Una técnica de filtrado selectivo para reducir el *aliasing* en imágenes sintéticas

Francisco Morán Burgos

Grupo de Tratamiento de Imágenes
Depto. de Señales, Sistemas y Radiocomunicaciones
ETSI Telecomunicación - Universidad Politécnica de Madrid
Ciudad Universitaria s/n - 28040 Madrid
Tel: (91) 336.7353
Fax: (91) 336.7350
E-mail: **fmb@gti.ssr.upm.es**
WWW: **http://www.gti.ssr.upm.es/~fmb**

Resumen

La lucha contra los perniciosos efectos del *aliasing* empezó hace al menos veinte años en el campo de las imágenes sintéticas y ha sido desde entonces una de las principales preocupaciones para los diseñadores de algoritmos de visualización. En este artículo se propone una variante de uno de los más sencillos, el clásico *z-buffer*, para incorporarle un filtrado posterior que reduzca los defectos debidos al submuestreo. Con un pequeño aumento del tiempo de cálculo y de la memoria necesarios para el algoritmo original, se consigue filtrar aquellas zonas donde esos defectos son más importantes y aumentar considerablemente la resolución aparente de las imágenes.

Palabras clave

Teoría de muestreo, *aliasing*, filtrado adaptativo; imágenes sintéticas, *rendering*, métodos de eliminación de superficies ocultas, *z-buffer*.

Agradecimientos

De los retoques de la idea original es co-responsable José Javier García Aranda, cuya ayuda en la programación y cuyos ánimos constantes —"¡esta vez sí que ya está!"— han sido de gran importancia. A la redacción final de este artículo han contribuido Narciso García Santos y José María Martínez Sánchez revisando pacientemente sus distintas versiones. A los tres, gracias.

0. Introducción

La generación de imágenes sintéticas, como todo proceso que involucre imágenes digitales, implica necesariamente una etapa de muestreo, con la que se logra codificar una señal bidimensional continua mediante una matriz de píxeles. Esta aproximación reduce doblemente el espacio de la señal original, al discretizar por una parte su resolución espacial y por otra su rango de valores. Lo segundo se suele paliar empleando 24 o más bits para codificar el color de cada píxel, logrando así una gama de más de 16 millones de colores, percibida como virtualmente continua por el ojo humano. Sin embargo, las limitaciones actuales en la resolución espacial de tarjetas gráficas y monitores dan lugar a posibles defectos al intentar reproducir la imagen original: si el número de píxeles (i.e., de muestras) dictado por el equipo es insuficiente para la imagen considerada, se producen errores durante su reconstrucción.

Muchas de las distintas soluciones adoptadas para resolver este problema en el mundo de la síntesis de imágenes han implicado aumentos significantes del coste computacional de los algoritmos de *rendering* (visualización fotorrealista de escenas 3D sintéticas), ya de por sí relativamente complejos en su mayoría. La técnica que se propone a continuación se basa en uno de los algoritmos más conocidos y sencillos de eliminación de superficies ocultas, el *z-buffer*, al que se añade un post-filtrado selectivo corrector de "dientes de sierra" que mejora sensiblemente la calidad de las imágenes.

1. Muestreo, *aliasing* y filtrado

Como explica cualquier texto introductorio al tratamiento de señal [7], de imagen [9], o incluso de gráficos por ordenador [5], que dedique un capítulo a la teoría de muestreo, para garantizar la posible reconstrucción posterior perfecta de una señal, ésta debe ser muestreada con una frecuencia mínima establecida por Nyquist en el doble de su ancho de banda. En otro caso, durante el proceso de reconstrucción se da un fenómeno conocido como *aliasing* o "solapamiento espectral", consistente en que las altas frecuencias de la señal original aparecen en la recuperada como variaciones de frecuencia mucho menor. En el caso particular de las imágenes, este fenómeno se manifiesta en la forma de los mencionados "dientes de sierra", efectos de Moiré y objetos "evanescentes", que desaparecen entre muestras demasiado escasas.

Para empeorar las cosas, esa frecuencia mínima de muestreo no basta por sí sola para poder recuperar perfectamente la imagen original: si durante la etapa de reconstrucción los pulsos rectangulares con los que se deben representar las muestras no se filtran adecuadamente, pueden aparecer en la imagen reconstruida altas frecuencias inexistentes en la resultante del muestreo. Este segundo fenómeno,

conocido como *rastering* en el campo de la imagen digital, se puede dar con cierta intensidad en los monitores del tipo LCD y en las impresoras. Sin embargo, no nos ocuparemos más de él, porque en los monitores del tipo CRT no suele tener tanta importancia como el primero, si el haz está bien enfocado.

Es claro, según el resultado de Nyquist, que si la imagen original tiene frecuencias superiores a medio ciclo por píxel, será imposible reconstruirla correctamente. Así pues, caben dos soluciones para eliminar o reducir el *aliasing*: aumentar el número de píxeles, o limitar el ancho de banda de la imagen, filtrándola antes de muestrearla. La primera solución tiene el paradójico inconveniente de ser "ideal", por ignorar la existencia del límite de resolución espacial de tarjetas gráficas y monitores; la segunda es a la que se suele recurrir una vez alcanzado ese límite, ya sea técnica o económicamente.

Quizá la necesidad de filtrar la imagen original antes de muestrearla no parezca muy intuitiva a quien no tenga una base de tratamiento de señal. Sin embargo, la idea que hay detrás de ello sí lo es: si se toman muestras puntuales de la imagen (lo que se conoce como *point-sampling*), y no se hace con la frecuencia suficiente como para que esas muestras sean representativas de las variaciones de la imagen, la interpolación que luego se pretenda hacer a partir de ellas será necesariamente pobre. Una mejora obvia, cuando no se puede aumentar el número de muestras, es tratar de que cada una de ellas contenga más información sobre la imagen original que la que ofrece un punto aislado de ésta. Por ejemplo, haciendo que el valor de cada muestra sea una media de los valores de la imagen en un entorno pequeño, pero de área no nula, del punto correspondiente (*unweighted area-sampling*). Incluso, refinando un poco más esta mejora, la contribución de los valores de los puntos del entorno a la media puede ponderarse según la proximidad al punto central elegido como representante (*weighted area-sampling*). Pues bien, el tomar la media (el hallar la integral, en definitiva) de una función, aun localmente, es una operación de filtrado "paso-bajo", que es aquélla que deja pasar las frecuencias bajas de la señal original, reduciendo sus componentes de alta frecuencia y por tanto su ancho de banda.

Para calcular esa media, cuando no se dispone de una descripción analítica de la imagen original es necesario utilizar una técnica conocida como "supermuestreo", consistente en tomar más de una muestra (previa) para promediar sus valores y así determinar el de un único píxel (muestra final). Este supermuestreo puede realizarse de forma regular, situando las muestras previas sobre los nodos de una rejilla, o estocástica, distribuyéndolas irregularmente de acuerdo con la función densidad de probabilidad de una variable aleatoria dada (de Poisson, de *jitter*, etc.). Por otra parte, el supermuestreo puede ser exhaustivo, si afecta a todas las muestras finales, o adaptativo, si sólo se realiza en parte de ellas (lógicamente, allí donde se pueden producir más problemas debidos al *aliasing*).

2. Partes vistas y ocultas en imágenes sintéticas

Una inmejorable introducción a los distintos métodos existentes para eliminar superficies ocultas al proyectar escenas tridimensionales y a las imágenes sintéticas en general puede encontrarse en [5]. Los más antiguos son probablemente los del tipo *scan-line*, que fueron desarrollados a finales de los años '60, y que recorren, para cada línea de la imagen, una lista de los polígonos que componen la escena, determinando cuáles son visibles tras la proyección considerada. También de aquella época es el algoritmo de subdivisión de áreas de Warnock, que particiona el plano de proyección, subdividiéndolo recurrentemente hasta alcanzar el nivel en que la decisión de qué polígonos ocultan a cuáles es fácil de tomar. A principios de los '70, se desarrollaron diversos algoritmos clasificados como *depth-priority* que concentraban sus esfuerzos en ordenar los polígonos de manera que al pintarlos en ese orden quedara garantizada la correcta resolución de partes vistas y ocultas.

Pero quizá el algoritmo más conocido y sencillo de aplicar, y seguramente por ello el más utilizado, es el *z-buffer*, que Catmull publicó en 1974. Consiste en ir almacenando la profundidad (denotada habitualmente por la coordenada z) de cada punto 3D que se proyecte sobre un píxel de la imagen, para poder saber, cuando otro punto 3D compita por el mismo píxel, cuál de los dos ocluye al otro. De esta manera, no se necesita ordenar previamente los polígonos, que pueden ser pintados en cualquier orden, a diferencia de lo que ocurre con los de tipo *depth-priority*, y tampoco se necesita disponer de una lista de todos ellos, como ocurre en el caso de los de tipo *scan-line*. Una variante también popular del *z-buffer* que reduce sensiblemente los efectos del *aliasing* y que permite tratar correctamente objetos translúcidos es el *a-buffer* [1].

En una clase aparte están otros algoritmos más sofisticados, como los de *ray-tracing*, cuyos primeros ejemplos datan de finales de los '60, y que determinan la visibilidad de las superficies trazando rayos imaginarios desde la posición de la cámara hasta cada uno de los objetos de la escena. Permiten sin duda mayor lujo y verosimilitud en las imágenes obtenidas, sobre todo desde la introducción de reflexiones y refracciones por Whitted y Kay a finales de los '70, pero también son sin duda los más caros computacionalmente.

En [3], [4] y [1] se pasa revista brevemente a las primeras técnicas mencionadas desde el punto de vista de la posible incorporación de métodos de *anti-aliasing*. En particular, Crow hace notar cómo el algoritmo original del *z-buffer* es incapaz por sí sólo de evitar el problema; y Carpenter, que la variante consistente en aplicar el *z-buffer* "línea a línea" sólo logra eliminarlo de las aristas casi verticales, pero no de las casi horizontales.

3. Motivación y circunstancias del presente trabajo

La técnica de *anti-aliasing* presentada a continuación está integrada en un sistema de rotulación 3D desarrollado en principio para ampliar la funcionalidad de una tituladora 2D para post-producción en televisión. En una primera fase, se escribieron módulos para la construcción del texto 3D por extrusión del 2D y para la visualización del modelo obtenido. Para esto último, se eligió por sencillez y velocidad (de desarrollo y de cálculo) el algoritmo del *z-buffer*, tal y como fue propuesto originalmente. Por esta razón, cuando hubo que abordar la solución al *aliasing*, inaceptable en señales que habían de ser emitidas y que debían ceñirse a las recomendaciones del CCIR (ahora UIT-R) sobre ancho de banda, se analizaron varias posibilidades.

La más obvia y fácil era el filtrado basado en un supermuestreo exhaustivo previo. El problema de esta solución, que además podría ser defendible como la más correcta desde un punto de vista matemático, es que es del todo inaceptable desde un punto de vista ingenieril: si el *z-buffer* tiene algún defecto, es el de necesitar mucha memoria para almacenar, además de los valores de los píxeles, los de las profundidades asociadas, de manera que generar $n \times m$ muestras más (suponiendo que se supermuestree n veces en una dirección y m en la otra) implica almacenar también $n \times m$ valores de z más. Teniendo en cuenta que una regla empírica basada en pruebas subjetivas establece que lo mínimo aceptable suele ser tomar $n = m = 3$, y preferiblemente $n = m = 4$, el supermuestreo exhaustivo hubiera obligado a aumentar en un orden de magnitud una cantidad de memoria ya grande de por sí. Por otra parte, la etapa de filtrado posterior al cálculo de la imagen previa, aun siendo trivial de programar, habría sido seguramente demasiado lenta en ejecución.

Otra posibilidad, nada despreciable, era utilizar el *a-buffer* propuesto por Carpenter, que además permite tratar superficies semi-transparentes. Sin embargo, esto hubiera exigido reenfocar el módulo de modelado para que entregara una lista completa de todos los polígonos al de visualización, en lugar de pasárselos de uno en uno, incurriendo quizá en problemas por limitaciones de memoria. Y una vez más, posiblemente hubiera hecho aumentar el tiempo de cálculo de las imágenes más de lo aceptable.

Y otra más, era idear un método de filtrado que no se tuviera que aplicar sobre toda la imagen indiscriminadamente, y que no necesitase de toda la base de datos del modelo 3D durante la etapa de *rendering*, que seguiría utilizando el *z-buffer* tradicional.

4. Procedimiento

La técnica propuesta aquí no trata de mejorar el algoritmo de Catmull, sino de aprovechar la información disponible sobre la escena al generar la imagen para

eliminar el *aliasing* **selectivamente** allí donde más se haya producido, **después** de que se haya producido. En concreto, se concentraron todos los esfuerzos en filtrar los "dientes de sierra" en las fronteras de los polígonos, que es donde hay variaciones abruptas del color, puesto que sus interiores ya quedaban adecuadamente pulidos por los métodos de interpolación de Gouraud o Phong.

4.1. Algoritmo base

El algoritmo de partida podría expresarse como sigue:

1a) proyectar los polígonos presentes en la escena uno tras otro, en el orden arbitrario en que sean entregados por el módulo de modelado, mediante el algoritmo del *z-buffer* y tomando una sola muestra por píxel (i.e., con *aliasing*), pero

1b) ir guardando en una lista las aristas de los polígonos, expresando las coordenadas de sus vértices en el referencial del dispositivo;

2) al recibir la orden de "no más polígonos", recorrer cada una de las aristas, reemplazando el color de cada uno de sus píxeles por una mezcla de los colores de los polígonos que comparten la arista.

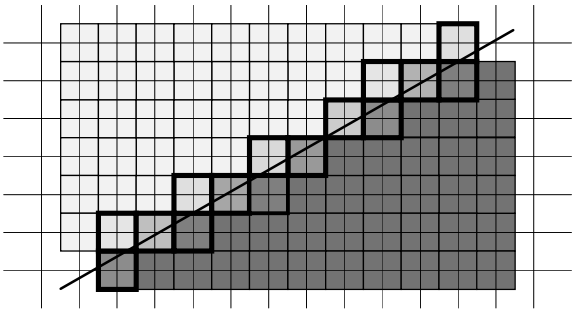

Figura 1. Mezcla de colores en los píxeles fronterizos.

Conviene explicar mejor por qué y cómo se debe realizar esta última "mezcla": al rellenar las zonas de la imagen donde se proyecta un polígono, hay píxeles claramente interiores al polígono que quedan enteramente cubiertos por él y otros, los más próximos a su frontera, que sólo lo están parcialmente. De éstos, lo razonable es considerar que pertenecen al polígono aquéllos que éste cubre en más del 50%, pero no deja de ser erróneo colorearlos enteramente de la misma manera que a los interiores. Lo correcto es que contribuyan al color de esos píxeles los polígonos que los comparten, y según la proporción en que los cubren, como muestra la Figura 1.

En cuanto a qué píxeles, de entre los vecinos de uno fronterizo, deben contribuir a la mezcla de colores, se hicieron pruebas hasta llegar a la conclusión empírica de que parecía suficiente tomar los dos que quedaban separados por la "dirección principal" de la arista. Es decir, los del Norte y el Sur para aristas de pendientes menores a 45° en valor absoluto, y los del Este y el Oeste para las demás.

Cabe hacer notar que, aun siendo posible saber durante el relleno de cada polígono qué píxeles quedan completamente cubiertos por él y cuáles sólo mayoritariamente, como se verá más adelante, esta mezcla no se debe hacer hasta haber acabado de pintar todos los polígonos: antes es imposible saber qué polígonos competirán por los píxeles fronterizos, y por lo tanto, qué colores hay que mezclar.

También es preciso concretar el sentido del "recorrido" de las aristas: se deben seguir líneas de píxeles 4-conexos al hacerlo. Es decir, que de un píxel se puede pasar a su vecino del Norte, del Sur, del Este o del Oeste, pero no a los del NO, NE, SO, ni SE (salvo en el caso patológico, mostrado en la Figura 1, de que la línea real pase verdaderamente de un píxel a otro por sus esquinas). Esto se puede hacer muy eficientemente con un algoritmo de tipo Bresenham que además calcule, también de manera incremental, el área cubierta en cada píxel por los polígonos vecinos [8].

Finalmente, es esencial aclarar que la operación de mezcla sólo debe llevarse a cabo, en cada píxel generado al recorrer una arista, si la z calculada para ese píxel, por interpolación entre las de los extremos de la arista, es menor que la que quedó almacenada en el elemento correspondiente del *z-buffer* (se siguió la convención de asignar $z = 0$ a los puntos más próximos a la cámara, y $z = z_{máx}$ a los más lejanos). De no tener esta precaución, podrían mezclarse los colores de píxeles más cercanos que los de un polígono oculto tras ellos, al recorrer una arista de ese polígono.

4.2. Algoritmo completo

4.2.1. Ajuste de z *en los extremos de los* spans

Cuando se rellenan los *spans* (tramos de píxeles contiguos de una misma línea) correspondientes a un polígono, hay que ir generando incrementalmente, además de los colores de los píxeles, las profundidades que deben ser almacenadas en el *z-buffer*. Desgraciadamente, vuelve a aparecer en ese momento el problema ya mencionado de que un píxel no corresponde a un punto matemático de área nula: incluso haciendo coincidir los nodos de la rejilla de coordenadas x e y enteras con los centros de los píxeles (como en la Figura 1), se comete un error al suponer que todos los puntos 3D del polígono que se proyectan sobre el área no nula cubierta por un píxel dado tienen la misma z.

Este error es especialmente grave cuando se comete en píxeles fronterizos, que ni siquiera pertenecen por entero a ningún polígono. Pero es que además, y precisamente porque los píxeles fronterizos no pertenecen estrictamente a ningún polígono, al recorrer las aristas se salta necesariamente en ocasiones de un polígono a otro, generando un conjunto de píxeles cuya z tendría que satisfacer, para ciertos

píxeles (x, y), la ecuación de un polígono, «$a_1x + b_1y + c_1z + d_1 = 0$», y para ciertos otros la de otro, «$a_2x + b_2y + c_2z + d_2 = 0$». Estas ecuaciones no están disponibles si se guardan solamente las coordenadas de los vértices de las aristas, y de hecho no pueden estarlo a no ser que se almacene explícitamente información sobre qué polígonos comparten qué aristas.

Por eso se optó, durante la etapa de rellenado, por asignar a esos píxeles fronterizos la misma z que sería generada para ellos durante la etapa de filtrado por interpolación entre las de los extremos de la arista correspondiente. Así, para eliminar el problema bastaba con establecer que, en caso de que dos puntos 3D compitiendo por un mismo píxel tuvieran idéntica profundidad, sería el último en llegar el que determinaría el color del píxel.

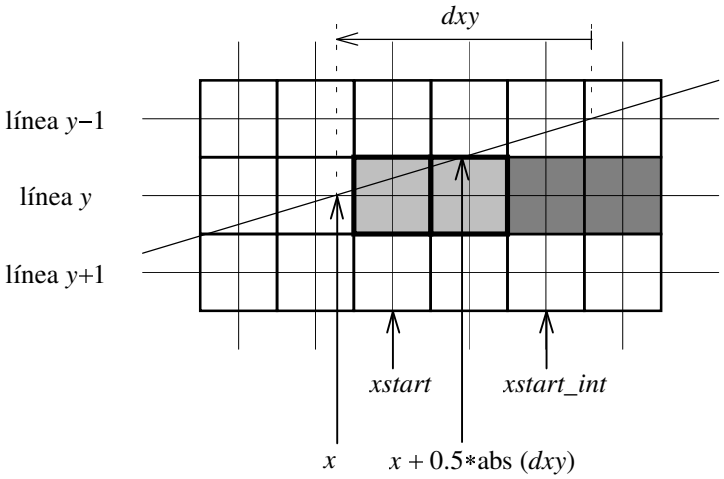

Figura 2. Píxeles afectados por el ajuste de *z*.

Queda por explicar, sin embargo, cómo determinar a qué píxeles, de los primeros y últimos de cada *span*, se debe aplicar el ajuste de z. Al trabajar con polígonos no convexos, e incluso en ocasiones no conexos (por ejemplo, el carácter 2D "8" es un polígono inconexo formado por tres sub-polígonos conexos), era natural utilizar un algoritmo de relleno de *spans* que mantuviera, para cada línea, una AET (*Active Edge Table*: tabla de aristas activas) [5]. En esta tabla, cada entrada tiene la información referente a una arista, actualizada para la línea horizontal en curso.

En particular, cuando se está barriendo la línea y (cf. Figura 2), dos cantidades almacenadas en la entrada de la AET correspondiente a una arista dada son la x del punto de corte de la arista con la línea y y la inversa de la pendiente de la recta que contiene a la arista, a la que se ha llamado *dxy* (el incremento que debe sufrir x al pasar de la línea y a la y+1). Si se conviene en llamar *xstart* (*xstart_int*) a la abscisa entera del píxel a partir del cual se entra mayoritariamente (completamente) dentro del polígono que queda a la derecha de una arista, y se utiliza la notación estándar

del lenguaje “C” para el redondeo de una cantidad de coma flotante, como *x* y *dxy*, al entero inferior (función “floor”) y superior (función “ceil”), es sencillo ver que:

- *xstart* = ceil (*x*)
- *xstart_int* = round (*x* + 0.5∗abs (*dxy*)) + 1 /* round (*x*) = floor (*x* + 0.5) */

Análogamente, se deben calcular las abscisas de los píxeles hasta los cuales se está completamente o mayoritariamente dentro del polígono que queda a la izquierda de una arista:

- *xstop_int* = round (*x* − 0.5∗abs (*dxy*)) − 1 /* round (*x*) = floor (*x* + 0.5) */
- *xstop* = floor (*x*)

Después, se puede proceder a rellenar el correspondiente *span*, sabiendo que todos los píxeles desde *xstart* hasta *xstop* (ambos inclusive) deben tener el color del polígono, pero que sólo desde *xstart_int* hasta *xstop_int* (ambos inclusive) deben tener la *z* correspondiente al plano del polígono.

4.2.2. Marcado de los píxeles ya mezclados y de sus vecinos

Hay zonas de la imagen donde las aristas abundan más que en otras, por ser los polígonos de menor tamaño: en el caso de los caracteres 3D, por ejemplo, las “tapas” dan lugar a facetas grandes, pero los “lomos” creados por la extrusión generan a veces polígonos muy pequeños. El exceso de filtrado en zonas de la imagen muy ricas en aristas, no es sólo razón de pérdida de eficiencia, sino incluso de error: dado que el mecanismo propuesto reemplaza los colores de los píxeles por las mezclas calculadas, si se tolera un efecto acumulativo se producen emborronamientos en la imagen.

Por este motivo, se decidió marcar a los píxeles ya mezclados, así como a sus vecinos contribuyentes a la mezcla, de manera que no fueran susceptibles de mezclas posteriores. Esta operación de marcado se llevó a cabo anulando la *z* de los elementos correspondientes en el *z-buffer* (i.e., acercándolos completamente a la cámara), aunque también se hubiera podido crear un *bitmap* que recogiese los píxeles marcados, para no desvirtuar el significado original del *z-buffer*.

La introducción del mecanismo de marcado de píxeles hizo más patente un problema que ya se daba de manera camuflada: el resultado dependía del orden en que se procesaban las aristas almacenadas. Hasta entonces, las aristas que constituían un polígono se insertaban de manera ordenada en una lista, pero sólo porque el hacerlo así facilitaba el descarte de las repetidas (cada arista pertenece a dos polígonos, al menos en superficies cerradas), que no interesaba recorrer dos veces. Sin embargo, se descubrió que el orden dentro de esa lista era esencial para la corrección del resultado, por lo cual se introdujo, en modelador y visualizador, el concepto de “prioridad” de una arista.

4.2.3. Asignación de prioridades a las aristas

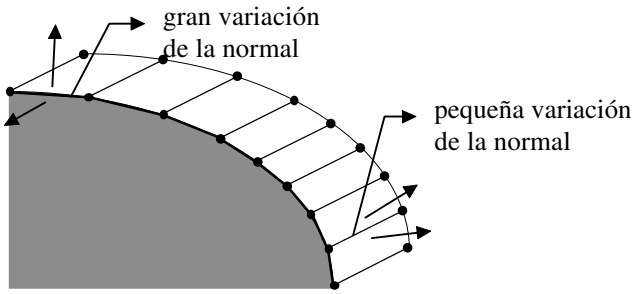

Figura 3. Aristas prioritarias y secundarias.

La Figura 3 ilustra la importancia relativa que la operación de filtrado propuesta tiene en distintas clases de aristas para reducir el *aliasing*: las transiciones de color entre dos facetas contiguas son potencialmente tanto más bruscas cuanto mayor sea el ángulo que forman sus normales. Por lo tanto, las aristas compartidas por dos polígonos casi coplanarios deben ser consideradas secundarias, a efectos del filtrado, frente a las que separan a dos facetas ortogonales (en el caso de los caracteres, una de las tapas y cualquiera de las facetas del costado).

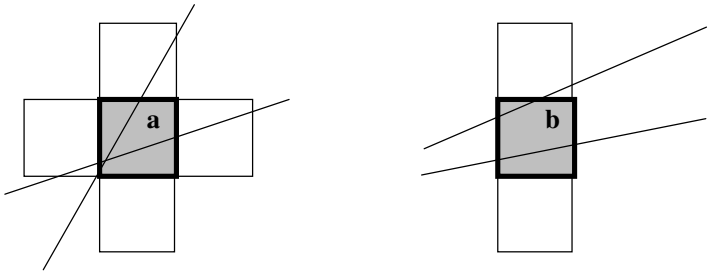

Figura 4. Píxeles atravesados por varias aristas.

El problema mencionado antes se daba en píxeles atravesados por dos o más aristas y en los que, dependiendo del orden en que fueran procesadas, se obtenía una mezcla u otra. La Figura 4 refleja dos situaciones posibles: en el píxel "a" se mezclaría el color de vecinos diferentes (aristas con diferente "dirección principal"); en el "b", serían los mismos vecinos los que intervendrían en la mezcla en los dos casos, pero con distinta ponderación (aristas con la misma "dirección principal", pero dividiendo en distinta proporción al píxel). Al ordenar las aristas, gracias a la asignación de prioridades, y al garantizar que los píxeles sólo serían retocados una vez, se suprimió esta aleatoriedad del resultado.

En un primer momento se pensó en realizar una clasificación binaria de las aristas, marcándolas con un único *bit*, lo cual hubiera sido seguramente suficiente

para tratar objetos como los caracteres 3D. Pero finalmente se codificó esa prioridad con un octeto, en función del coseno del ángulo formado por las normales de los polígonos separados (si éstos forman un ángulo mayor o igual a 90°, se asocia a la arista en cuestión un valor nulo, y si el ángulo es nulo, un valor máximo). Ese valor debe ser suministrado por el módulo de modelado para que el de visualización ordene las aristas en la lista que va confeccionando al rellenar los polígonos, y luego las procese empezando por las de menor valor.

4.2.4. Resumen

El algoritmo completo, incluyendo los refinamientos del de partida, podría expresarse así:

1a) proyectar los polígonos entregados uno a uno por el modelador mediante el algoritmo del *z-buffer*, ajustando la *z* correspondiente a los píxeles de los extremos de los *spans*;

1b) guardar en una lista las aristas de cada (sub-)polígono, ordenándolas por valores de prioridad crecientes (de cada arista se deben conservar sólo su prioridad y las coordenadas de sus vértices, expresadas en el referencial del dispositivo);

2) una vez proyectado el último polígono de la escena, recorrer las aristas en orden de prioridad creciente, sustituyendo el color de los píxeles afectados por una mezcla de los de sus vecinos y marcándolos (tanto a ellos como a sus vecinos) para que no se vuelvan a modificar.

4.2.5. Resultados

Texto	Filtrado	Color constante	Gouraud	Phong
F	NO	1,4	1,5	6,7
(71 polígonos)	SI	1,7 (+21%)	1,8 (+20%)	7,0 (+4%)
CEIG'95	NO	1,6	1,9	4,4
(541 polígonos)	SI	2,4 (+50%)	2,7 (+42%)	5,3 (+20%)

Tabla 1. Tiempos de cálculo de "F" y "CEIG'95" (segundos).

En las imágenes anejas, producidas con la versión de MS-Windows de "3DTXT", se puede apreciar el efecto del filtrado, que reduce sensiblemente el *aliasing* causante de los "dientes de sierra", y la Tabla 1 da los tiempos de cálculo invertidos por el programa en visualizar dos textos sencillos. A pesar de la poca precisión que dan las medidas de tiempo ofrecidas por el cronómetro de MS-Windows, se puede observar que el tiempo de cálculo exigido por el proceso de filtrado es independien-

te del método empleado para interpolar el color (como cabía esperar), y que hay una gran variación del incremento relativo del tiempo de visualización entre unas escenas y otras.

Esto último es también lógico, puesto que ese incremento relativo es función del cociente entre el número de píxeles pintados durante el relleno de los polígonos y el número de píxeles mezclados en el filtrado posterior: cuando hay muchos polígonos pequeños, una gran parte de los píxeles pertenecen a una u otra arista, y deben ser modificados durante la etapa de mezcla de colores. Por eso el programa obtiene los mejores resultados cuando hay unos pocos cientos de polígonos de tamaño medio o grande en la escena, que suele ser precisamente el caso de los rótulos 3D.

5. Conclusiones

La técnica propuesta para reducir los efectos debidos al *aliasing* de bordes en imágenes sintéticas generadas con *z-buffer* requiere sólo un pequeño incremento de la memoria que necesita intrínsecamente el algoritmo original. Por otra parte, se puede afirmar que el tiempo empleado en el post-proceso es muy inferior al ya requerido para proyectar y rellenar los polígonos con cualquiera de los métodos de interpolación del color (Phong o Gouraud), o incluso sin ella (color constante: *flat shading*).

Aunque el módulo de visualización se desarrolló originalmente en conjunto con el de modelado del texto 3D, son independientes entre sí, por lo que lo único que requiere el visualizador es que le sea entregada, con cada polígono, una lista de prioridades de sus aristas para filtrarlas luego en el orden correcto. Actualmente se estudia cómo filtrar los bordes generados por intersecciones de polígonos sin que el modelador tenga que incluir una lista explícita de las aristas que aparecen implícitamente por este motivo.

6. Bibliografía

1. L. Carpenter: *The A-buffer, an Antialiased Hidden Surface Method*, ACM Computer Graphics, **18**-3 (proc. of SIGGRAPH'84), 1984, pp. 103-108.

2. E. Catmull: *A Hidden-Surface Algorithm with Anti-Aliasing*, ACM Computer Graphics, **12**-3 (proc. of SIGGRAPH'78), 1978, pp. 6-11.

3. F. Crow: *The Aliasing Problem in Computer Generated Shaded Images*, Communications of the ACM, **20**-11, 1977, pp. 799-805.

4. F. Crow: *A Comparison of Antialiasing Techniques*,
IEEE Computer Graphics and Applications, **1**-1, 1981, pp. 40-48.

5. J. Foley, *et al.*: *Computer Graphics, Principles and Practice* (2nd. ed.),
Addison-Wesley, 1990, pp. 617-646, 693-695.

6. N. Max: *Antialiasing Scan-Line Data*,
IEEE Computer Graphics and Applications, **10**-1, 1990, pp. 18-30.

7. A. Oppenheim, A. Willsky, Y. Young: *Signals and Systems*,
Prentice Hall, 1983, pp. 527-531, 544-545.

8. M. Pitteway, D. Watkinson: *Bresenham's Algorithm with Grey Scale*,
Communications of the ACM, **23**-11, 1980, pp. 625-626.

9. G. Wolberg: *Digital Image Warping*,
IEEE Computer Society Press, 1990, pp. 8-9, 106-111, 163-186.

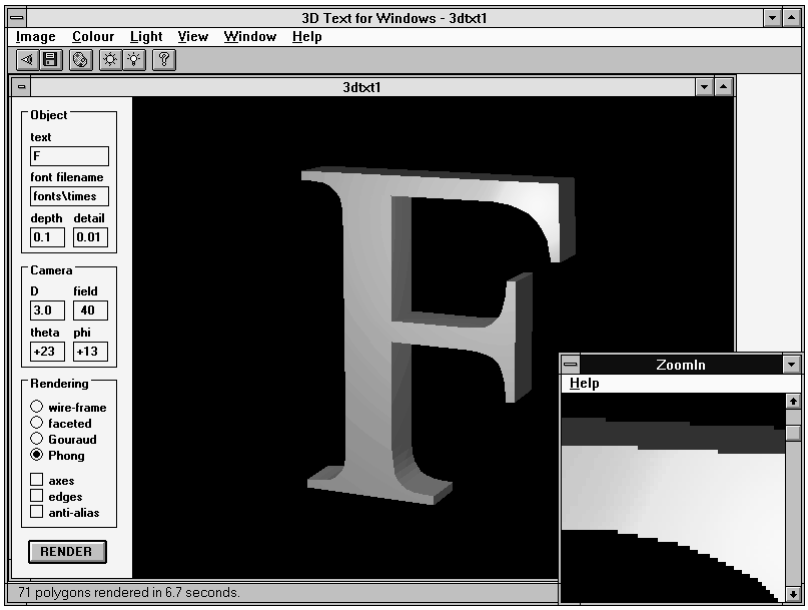

Imagen 1. Texto con *aliasing* (antes del filtrado).

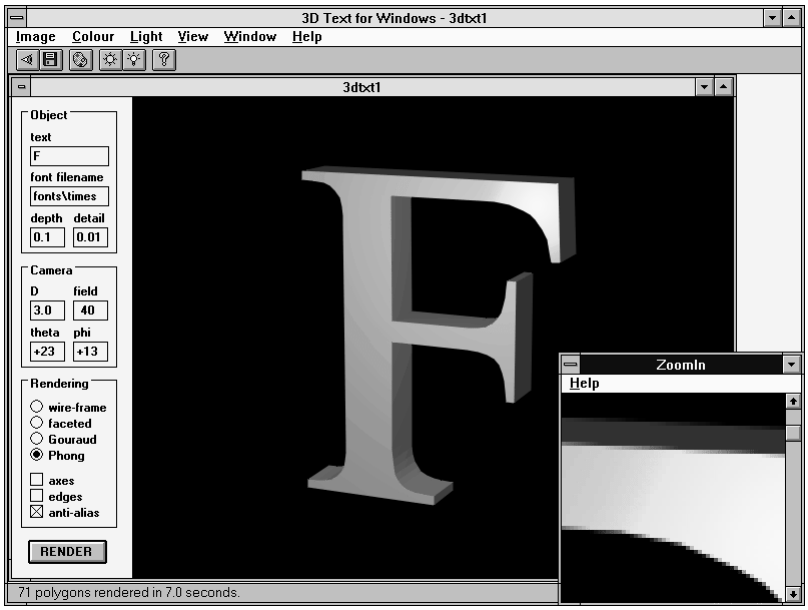

Imagen 2. Texto con menos *aliasing* (después del filtrado).