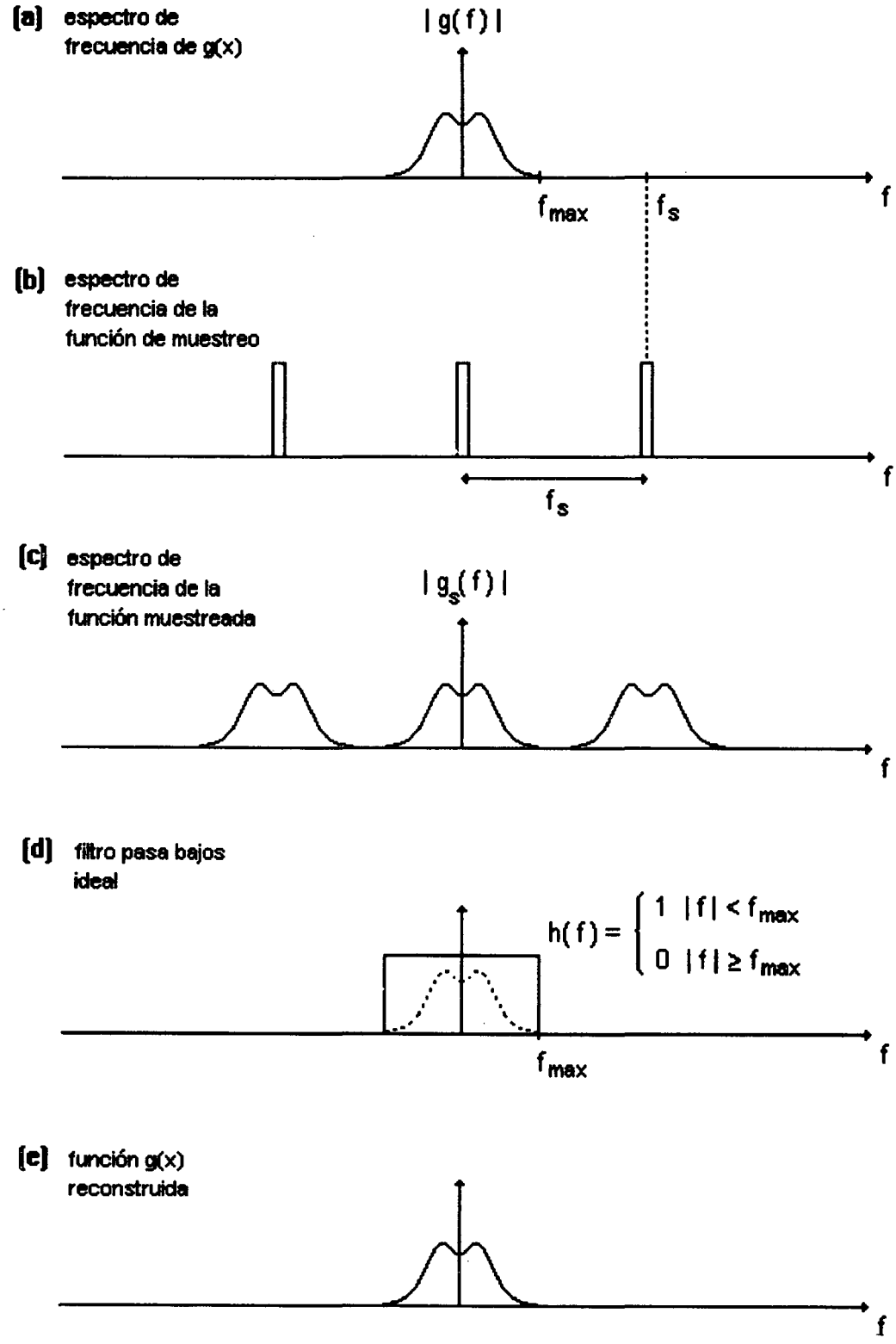


FIGURA 5. Muestreo y reconstrucción de una onda seno.

duce a la salida, conocido como *respuesta al impulso* del filtro.

En un sistema gráfico, la respuesta al impulso aparece como la imagen producida por el sistema, como respuesta a un punto (ideal) en la imagen de entrada. En este caso el impulso puede ser asimilado a un punto blanco, infinitesimalmente pequeño, sobre un fondo completamente negro.

Para muestrear en el dominio del espacio se multiplica g(x) por la función de muestreo. El proceso equivalente en el dominio de la frecuencia es llamado *convolución* e involucra al espectro de frecuencia de la función de muestreo y al espectro de frecuencia de g(x). El resultado –convolución de (a) y (b)– es el espectro de frecuencia que se ve en la parte (c), espectro de la versión muestreada de g(x). Esta versión muestreada es entonces multiplicada por un filtro de reconstrucción, o *núcleo de convolución* en el dominio de la frecuencia (d), para reproducir la señal original (e).

En la Figura 6 se muestra el proceso anterior pero ahora utilizando una frecuencia de muestreo menor que la frecuencia de Nyquist. En la parte (c) se puede ver que en el espectro de frecuencia de la versión muestreada aparecen los espectros de frecuencia de g(x) traslapados. De esta manera, las altas frecuencias (correspondientes a detalles en la imagen) aparecen como interferencia en las regiones de bajas frecuencias (e), equivalentes a distorsiones en la imagen de salida.

Estas distorsiones pueden ser reducidas de dos maneras diferentes. La primera consiste en aumentar la frecuencia de la retícula de muestreo, esto es, aumentar la resolución de la matriz de pixeles utilizada (monitor, etc.). Esto conlleva desventajas técnicas y económicas obvias. Por otra parte, el espectro de frecuencia de una imagen puede extenderse infinitamente, por lo que aumentar la frecuencia de muestreo no necesariamente resuelve el problema.

La segunda forma se basa en la aplicación de *filtros digitales* que atenúan las altas frecuencias presentes en la imagen, tratando de ajustar la señal a una frecuencia de muestreo predeterminada (resolución). En computación gráfica se utilizan básicamente dos métodos para eliminar o reducir las distorsiones causadas por el *aliasing:* el primero es conocido como muestreo por áreas (*area sampling*) o prefiltrado y el segundo como supermuestreo o posfiltrado (*supersampling*). Un tercer método, aún en investigación, es el llamado muestreo irregular, no uniforme o estocástico, donde se utiliza una retícula en la cual la posición de los puntos de muestreo ha sido determinada probabilísticamente (mediante una técnica de tipo Monte Carlo, por ejemplo). Dentro de este enfoque se han desarrollado algoritmos tales como el muestreo de Poisson (*Poisson Sampling*) y el muestreo por difusión de puntos (*Point-difussion Sampling*) entre otros.

Los algoritmos presentados más adelante son ejemplos de técnicas de muestreo por áreas –trazado de líneas suaves– y de posfiltrado-eliminación de bordes escalonados en la aplicación de texto. A continuación se presenta una breve revisión de estas dos técnicas de filtrado.

Prefiltrado o muestreo por áreas

La idea clave detrás de la técnica del muestreo por áreas está en considerar que el pixel de salida representa un área y no un punto como se asume en el muestreo puntual (*point sampling*). Este debe ser visto entonces como una ventana a la imagen de entrada. En vez de muestrear un punto, se aplica un filtro pasabajos sobre el área proyectada y se determina así el contenido de información que está siendo efectivamente mapeado al pixel en la imagen de salida. El filtro pasabajos lleva a cabo la etapa de *prefiltrado* y el área proyectada se conoce como *preimagen*. Al limitar la frecuencia de la imagen de entrada *antes* de ser remuestreada, el filtro ayuda a disminuir las distorsiones.

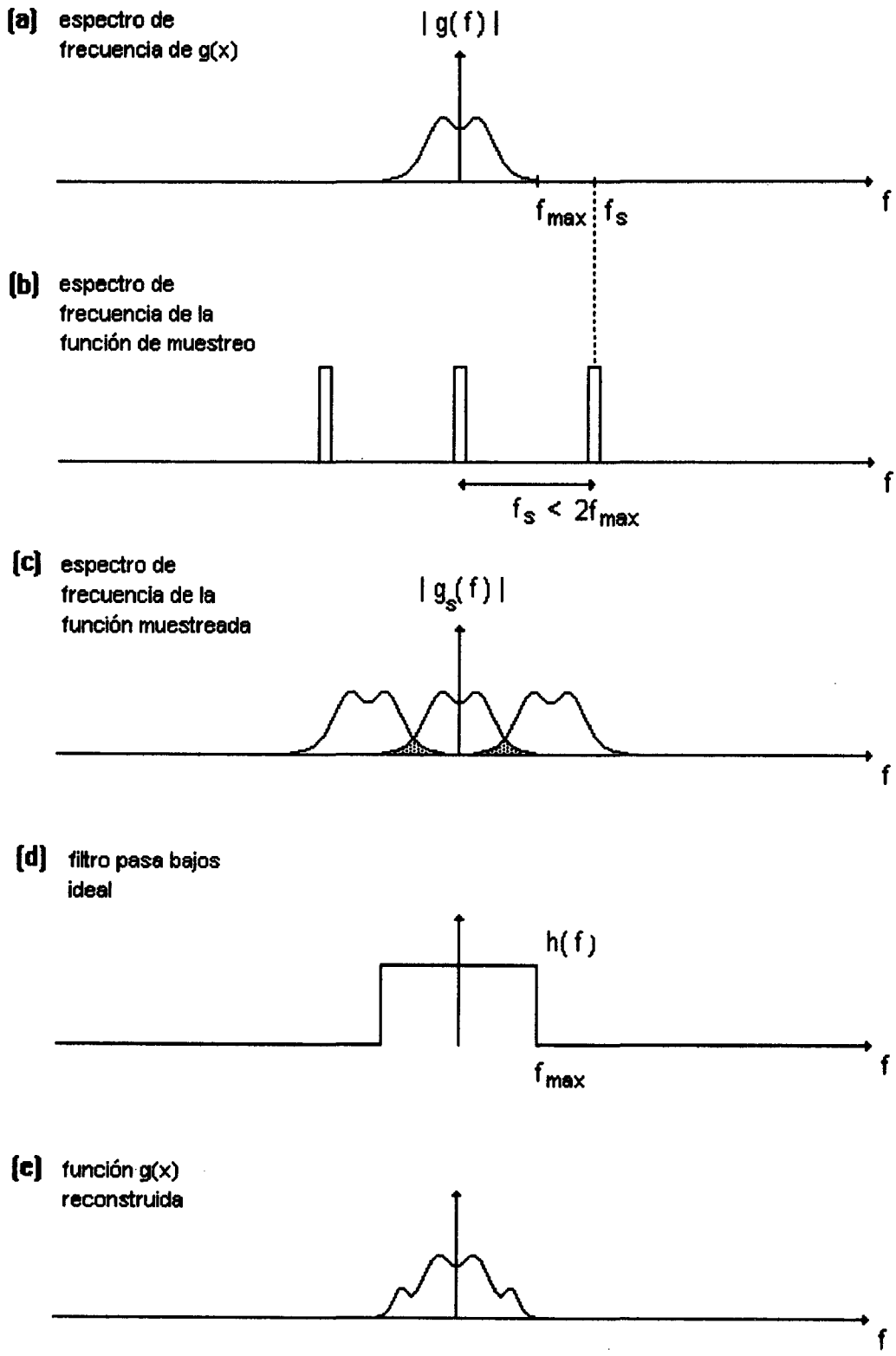


FIGURA 6. Muestreo y reconstrucción de una onda seno, utilizando una frecuencia de muestreo inadecuada ($f_s < 2f_s$).

Esta técnica fue desarrollada por Catmull en 1978. Si bien la técnica original planteada por Catmull es prohibitiva en términos computacionales, ésta ha dado origen a múltiples enfoques prácticos.

El algoritmo se basa esencialmente en trabajar a nivel de sub-pixel en el dominio continuo de la imagen, calculando el área efectiva del pixel que es cubierta por el elemento gráfico (línea, polígono, etc.) y utilizando este valor como factor de intensidad. En el caso particular que nos interesa (línea), se utiliza la siguiente función[6] para calcular la fracción de círculo cubierta por un semiplano (Figura 7):

Ahora, teniendo en cuenta que r = 1/2, el área de intersección entre una línea de ancho 2a y un pixel se puede expresar en términos de la función cov (d) y en términos del ancho de la línea:

TABLA 1
Intersección línea-pixel*

	rango de p	fracción de círculo
a < r		
	0 ≤ p < a	1 - cov(a-p) - cov(a+p)
	a ≤ p < r-a	cov(p-a) - cov(p+a)
	r-a ≤ p	cov(p-a)
a ≥ r		
	0 ≤ p < a	1 - cov(a-p)
	a ≤ p	cov(p-a)

* Kelvin Thompson, Nth Graphics, Ltd., Austin, Texas.

La fracción obtenida multiplicada por el área del círculo (πr^2), representa el área efectiva de intersección.

La razón de modelar el pixel de salida (p.e. monitor) como un círculo corresponde a la técnica

$$\text{cov}(d, r) = \begin{cases} d >= r, & 0 \\ d < r, & \frac{1}{2} - \frac{d\sqrt{r^2 - d^2}}{\pi r^2} - \frac{1}{\pi}\arcsen\frac{d}{r} \end{cases}$$

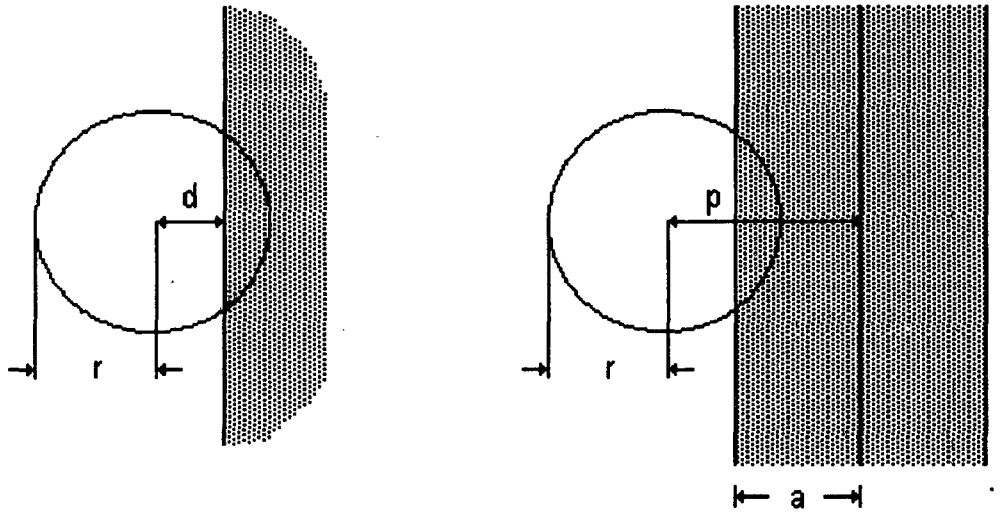


FIGURA 7.

6 Resultado de integrar la ecuación del semicírculo entre *d* y *r*.

de filtrado propuesta por Greene y Heckbert en 1986 (con orígenes en Feibush, Levoy y Cook, 1980), y la razón se encuentra en el hecho de que el mapeo inverso de un círculo (su preimagen en el espacio de la textura) siempre es una elipse. Sin importar el tamaño, ecentricidad u orientación, la forma es invariante y este hecho es explotado por el método. Esta técnica está basada en algoritmos desarrollados para el mapeo de texturas.

Supermuestreo o posfiltrado

El proceso de utilizar más de una muestra por pixel es conocido como supermuestreo. Se utiliza una retícula de muestreo más densa que la retícula de salida. Si se tiene una relación de 3 a 1, por ejemplo, se tienen 9 muestras por cada pixel de salida. El valor de este pixel se calcula a partir del promedio de las muestras tomadas en las respectivas preimágenes.

Formalmente el método consiste en tres etapas, pero en la práctica la segunda y la tercera se combinan en una sola:

1. El dominio continuo de la imagen es muestreado a *n* veces la resolución del monitor o dispositivo de salida. En la práctica esto significa que la imagen es generada a una resolución *n* veces mayor que la resolución del monitor.

2. La imagen muestreada es sometida a un filtro pasa bajos a la frecuencia de Nyquist del monitor.

3. La imagen filtrada es remuestreada a la resolución del monitor.

Los algoritmos desarrollados dentro de esta técnica difieren en el valor de *n* y en la forma del filtro utilizado. Usualmente para una resolución de 512 x 512 se utiliza un supermuestreo de 2048 x 2048 (n = 4). La imagen de alta resolución generada es reducida a la resolución final de 512 x 512 promediando el valor de los pixeles y esto equivale a aplicar una convolución con un filtro cuadrado (*box filter*). Aplicando un filtro cuyos pesos cambian a través de su núcleo se pueden obtener mejores resultados. Crow (1981) por ejemplo plantea la utilización de "ventanas de Barlett", tres de la cuales se muestran en la siguiente tabla:

TABLA 2
Ventanas de Barlett utilizadas en el posfiltrado de una imagen supermuestreada

3 × 3	5 × 5	7 × 7
1 2 1 2 4 2 1 2 1	1 2 3 2 1 2 4 6 4 2 3 6 9 6 3 2 4 6 4 2 1 2 3 2 1	1 2 3 4 3 2 1 2 4 6 8 6 4 2 3 6 9 12 9 6 3 4 8 12 16 12 8 4 3 6 9 12 9 6 3 2 4 6 8 6 4 2 1 2 3 4 3 2 1

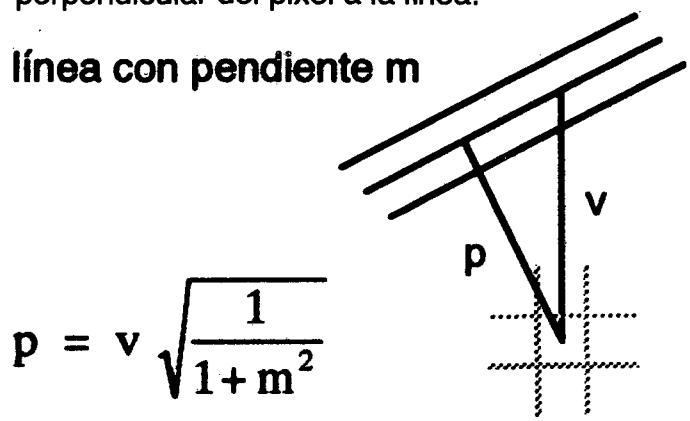
El mecanismo radica en centrar la ventana sobre un pixel de la supermuestra y calcular la suma de los productos de cada pixel por el peso correspondiente en núcleo del filtro. Este valor, dividido por la suma de los pesos del núcleo, es el valor del pixel correspondiente en la imagen de salida. Desplazando la ventana a través de la imagen de alta resolución se obtienen los pixeles de la imagen filtrada.

Un efecto secundario del supermuestreo está en el desenfoque de la imagen original. Esto ocurre a causa de la integración que se lleva a cabo entre los vecinos de cada pixel. Esto significa que existe un compromiso en la elección del tamaño del filtro. Un filtro grande tendrá una frecuencia de corte más baja y por lo tanto será más eficiente al reducir las distorsiones causadas por el *aliasing*. Sin embargo, el efecto de desenfoque será mayor y adicionalmente será más costoso en términos computacionales.

Algoritmo para el trazado de líneas suaves

El algoritmo se basa en calcular la intensidad de cada pixel de acuerdo al área que es cubierta por la línea. Para esto se utiliza una extensión del algoritmo de Bresenham y la función desarrollada en la sección 3 (Tabla 1). En cada iteración, el algoritmo usual avanza un pixel en la dirección del eje mayor y cero o uno en la dirección del eje menor, para determinar la posición del siguiente pixel. En el ejemplo de la Figura 8, el eje mayor es el eje X (pendiente en el rango [-1, 1]).

cular esta distancia se utiliza la relación que existe entre la distancia vertical y la distancia perpendicular del pixel a la línea:



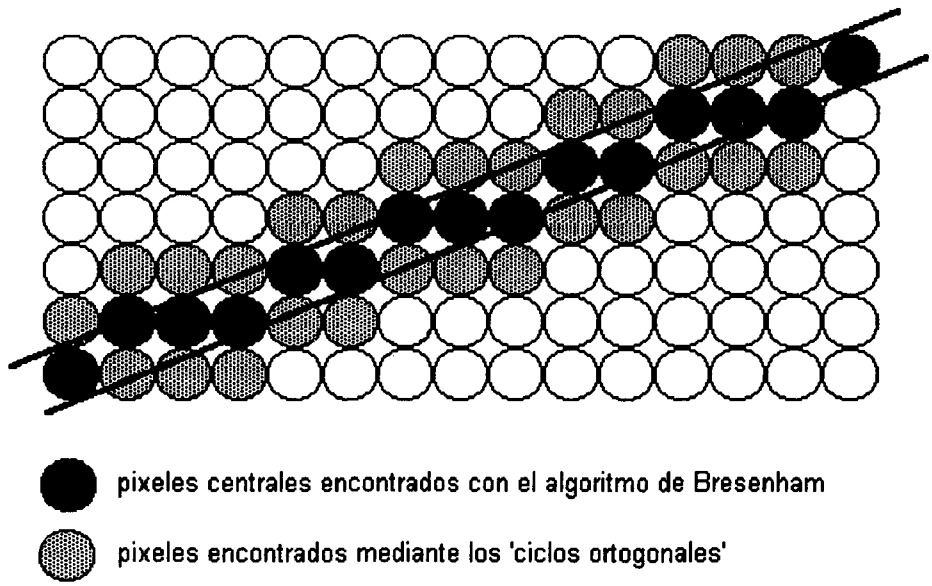


FIGURA 8.

A este algoritmo se le agregan dos ciclos, llamados "ciclos ortogonales", dentro del ciclo principal. Una vez determinada la posición del pixel central, el primer ciclo examina los pixeles adyacentes en la dirección positiva del eje menor y el segundo examina los pixeles adyacentes en la dirección negativa. En cada pixel visitado, el algoritmo calcula la distancia que hay desde el centro del pixel a línea central del segmento que se está pintando. Esta distancia es utilizada (generalmente a través de una tabla) para calcular el área efectiva del pixel que es cubierta por el segmento de línea. Para cal-

```
// línea con antialiasing:  basado en el algoritmo
                            de Kelvin Thompson,
//                          Nth Graphics, Austin, Texas.
//
// equipo:                  Silicon Graphics, Personal Iris 35
// pixeles:                 RGBA/8 bits@canal
// enteros:                 32 bits

// PIXEL
typedef unsigned char BYTE;
typedef struct PIXEL PIXEL;
struct PIXEL {
        BYTE r;             // rojo
        BYTE g;             // verde
        BYTE b;             // azul
        BYTE a;             // opacidad
```

```
};

// punto fijo
typedef int FXD;

// macros
# define FXD_FRACCION          16
# define FXD_ZERO              0
# define ANCHO_MAX             1280                    // monitor: 1024 x 1280

# define SWAP (x, y)           ( (x)^=(y)^=(x)^=(y) )  // intercambia dos enteros
# define ADDR (x, y)           ( (x)+(y*ANCHO_MAX) )   // 2D -> 1D

                               // tablas precalculadas
# define AREA (d)              area [d]                // d en [0, 255]
# define VP_FACTOR (m)         factor [m]              // dist. vertical -> dist. perp.
                                                       // sqrt (1/(1+m^2) )

# define FXDMUL (fx1, fx2)     ( (fx1 << FXD_FRACCION)*fx2)  // multiplicación punto fijo

# define BLEND (cl, cf)        ( (((255-(cl))*(cf)) >> 8) + (cl))  // cl: color línea {0, 255}
                                                                   // cf: color fondo {0, 255}

# define DY_GT_DX              1                       // abs (dy) > abs (dx)
# define DY_LT_ZERO            2                       // dy < 0

// incremento p/pixeles adyacentes
static int incAdy {4} = {ADDR(1, 0), ADDR(0, 1), ADDR(1, 0), ADDR(0, -1)};

// incremento p/pixeles diagonales
static int incDiag {4} = {ADDR(1, 1), ADDR{1, 1}, ADDR(1, -1), ADDR(1, -1)};

// incremento p/pixeles ortogonales
static int incORT {4} = {ADDR(0, 1), ADDR(1, 0), ADDR(1, 0)};

extern FXD fxPmax;             // máxima distancia perpendicular;
                               // 1/2 ancho de línea + radio del pixel

// línea c/antialiasing del punto (xi, yi) al punto (xf, yf),
// color (r, g, b), opacidad a, sobre imagen en buffer pBuf

void
línea_aa (int xi, int yi, int xf, int yf, BYTE r, BYTE g, BYTE b, BYTE a, PIXEL* pBuf)
{
```

```
int         nBres;              // variable de decisión, algoritmo de Bresenham
int         nAdyInc;            // incremento adyacente p/nBres
int         nDiagInc;           // incremento diagonal p/nBres

int         nAdyDirInc;         // incremento en la dirección p/pixel adyacente
int         nDiagDirInc;        // incremento en la dirección p/pixel diagonal
int         nOrtDirInc;         // incremento en la dirección p/pixel ortogonal

FXD         fxDistBres;         // distancia perpendicular, pixel Bresenham
FXD         fxIncAdyBres;       // incremento adyacente, pixel Bresenham
FXD         fxIncDiagBres;      //  incremento diagonal, pixel Bresenham
FXD         fxDistCur;          // distancia perpendicular, pixel visitado
FXD         fxIncOrtCur;        // incremento ortogonal, pixel visitado

int         nDX;                // delta en x
int         nDY;                // delta en y
int         nDir;               // categoría según pendiente dy/dx

FXD         fxPendiente;        // pendiente dy/dx

PIXEL*      pBres;              // dirección pixel Bresenham
PIXEL*      pCur;               // dirección pixel visitado

register float fOP              = (float)a/255.0;           // opacidad línea {0, 1}
register float fFACT;

if (xi > xf)                    // siempre de izquierda a derecha
{
        SWAP (xi, xf);
        SWAP (YI, YF);
}

// DELTAS
nDX = xf - xi;                  // dx > 0
nDY = yf - yi;

// categoría s/pendiente
nDir = 0;                       // 0 ≤ m < 1
if (nDY < 0)                    // m < 0
{
        nDir I=DY_LT_ZERO;
        nDY = -nDY;             // siempre m > 0 (simetría)
}
if (nDY > nDX)                  // abs(m) > 1
{
        nDir I=DY_GT_DX;
        SWAP(nDX, NDY);         // siempre primer cuadrante (simetría)
```

```
// direcciones iniciales
pBres        = pBuf + ADDR( (long)xi, (long)yi );
nAdyDirInc   = incAdy[nDir];
nDiagDirInc  = incDiag[nDir];
nOrtDirInc   = incOrt[nDir];

// distancias perpendiculares
fxPendiente  = (nDY << FXD_FRACCION) / nDX;
fxIncOrtCur  = VP_FACTOR (fxPendiente);
fxIncAdyBres = FXDMUL (fxPendiente, fxIncOrtCur);
fxIncDiagBres = fxIncOrtCur - fxIncAdyBres;
fxDistBres   = FXD_ZERO;

// Bresenham
nAdyInc      = nDY << 1;
nDiagInc     = (nDY - nDX) << 1;
nBres        = nAdyInc - nDX;

do {
    // pixel central
    fFACT        = fOP*AREA (abs (fxDistBres));
    pBres ->r    = BLEND ((BYTE) (fFACT*r), pBres->r);
    pBres ->g    = BLEND ((BYTE) (fFACT*g), pBres->g);
    pBres ->b    = BLEND ((BYTE) (fFACT*b), pBres->b);

    // ciclo ortogonal ↑
    for ( fxDistCur = fxIncOrtCur - fxDistBres, pCur = pBres + nOrtDirInc;
          fxDistCur < fxPmax;
          fxDistCur += fxIncOrtCur;          pCur += nOrtDirInc )
    {
        fFACT        = fOP*
        pBres -> r   = BLEND ((BYTE) (fFACT*r), pBres-> r);
        pBres-> g    = BLEND ((BYTE) (fFACT*g), pBres-> g);
        pBres-> b    = BLEND ((BYTE) (fFACT*b), pBres-> b);
    }

    // ciclo ortogonal ↓
    for ( fxDistCur = fxIncOrtCur + fxDistBres, pCur = pBres - nOrtDirInc;
          fxDistCur < fxPmax;
          fxDistCur += fxIncOrtCur;          pCur = nOrtDirInc)
    {
        fFACT        = fOP* AREA(abs(fxDistCur));
        pBres->r     = BLEND ((BYTE) (fFACT*r), pBres->r);
        pBres->g     = BLEND ((BYTE) (fFACT*g), pBres->g);
        pBres->b     = BLEND ((BYTE) (fFACT*b), pBres->b);
```

```
        }

        // próximo pixel s/algoritmo de Bresenham
        if (nBres < 0)
        {       nBres += nAdyInc;
                pBres += nAdyDirInc*sizeof(PIXEL);
                fxDistBres += fxIncAdyBres;

        }
        else
        {
                nBres += nDiagInc;
                pBres += nDiagDirInc*sizeof(PIXEL);
                fxDistBres += fxIncDiagBres;
        }

        -nDX;

    }   while (nDx >= 0);

}   // línea_aa
```

Algoritmo para la aplicación de texto con filtrado

El algoritmo está basado en la técnica presentada en la sección 4. A continuación se analizan las partes más relevantes del código desarrollado para un programa que se utiliza en la generación de fotodocumentos bajo ambiente MS-Windows. En esta aplicación se debía solucionar el problema de aplicar texto sobre imágenes, en cualquiera de las fuentes disponibles y con suavizado de bordes escalonados.

En primer lugar se aplica el texto en un dispositivo de memoria (CreateCompatibleBitmap) KERNEL_SIZE veces más grande que el área de texto original. KERNEL_SIZE es el tamaño del núcleo de filtrado utilizado y puede ser 3, 5, 7, o un valor mayor, dependiendo de la calidad final deseada, equipo destino, etc.. Aquí se debe tener en cuenta que luego, al hacer el remuestreo de la imagen para retornar a la resolución de pantalla, el número de multiplicaciones y sumas crece según el cuadrado del tamaño del núcleo. Adicionalmente, y como se mencionó antes, el efecto de desenfoque también crece con el tamaño del núcleo.

```
// se aplica el texto en un dispositivo de memoria

HDC hdc = GetDC (pOwner->HWindow);

// crea un 'device context' compatible con el monitor
HDC hMemDC = CreateCompatibleDC(hdc);
if (hMemDC == NULL)
{
        return FALSE;
}

// ancho/alto de área de texto
int nTextWidth = rText.righ - rText.feft;
int nTextHeigth = rText.bottom - rText.top;
```

```
// se crea un bitmap compatible, KERNEL_SIZE veces
   más grande
nTextWidth *= KERNEL_SIZE;
nTextHeight *= KERNEL_SIZE;

HBITMAP hBitmap = CreateCompatibleBitmap(
                              ⌂ hdc,
                                nTextWidth,
                                nTextHeight);
if (hBitmap == NULL)
{
        DeleteDC(hMemDC);
        return FALSE;
}

// se selecciona el bitmap creado como objeto gráfico
en uso
HBITMAP hPrevObject = SelectObject(hMemDC, hBitmap);

// se borra el bitmap
BitBit (hMemDC,
        0,
        0,
        nTextWidth,
        nTextHeight,
        hdc,
        0,
        0,
        BLACKNESS);
// se pinta el texto en el dispositivo de memoria
pText->apply(pOwner->HWindow,
             hMemDc,
             rText.left,
             rText.top,
             SRCCOPY);
// recupera objeto gráfico previo,
// libera dispositivo de memoria
SelectObject(hMemDc, hPrevObject);
DeleteDC(hMemDC);
```

La función miembro apply(), de la clase Text, simplemente agranda el tamaño de la letra acorde a KERNEL_SIZE y luego pinta sobre el dispositivo destino:

```
.
.
.
LOGFONT lf;
_fmemcpy (&lf, &lfFont, sizeof(LOGFONT));
if (antiAlias())
{
        lf.lfHeight*=KERNEL_SIZE;
}
HFONT hfFont = CreateFontIndirect(&lf);
HFONT hfPrevFont = SelectObject(hdc, hfFont);
SetTextColor(hdc, getColor());
SetBkMode(hdc, TRANSPARENT);
TextOut (       hdc,
                r.left - nHscrollPos,
                r.top - nVscrollPos,
                szText,
                strlen(szText));
SelectObject(hdc, hfPrevFont);
DeleteObject(hfFont);
```

Una vez generada la imagen con el texto, de alta resolución, se le aplica un proceso de desenfoque (blur) de radio de efecto 2, 3, o mayor, dependiendo como siempre del equipo destino, tamaño de la letra, estilo (normal, negrita, etc.) y calidad final deseada.

Una vez aplicado el proceso de desenfoque se debe reducir la imagen a la resolución del monitor. Para esto se utiliza un núcleo de interpolación apropiado, una ventana de Barlett por ejemplo (ver Tabla 2), que se hace pasar sobre la imagen, avanzando KERNEL_SIZE en cada paso. Cada vez que el núcleo es centrado sobre la imagen se toman KERNEL_SIZE*KERNEL_SIZE muestras, que promediadas según los pesos del núcleo conforman el valor del pixel en la imagen de salida.

Esta imagen con el texto ya suavizado se compone fácilmente con la imagen de fondo, pixel a pixel, teniendo en cuenta el factor de composición asignado (opacidad) para el texto.

BIBLIOGRAFIA

ANGEL, Edward, *Computer Graphics*, Addison-Wesley, MA, 1990.

BRESENHAM, J.E., *Ambiguities in Incremental Line Rastering*, IEEE Computer Graphics and Applications, 7(5), pp. 31-43, 1987.

CATMULL, E., *A Hidden-Surface Algorithm with Antialiasing*, Computer Graphics, 12(3), pp. 6-11, 1975.

CROW, F.C., *A Comparison of Antialiasing Techniques*, IEEE Computer Graphics and Applications, 1(1), pp. 40-49, 1981.

HARRINGTON, S., *Computer Graphics: A Programming Approach*, McGraw-Hill, NY, 1983.

NIÑO, LUIS., y HERNÁNDEZ, GERMÁN., "Transformaciones Geométricas de Imágenes Digitales", *Memorias del Tercer Encuentro de Geometría y sus Aplicaciones*, Universidad Pedagógica, 1993.

THOMPSON, Kelvin, *Graphics Gems*, Academic Press, MA, pp. 40-48, 105-106, 1990.

WATT, Alan, *Fundamentals of Three-Dimensional Computer Graphics*, Addison-Wesley, GB, 1989.

WOLBERG, George, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.