SUPLEMENTO MENSUAL DEDICADO A LA IMAGEN SINTÉTICA

RENDER manía

Número 2

BIBLIOTECA 3D

Cómo utilizar objetos SOR

TALLER VIRTUAL

POV vs Polyray Diferencias entre dos imprescindibles

EN EL CD-ROM

POV para Windows

CÓMO...

Claves para crear los mejores POV-Efectos



LA PORTADA

HOMENAJE a un constructor de mundos imposibles

pcmanía



Todo los ficheros de inclusión y los ejemplos podéis encontrarlos en el directorio PCMANIA\RENDER52 del CdRom

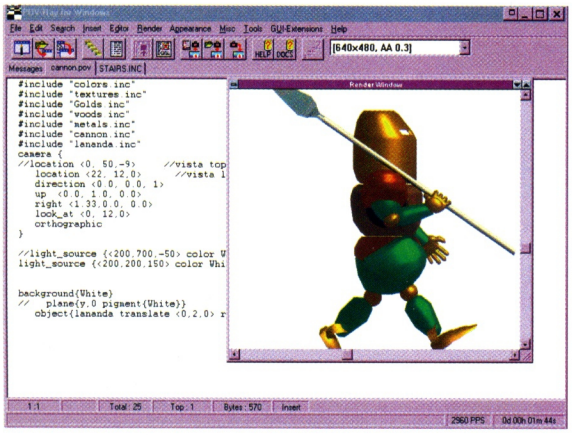
POV para Windows

Los povmaniacos están de enhorabuena. Hoy incluimos en el CD la versión 3.0 de POV para Windows. Esta versión funciona bajo Windows 3.1, 3.11, 95 y NT. Povwin, como llamaremos abreviadamente al programa, incorpora un editor interno, una potente ayuda, una autodemo y, sobre todo, todas las últimas características del raytracer más popular del planeta.





En las imágenes, distintas pruebas realizadas con Povwin.



En caso de que la versión disponible de Windows no sea la 95 o NT, necesitaremos Win32s, una utilidad que permite ejecutar aplicaciones de 32 bits sobre las versiones de Windows que funcionan a 16 bits. De todas maneras, los autores advierten que pueden darse fallos de funcionamiento en estas versiones. El funcionamiento de Povwin, en todo caso, debería ser impecable bajo Windows 95 y NT.

Los requerimientos de memoria para Povwin son superiores a los de su versión hermana para Dos. Mientras que esta última se conforma con 4 Megabytes de Ram y recomienda 8, Povwin necesita 8 Megabytes como mínimo y recomienda el doble. Esta exigencia de memoria son la única desventaja de Povwin con respecto a la versión Dos de POV, que por otro lado, puede traducirse en un tiempo de render superior para los ficheros que demanden mucha memoria, ya que POV tendrá que emplear memoria virtual de disco en cuanto se quede sin Ram.

Instalación

Veamos un ejemplo de instalación bajo Windows 95. Tras ejecutar el fichero EXE, el instalador nos avisará de que se procederá a instalar POV en cuanto pinchemos OK. Hecho esto aparecerá el documento legal de distribución de POV. El programa nos indicará que sólo deberemos seguir con la instalación si aceptamos todos los términos descritos en dicho documento. Si éste es el caso, el instalador nos pedirá que indiquemos el directorio donde irá POV. También aparecerá un mensaje señalándose la posibilidad de



crear backups con los ficheros (de versiones anteriores) reemplazados durante la instalación y, si deseamos hacerlo así, se pedirá un directorio para los backups.

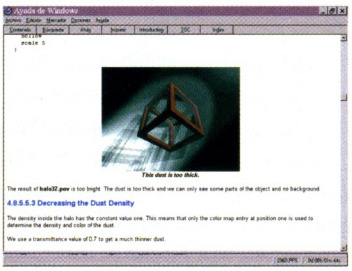
Hecho todo esto comenzará la instalación propiamente dicha y, al concluir ésta, se nos preguntará dónde queremos situar el icono de acceso. Seguidamente una nueva ventana nos preguntará si queremos asociar un fichero .ini a Povwin. (Los ficheros .ini contienen parámetros por defecto para los comandos de línea de órdenes de POV. En nuestro caso la constestación fue negativa). En el siguiente paso el programa nos avisará de que existen dos posibles opciones para los menús de Povwin; el novato y el avanzado. En nuestro caso se optó por el modo avanzado pero el usuario puede escoger el otro modo y cambiarlo en el menú "appearance", desde el mismo Povwin.

Finalmente el instalador nos pedirá un directorio donde colocar las imágenes generadas. Si no damos ningún nombre, Povwin utilizará como directorio de salida aquel donde se halle el fichero escénico dado como entrada. (Este es el caso en nuestra instalación). ¡Y esto es todo, pov-colegas! Después del correspondiente mensaje de instalación completada, aparecerá un texto con diversos detalles acerca de POV y se nos preguntará si queremos ver una demo, lo cual nos vendrá muy bien para testear si ha habido o no algún problema con la instalación.

El entorno de POVWIN

Povwin dispone de un editor interno y una ventana para los renders. Podremos lanzar un render y, al mismo tiempo, seguir trabajando en el editor y realizar cambios en el fichero cuya imagen se está generando. El render trabaja sobre la última grabación del fichero pov que está actualmente en edición, no sobre los cambios realizados después de la última grabación. (No recordar este pequeño detalle puede ocasionarnos algún contratiempo).

El código del editor no ha sido programado pensando en Windows 3.1 o 3.11. Esto no quiere decir que vayamos a tener dificultades con estas versiones, pero si intentamos desactivar el editor y usar otro externo, se presentarán problemas. (Para desactivar el editor basta



con pulsar sobre la opción de editor, en el menú "Appearances"). En cuanto a la ventana de render, podemos reescalarla, desplazarla, etc, como la ventana de Windows que es, incluso mientras el proceso de render se está efectuando. Una vez editado un fichero .pov podemos renderizarlo pinchando sobre la opción "Start rendering" del menú "Render". El render se efectuará teniendo en cuenta los parámetros por defecto, los cuales pueden ser alterados pinchando sobre la opción "Edit settings/render" del mismo menú. Al seleccionar esta opción aparecerá una ventana con la que podremos cambiar la resolución para el render, el fichero .ini en uso y otras cosas.

Trabajo en equipo

Las opciones del editor son las típicas; se puede trabajar con varios ficheros, copiar texto, etc. Aparte de esto, el editor dispone, en el menú "Insert" de opciones para insertar sentencias, operaciones, texturas y otras palabras del lenguaje escénico en el texto. Con ello podremos insertar un ejemplo de uso de la palabra elegida, lo cual será útil si hemos olvidado la sintaxis o queremos escribir algo rápidamente. Además, el editor incluirá unas líneas de comentarios donde explicará el uso de la palabra elegida.

Hay que recordar que el render debe ser invocado sólo si el fichero .pov apropiado está siendo editado. Si ordenamos el render mientras se está trabajando en un fichero .inc, Povwin dará errores. (A menos que se esté utilizando la opción "Select File and Render" del menú "Render"). El render podrá ser detenido en cualquier momento con la opción "Stop Rendering" del menú "Render".

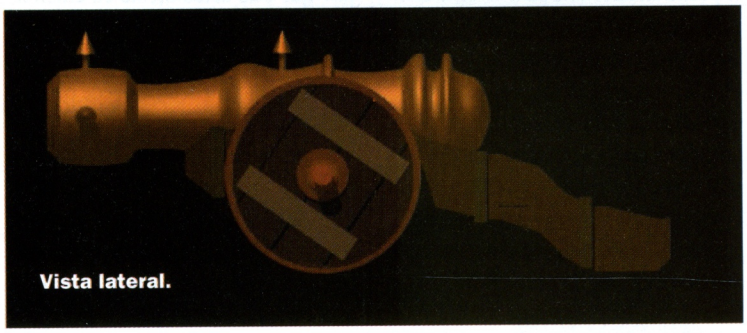
La ayuda

Uno de los apartados más interesantes es la ayuda a la que se accede desde la ventana "Help". Podremos consultar la documentación de POV (el doctorial), buscar un tema, una palabra, consultar el índice, etc. Hay dos detalles importantes a apuntar sobre la documentación que adjunta Povwin. En primer lugar no es idéntica a la de la versión de Dos. Ahora adjunta gráficos e imágenes que nos serán de gran ayuda destacando, por ejemplo las escenas incluidas para explicar los parámetros de atmósfera y niebla. Sin embargo (¡Ay!) la documentación no está completa. Hay algunos apartados que aún faltan por escribir.

Cañones que no matan

El modelo de cañón medieval cuyo proceso de creación vamos a estudiar hoy debe su existencia a una pequeña broma: José González -el autor del orco 3D que presentamos en Pcmanía 51- cometió el error de asegurar que, en feroz batalla, una horda de sus orcos barrería a los lanceros que el autor de estas líneas creó para el número anterior.





Vista lateral.

En opinión de este artista, sus orcos resultaban más feroces, grandes y fuertes que los lanceros, los cuales, tal vez por culpa de sus caricaturizadas dimensiones, parecían más predispuestos a amasar pan que a trabarse en dura lucha con sus tremendas criaturas verdes.

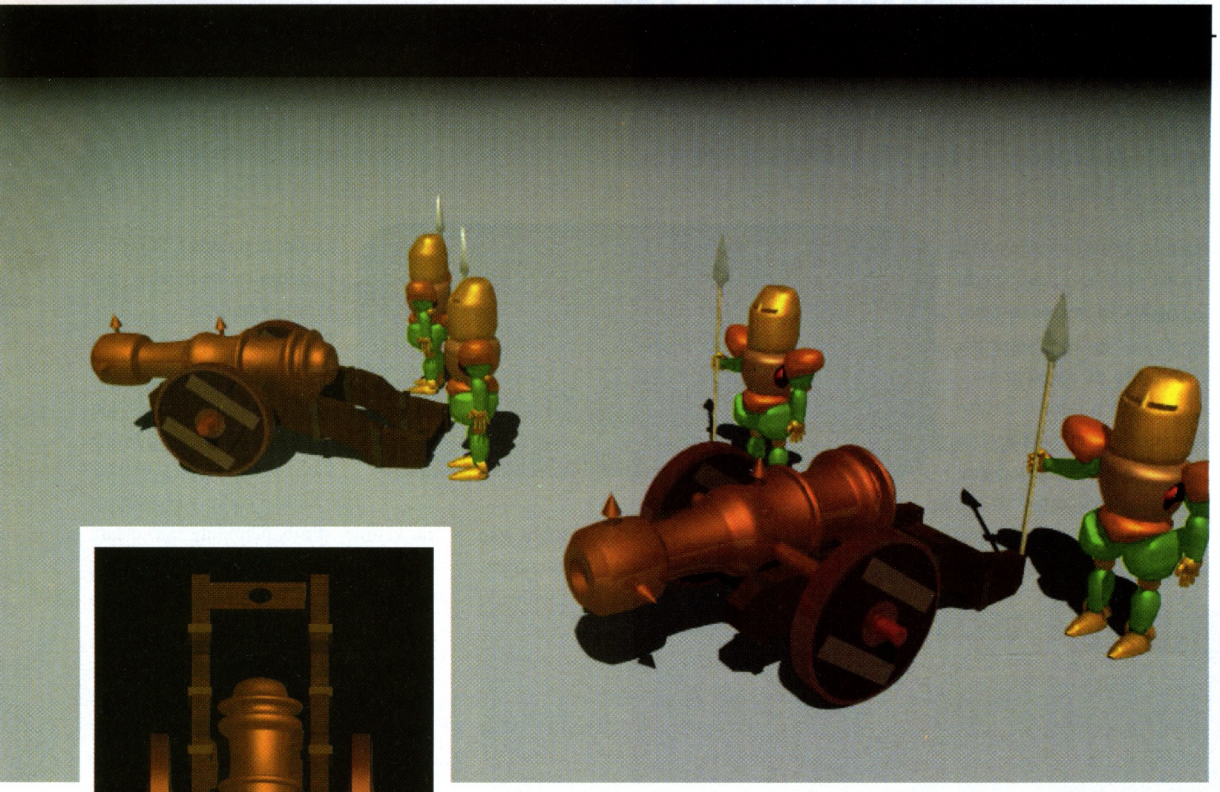
Molesto por este insulto al honor de sus lanceros virtuales, el autor juró venganza y prometió preparar una batalla donde sus personajes apalizaran a los malolientes y presumidos orcos. Sin embargo (¡Ay!) pronto quedó patente, al comenzar a idear escenas con ambos tipos de personajes, que José González tenía cierta razón: los orcos son verdaderos monstruos, es natural representarlos con, al menos, una cabeza más de altura que los lanceros. Además para que las escenas resultasen bien era preciso representar muchos orcos: ¿Cómo podía pues nivelarse la batalla?

La respuesta está en el cañón que hoy presentamos. El cañón ha sido creado con POV 3.0 y nos servirá (piques aparte) para estudiar una de las nuevas características de este ray tracer, las superficies de revolución.

La documentación

Antes de comenzar con los detalles de la construcción del cañón, conviene hacer un inciso sobre las fuentes de inspiración que han servido para crear este modelo. Como ya saben quienes son capaces de tragarse estas líneas mes tras mes, el primer paso antes de crear un objeto es tener una idea clara del modelo que se pretende realizar. Por esta razón suele ser buena idea intentar hacer unos dibujos en papel, en diferentes vistas, a fin de comprobar si la idea está lo suficientemente clara.

Pues bien, en este caso pronto quedó patente que éste no era el caso. Los dibujos realizados eran muy generales y carecían de detalles. Cuando sucede algo como esto lo mejor es buscar nueva documentación. En este caso la documentación escogida no se halló rebuscando entre libros de historia sino en los libros de ejércitos Warhammer, en concreto en el dedicado al Imperio.



Vista top del cañón.

Para quien no lo conozca, Warhammer es un juego de mesa centrado en un mundo medieval imaginario donde orcos, humanos, enanos y elfos guerrean entre si. En este juego -también existe una versión para PC- se emplean miniaturas de estos personajes que tienen un alto nivel de detalle. Pero, aparte del juego en si, hay publicados muchos libros que incluyen fotografías de estos personajes y de sus armas y utensilios. También se estudiaron otros cañones que vienen representados en "Codex Orkos", el suplemento de Warhammer 40000.

Por último hay que decir que nuestro cañón no es completamente idéntico a ninguno de los estudiados en estos libros (El gran cañón imperial, el traktor, el trikitrake, etc). La documentación sirvó sobre todo para tener las ideas más claras. O sea para comprobar cosas tales como las distintas formas de las ruedas que se emplean para estas piezas, la unión del cañón con la cureña, la forma de ésta, etc. Ni que decir tiene que los cañones de Warhammer son también imaginarios pero, a pesar de esto, se prefirió estudiar este tipo de documentación en vez de otra "más real" ya que estos cañones resultaban muy atractivos.

Superficies de revolución

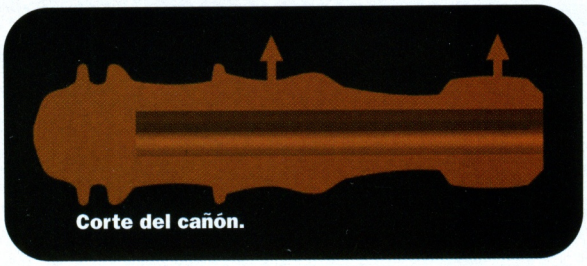
Como ya hemos explicado en alguna otra ocasión, se llama superficies de revolución a los objetos tridimensionales que obtenemos al rotar una forma bidimensional en torno a un eje dado. Un ejemplo típico de objeto que se presta muy bien a ser creado con este procedimiento es el peón de ajedrez; en este ejemplo podemos imaginar, o dibujar en una hoja, un peón visto de perfil alzándose sobre una línea que más tarde será el plano del suelo. Si dibujamos una línea vertical que cruce el centro del peón dividiéndolo en dos y borramos uno de los lados tendremos la forma bidimensional que hay que emplear. Así, para obtener nuestro peón tridimensional, habríamos de rotar esta forma 2D 360 grados en torno al eje Y (suponiendo que el suelo sea el plano X-Z, como siempre).

En su última versión POV incluye una nueva instrucción para crear superficies de revolución (realmente hay dos, pero hoy no vamos a hablar de lathe). Esta sentencia es Sor (de ahora en adelante llamaremos objetos Sor o simplemente Sor a las superficies de revolución) y su formato es…

```
sor {
   número_de_puntos,
  <punto0>, <punto1>, ..., <punto n-1>
   [ open ]
   [ sturm ]
 }
```

Los puntos del 0 a n-1 son valores para el plano X-Y donde vamos a di-

bujar la forma bidimensional (en el ejemplo, el medio perfil del peón). El primer parámetro de Sor es, precisamente, el número de estos puntos. En cuanto a open y sturm son palabras opcionales. Por defecto la forma tridimensional que obtendremos es sólida pero incluyendo open el objeto quedará sin cerrar. Sturm requiere una explicación algo más larga.


Corte del cañón.

En programas de modelado poligonal como Imagine, 3D Studio y otros, las superficies de revolución que se obtienen al rotar la forma 2D son mallas poligonales cuyo acabado puede suavizarse opcionalmente. En POV, sin embargo, Sor no crea ninguna malla. Según el manual de POV: "Los objetos Sor se generan rotando el gráfico de una función sobre un eje". Esto quiere decir que los puntos de la forma 2D a emplear no son vértices cuya rotación generará una malla, sino puntos de control sobre curvas que actúan de forma similar a los splines de lathe (otra nueva sentencia de POV). La forma final quedará determinada por la posición de estos puntos de control. En el manual de POV se incluye la fórmula que utiliza POV para generar estas curvas pero no la reproduciremos aquí. Para el usuario será suficiente con saber que la forma final 3D pasará por estos puntos de control y que, gracias al empleo de esta fórmula, el objeto final tendrá una apariencia suavizada.

Sin embargo, no todo es de color de rosa. La generación de un objeto Sor puede requerir una precisión de cálculo superior a la usada normalmente por los algoritmos de POV. Cuando esto ocurra veremos como partes del objeto quedan sin dibujar o como sobre éste aparecen muchos puntos mal sombreados. Esto puede subsanarse añadiendo la palabra sturm, la cual dice a POV que utilice otro algoritmo más preciso (y también más lento) para los cálculos.

Para utilizar Sor conviene recordar las siguientes cosas:

A) Los puntos definidos rotarán 360 grados en torno al eje Y. O sea que cuanto más lejos sean definidos de dicho eje, más ancho será el objeto 3D final.

B) Podemos realizar operaciones CSG sobre los objetos CSG a menos que se haya incluido la palabra "open" (en cuyo caso el objeto no será "sólido", naturalmente).

C) Al escribir los puntos para la forma 2D inicial del Sor hay que tener en cuenta que cada uno deberá tener una altura en Y superior a la del punto precedente.

D) Los valores de X para los puntos deberán estar en el lado positivo del eje.

Como la forma usada (el perfil del cañón visto desde arriba) se dibuja en el plano X-Y, el siguiente paso, una vez estemos satisfechos con el resultado 3D, es rotar el objeto de forma que quede paralelo al plano del suelo. Para ello efectuaremos las traslaciones y rotaciones correspondientes. Hecho esto podemos efectuar algunas operaciones adicionales para dar más realismo al cañón: el recorte CSG (para el ánima), y los objetos para el punto de mira.

Por último hay que indicar un detalle más: a pesar del uso de sturm la boca del cañón presentaba fallos de visualización -como si el interior fuese visible y el exterior no visible-. Probablemente esto estará relacionado con el hecho de que los puntos primero y último de la forma 2D se emplean únicamente para controlar la forma de las curvas inicial y final. O al menos eso dice la documentación. En fin, el caso es que después de numerosas pruebas el autor del cañón desistió de averiguar la forma en que afectan estos puntos a dichas curvas. Sin embargo, como esto sólo parece afectar a los extremos de la forma 3D, la cosa se solucionó agregando un par de objetos simples como union para formar la boca del cañón. (Es una chapuza, sí, pero funciona).

El cañón "Mamapupa"

El siguiente paso es la creación de la cureña (la pieza que sostiene al cañón). Para crearla se empleó una objeto extrude que quedó dividido en dos formas gracias a una operación de intersección. Este objeto fue creado usando la nueva sentencia Prism que POV 3.0 ha incluido para generar este tipo de objetos. Prism ya fue explicada con cierto detalle en el número 47 de Pcmanía por lo que no vamos a extendernos nuevamente sobre ella. (Recordemos que los objetos extrude se crean desplazando en un eje formas bidimensionales. Prism emplea splines para asegurar que los contornos de estos objetos quedan suavizados).



Únicamente cabe decir que, en la idea inicial, la cureña tenía una forma distinta en la que la figura resultaba mucho más compleja en su perfil top (vista desde arriba, una vez orientada). Para realizar esta pieza el plan hubiera consistido en crear una operación CSG entre dos objetos extrude; uno con el perfil lateral de la cureña y otro con el perfil superior. Esta técnica -que ya utilizamos en las almenas del castillo- permite crear objetos bastante complejos pero resulta un poco tediosa ya que estamos obligados a orientar dos objetos extrude (recordemos que Prism genera estos objetos a partir de una malla cerrada de puntos dispuesta en el plano X-Z). Los dos objetos sobre los que hay que operar suelen requerir orientaciones distintas puesto que corresponden a vistas distintas y ello puede resultar lioso. Al final, sin embargo, la cureña que se diseñó tiene una sencilla forma de rectángulo en su vista superior y por ello no resultó necesario utilizar este truco.

Pero volviendo a nuestra pieza, después de colocar unos refuerzos -simples boxs- a la cureña (a fin de cuentas debe sostener la pieza), lo siguiente fue añadir el eje del cañón y el taco que permite a éste apuntar a diversas alturas (la verdad es que el autor de la pieza desconoce cómo hacían esto los artilleros, pero…). Luego se crearon las ruedas con varias operaciones CSG empleando cilindros y cajas, se aplicaron varias texturas de metal y madera extraídas de las librerías de POV y nuestra flamante pieza quedó terminada. Pueden crearse muchas variaciones a partir de este diseño básico cambiando las ruedas, alterando el objeto Sor que forma el cañón, incluyendo adornos diversos, etc.

Las dimensiones

Como se supone que la pieza de artillería va a ser empleada por nuestros lanceros, el paso final consistió en escalarla de modo que las dimensiones de ambos modelos quedaran congruentes. Ello fue muy simple ya que la pieza había sido construida de modo que las ruedas tocaran el suelo en el plano X-Z situado en Y=0. Por esta razón no hubo que hacer operaciones adicionales de centrado en ningún eje, ya que el cañón fue construido de modo que su centro de gravedad coincide con el eje Y de coordenadas.

Normalmente al modelar un nuevo objeto con cualquier programa el autor de estas líneas no se preocupa nunca de las dimensiones sino sólo de la forma del objeto. Esto es algo casi obligado puesto que el diseño inicial pasa muchas veces por el empleo de hojas cuadriculadas. Por ello los valores obtenidos para los vértices garantizan que, casi con seguridad, las dimensiones entre distintos objetos de POV no van a coincidir nunca (deben caber en una sola hoja).

¡Más madera!

Cuando José González vio las escenas con cañónes que se estaban preparando para la inminente batalla prometida protestó arguyendo que existían diferencias en el armamento. Se marchó y algún tiempo después regresó con un montón de hojas llenas de catapultas y bombardas de todos los tamaños. Sin embargo la batalla ya había concluido (ya se había cerrado el número de Rendermanía que tenéis ahora en vuestras manos) y por ello el duelo queda aplazado para algún número posterior donde, eso sí, cambiaremos de armas y escenarios. Es una pena (es un decir, por supuesto).



Crear nebulosas estelares,

En el número anterior tuvimos ocasión de admirar una magnífica escena de Félix Higueras Jorna. Dicha escena -que reproducimos hoy en estas páginas- representaba un fondo estelar de estrellas y nebulosas con un acabado tan fantástico que al principio lo tomamos por una foto. Luego llegó la sorpresa, al comprobar que la imagen estaba hecha empleando otro de los nuevos efectos de POV 3.0, los halos.



El universo de Félix Higueras Jorna.

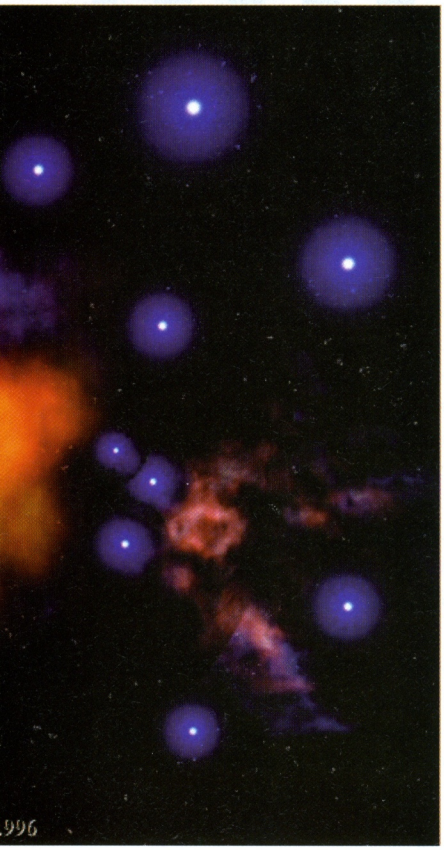
Según el manual de POV, los halos son un tipo de textura procedural que simula una nube de partículas. El nombre de la sentencia, halo, viene de la posibilidad que ofrece la misma de generar halos como los que tiene el sol u otras fuentes luminosas. Para controlar el acabado final del halo podremos manejar muchos parámetros diferentes con los que modificaremos la distribución de las partículas e indicaremos la manera en que éstas interactúan con la luz. De este modo podremos crear efectos de humo, fuego, explosiones, etc.

Dado que el halo funciona de manera similar a una textura procedural, lo primero que debemos hacer será asignarlo a un objeto (al que llamaremos objeto contenedor o contenedor a secas). La nube de partículas del halo llenará este objeto el cual, forzosamente, deberá estar hueco y tener la superficie translucida. Si una de estas dos condiciones no se cumple no veremos el halo.

La sintaxis completa de la sentencia halo es la que podemos ver en la figura 1. Pero... ¡que nadie se asuste! No es preciso utilizar todas estas palabras para crear buenos efectos. No es fácil lograr el efecto buscado pero podemos conseguir



explosiones y otros POV-Efectos



aproximaciones bastante aceptables.

Para comenzar observemos la figura 2 –la lista de instrucciones corresponde al ejemplo halo01 del manual–. Lo primero que advertiremos es que la descripción del halo se coloca de modo similar a otras palabras como pigment o normal, que describen propiedades. El objeto contenedor del halo es una esfera que cumple las propiedades descritas anteriormente gracias a las palabras rgbt y hollow.

FIGURA 1

"Formato completo del Halo"

```
halo {
attenuating | emitting | glowing | dust
[ constant | linear | cubic | poly ]
[ planar_mapping | spherical_mapping |
  cylindrical_mapping | box_mapping ]
[ dust_type DUST_TYPE ]
[ eccentricity ECCENTRICITY ]
[ max_value MAX_VALUE ]
[ exponent EXPONENT ]
[ samples SAMPLES ]
[ aa_level AA_LEVEL ]
[ aa_threshold AA_THRESHOLD ]
[ jitter JITTER ]
[ turbulence <TURBULENCE> ]
[ octaves OCTAVES ]
[ omega OMEGA ] [ lambda LAMBDA ]
[ colour_map COLOUR_MAP ]
[ frequency FREQUENCY ]
[ phase PHASE ]
[ scale <VECTOR> ]
[ rotate <VECTOR> ]
[ translate <VECTOR> ]
 }
```

La palabra rgbt sirve para indicar a POV los valores de rojo, verde, azul y transmisión que va a tener el objeto –en inglés transmittance–. La transmisión es un tipo de transparencia de POV por la que la luz que atraviesa al objeto no queda afectada por el color propio de este (como sucede con filter -rgbf-, donde lo que vemos a través del objeto transparente tiende a tomar la pigmentación del mencionado objeto). En el ejemplo el valor es 1, o sea transparencia total.

En cuanto a hollow, POV asume por defecto que los objetos son sólidos pero la inclusión de esta palabra cambia este estado de cosas. Con hollow los objetos quedarán huecos y su superficie se

FIGURA 2

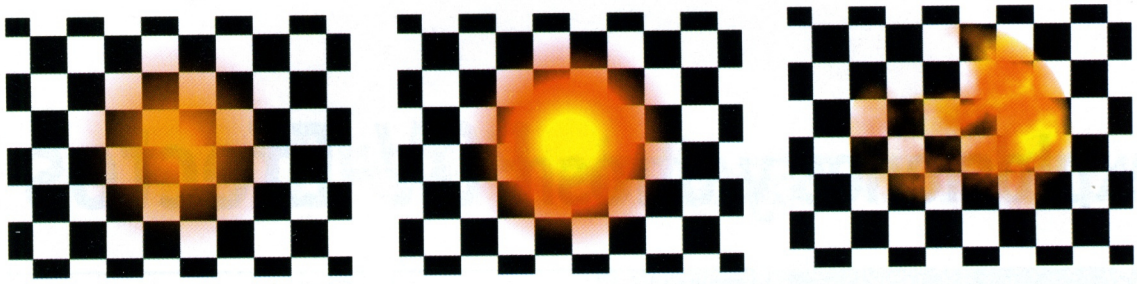
```
camera {
  location <0, 0, -2.5>
  look_at <0, 0, 0>
}
light_source { <10, 10, -10> color rgb 1 shadowless }
plane { z, 2
  pigment { checker color rgb 0, color rgb 1 }
  finish { ambient 1 diffuse 0 }
  scale 0.5
  hollow
}
sphere { 0, 1
  pigment { color rgbt <1, 1, 1, 1> }
  halo {
    emitting
    spherical_mapping
    linear
    color_map {
      [ 0 color rgbt <1, 0, 0, 1> ]
      [ 1 color rgbt <1, 1, 0, 0> ]
    }
    samples 10
  }
  hollow
}
```

hará infinitamente delgada. Pero ahora sigamos con el ejemplo. La siguiente palabra a tener en cuenta es emitting.

Halos de emisión de luz

Existen diversos tipos de halos y los más indicados para simular cosas como fuegos y explosiones son los llamados halos de emisión de luz. En estos halos todas las partículas funcionan como minúsculas fuentes de luz (aunque no arrojan luz sobre otros objetos).

La palabra emitting indica a POV que el halo será de este tipo. Luego sigue la palabra spherical_mapping. Esta palabra junto con la siguiente -linear- controlan el modo en que las partículas



Serie de ejemplos con halos de emisión y un experimento.

quedarán distribuidas y la densidad que existirá en cada punto. Según los señores del Pov-team, la función de mapeo de densidad elegida (aquí spherical_mapping) mapeará valores de puntos tridimensionales en un array unidimensional. Los valores mapeados oscilarán de 0 a 1 y serán empleados por la función de densidad elegida (linear) para conseguir valores de densidad para una distancia dada comprendida en ese rango. Nótese que a pesar de la palabra "mapeo", aquí no se está mapeando realmente nada, sino que se están calculando valores de distribución y densidad de partículas.

Como todas las posibles opciones de mapeado son relativas al centro, ello nos obligará a situar siempre nuestro objeto contenedor en el centro de coordenadas para colocarle el halo, hecho lo cual podremos desplazarlo a donde queramos. (Es importante recordar este detalle).

Existen varios tipos de mapeado y funciones de densidad entre las que elegir. El mapeado esférico es el más apropiado para crear explosiones y nebulosas y la función lineal resulta también adecuada para esto o para crear soles, ya que consigue una densidad máxima de partículas en el centro del objeto contenedor y mínima en su superficie.

Una vez escogidos estos parámetros el siguiente paso será establecer un mapa de colores. Con dicho mapa describiremos los colores que mostrará el halo dependiendo de la densidad de cada punto. Los colores de los intervalos iniciales del mapa (con rangos próximos a cero) se usarán para densidades bajas y los del final (valores próximos a 1) para densidades altas. En el ejemplo de la figura el halo será amarillo en el centro (mayor densidad) y tenderá a rojo en los bordes. En cuanto al valor "transmittance" de los intervalos del mapa, indica la translucidez del campo para la densidad que corresponde a cada color (los valores 0 indican zonas casi opacas).

Por último la palabra samples indica a POV cuantos muestreos han de hacerse para cada rayo que atraviesa el halo. Lógicamente, a mayor número de samples, mayor calidad tendrá el halo y menor probabilidad habrá de que apreciemos efectos extraños debidos a un muestreo insuficiente. Normalmente no tenemos que preocuparnos del muestreo -que por defecto es de 10- a menos que estemos trasteando con frecuency.

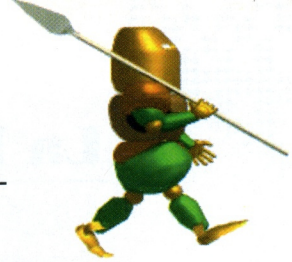
Nace una estrella

Podemos ver el resultado de generar el ejemplo anterior en la primera de las escenas de la serie de la explosión. Es un resultado pobre. Para mejorarlo basta con aumentar la densidad del campo. Como ya hemos dicho, para cada valor de densidad en cada punto (del muestreo del rayo) corresponde un color del mapa de colores y un valor de transmisión. El color final de cada pixel dependerá de las densidades de polvo halladas en los muestreos hechos siguiendo a cada rayo. (Los colores se sumarán atendiendo a la transparencia fijada, lo cual es algo que no sucede en los otros tipos de halo). Una manera de conseguir un campo más denso es poner un valor de transmittance negativo en los intervalos más densos del mapa de colores. De esta manera, como este valor se promedia con los intervalos adyacentes, el resultado es un espectacular aumento en la densidad del objeto halo. De esta manera, con -1 para transmittance en el punto de máxima densidad, la escena pasa a ser la que puede verse en la imagen situada junto al primer halo. El objeto así obtenido recuerda mucho a un sol.

Perfilando una explosión

Si queremos obtener una explosión o una nebulosa estelar debemos lograr un aspecto más caótico. Esto podemos conseguirlo añadiendo la palabra turbulence seguida de un factor de idem dentro del bloque de sentencias que forman el halo. La escena siguiente se ha conseguido aplicando una turbulencia de 1.5 al ejemplo de la escena precedente.

Con la turbulencia las partículas del halo son desplazadas de modo aleatoria, lo que puede hacer creer que hay corrientes dentro del halo. Por otro lado al hacer esto se revela la forma esférica del objeto contenedor. Esto ocurre porque la turbulencia desplaza partículas de zonas de alta densidad por distintas partes de la esfera. De hecho muchas partículas serán desplazadas fuera del objeto contenedor. Estas partículas dejarán de ser visibles puesto que POV sólo muestrea las que quedan dentro del contenedor.



Como dichas partículas de alta densidad han quedado, gracias a la turbulencia, esparcidas por toda el área de la esfera, el resultado será que la superficie del contenedor quedará "dibujada", ya que fuera del objeto las partículas, de repente, no se dibujan. En una palabra: ha aumentado el contraste de la distribución. (Recordemos que con la distribución inicial de partículas, estas tendían a densidad 0 en la superficie del objeto).

Además, al perderse muchas partículas, nuestro halo pierde brillantez. Hay un modo sencillo de remediar el efecto descrito. Consiste en escalar el halo (incluyendo una sentencia scale dentro de la declaración del halo) para reducir su tamaño. Hecho esto podemos incluir otra orden scale al final del bloque del objeto contenedor. Esta nueva orden deberá tener un factor de aumento del tamaño adecuado para invertir la reducción anterior. La nueva orden afectará las dimensiones del halo -ya que su bloque de sentencias está dentro de las de objeto contenedor-, dejando su tamaño igual al del principio. Ahora, sin embargo, el objeto contenedor es dos veces mayor y el problema habrá desaparecido (si esto no ocurre es que el factor de turbulencia es muy alto y deberemos usar un factor de escala mayor). Podemos ver el resultado en el otro ejemplo.

Por último hay que recordar un buen truco para aumentar el realismo. Este truco se basa en el uso de la palabra frecuency. Por defecto el mapa de colores no se repite en el campo de densidades; se usa el color 0 para la densidad 0, el color 0.3 para la densidad 0.3, etc. Con frecuency podemos indicar al programa cuantas veces queremos repetir el mapa dentro del campo de densidades. Así, como indica el manual, usando una frecuencia de 2, el color 0 se usaría para la densidad 0, el 0.5 para la densidad 0.25 y el 1 para la densidad .50. Para densidades por encima de este valor la tabla de colores volvería a repetirse desde 0 hasta 1.

Para las explosiones el manual recomienda un valor de 2 para frecuency y el uso de un mapa periódico de colores. ¿Que qué es esto? Pues se trata de una mapa de color normal, donde los intervalos han sido diseñados de tal forma que las transiciones de color son suaves, lo cual es lo más adecuado para nuestra explosión. Veamos el ejemplo:

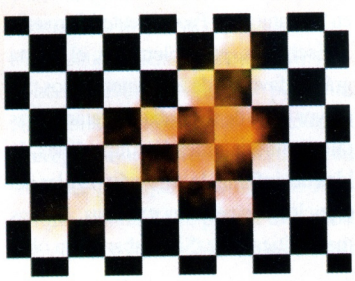
```
[ 0.0 color rgbt <1, 0, 0,  1> ]
[ 0.5 color rgbt <1, 1, 0, -1> ]
[ 1.0 color rgbt <1, 0, 0,  1> ]
```

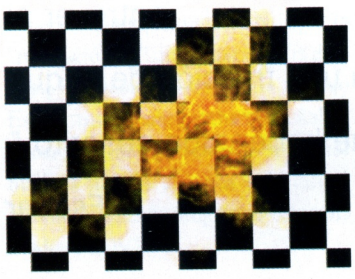
Este mapa es muy parecido al utilizado en los ejemplos anteriores. La única diferencia es que se añade un intervalo más para asegurarnos de que los colores del primer y último intervalo coinciden. De este modo el color comienza en rojo, tiende a amarillo y luego vuelve a tender a rojo. Para advertir la mejor la diferencia lo mejor es hacer pruebas usando alternativamente el mapa normal y el periódico.

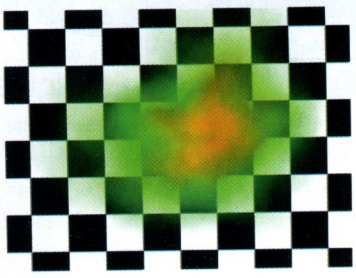
Por último conviene recordar que al aumentar la frecuencia aumenta la posibilidad de que se produzcan errores visuales debidos a un muestreo insuficiente. Para evitar esto bastará con aumentar el valor de samples, normalmente doblándolo (lo que también ¡ay! aumentará el tiempo de render).

Otros halos y experimentos

Tan sólo hemos empezado a arañar las posibilidades que nos permiten los halos de POV. Sólo con lo ya visto podemos empezar a realizar todo tipo de experimentos. Podemos, por ejemplo, trastear con los colores del mapa y usar colores que no sean tan "lógicos" como el rojo y amarillo o bien probar a usar más de un halo para un único objeto contenedor. El caso es que sólo hemos estudiado un tipo de halo. Aún quedan por estudiar los halos glowing (que emiten luz pero que también la absorben), los halos de atenuación o absorción de luz (que pueden emplearse para, por ejemplo, simular nubes o humos) y los halos de polvo. Además, aunque no tiene nada que ver con los halos, también podemos emplear la palabra atmosphere para conseguir "atmósferas" de todo tipo para nuestras escenas. ¡Adelante y a probar!







Homenaje a Escher



"Si supiérais lo que he visto en la oscuridad de la noche... A veces casi me ha vuelto loco la aflicción de no poder reproducir lo que veo. Comparado con ello, todo dibujo es un fracaso que no deja entrever ni siquiera una fracción de lo que tendría que haber sido descrito."

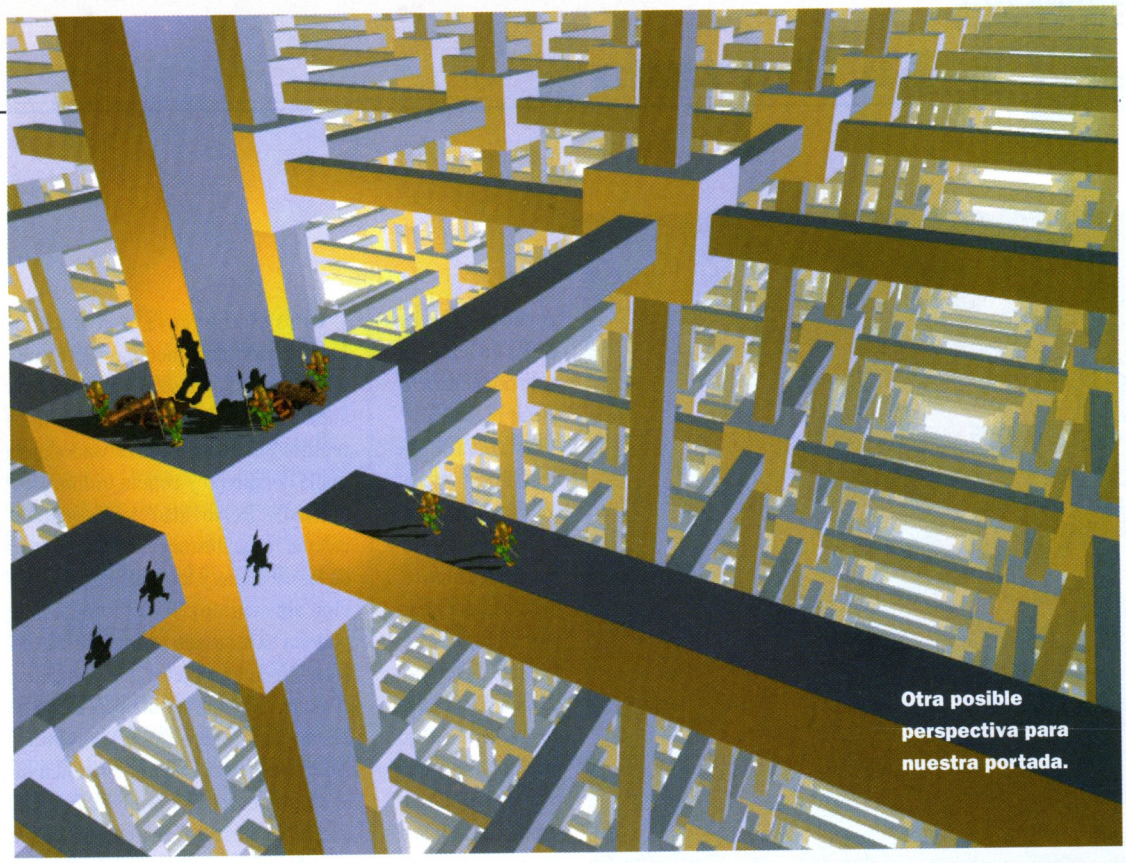
Escher

Quien conozca la obra de Maurits Cornelis Escher no habrá tenido dificultades en darse cuenta de la similitud entre la imagen de la portada y la litografía "Partición Cúbica del Espacio" realizada en 1952 por este artista. Esta semejanza es intencional ya que con nuestra portada de hoy pretendemos realizar un pequeño homenaje a este singular constructor de mundos imposibles.

¿Quién es M. C. Escher?

Es casi seguro que el lector habrá tenido la ocasión de admirar alguna vez las fantásticas escenas creadas por Escher ya que éstas se hallan por todas partes: en portafolios, posters, etc... Algunas de las ideas de Escher han sido, además, utilizadas en otras obras, como por ejemplo en la película "En el laberinto" de Jim Henson, en donde aparecía un laberinto de escaleras que no llevaban a ninguna parte y donde conceptos como arriba o abajo no tenían ningún sentido. (También se han realizado versiones de esta idea en el mundo del comic. La última que ha visto el autor de estas líneas está en "Creatura" de Paolo Eleuteri Serpieri).

Escher es, probablemente, el artista gráfico favorito de los matemáticos. La mayor parte de sus trabajos tratan temas tales como geometrías imposibles, la idea del infinito, la partición regular del espacio, perspectivas extrañas, figuras matemáticas, etc. Sin embargo, su obra



Otra posible perspectiva para nuestra portada.

suele agradar también al público en general debido al extraño toque de fantasía con el que adorna muchos de sus trabajos (Belvedere, Subiendo y bajando, Cascada, Cubo de escalera y muchos más…)

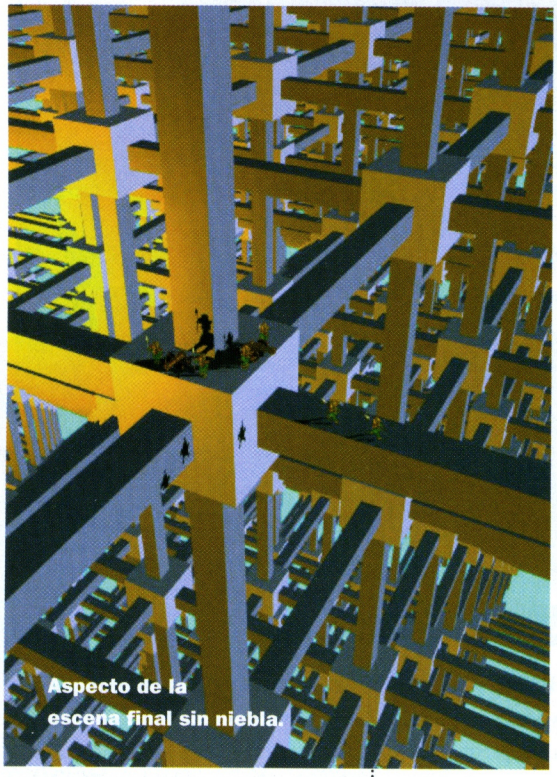
Escher fue un artista desconocido durante casi toda su vida (1898-1972). Su obra era de tan difícil clasificaciónque la critica la ignoraba. Los objetivos de su trabajo diferían demasiado de los del resto de la comunidad gráfica. En cuanto al dominio de la técnica gráfica empleada –dibujo, litografía, grabado en madera, etc.– aunque importante, era para él sólo un medio con el que plasmar las ideas que intentaba representar. Ideas que requerían docenas de bocetos y un trabajo muy superior al que podría suponerse a priori, ya que Escher carecía de conocimientos matemáticos. (Este hecho resultaba increíble para muchos matemáticos que reconocían conceptos matemáticos importantes en su obra). ¿Por qué tanto trabajo? Pues porque, aparte de la propia dificultad que entrañaban muchos de los temas, Escher tendía a representar sus ideas con la precisión de un programa de render (en lo tocante a la perspectiva).

No reproducimos hoy aquí ninguna de las obras de Escher pero el lector interesado no tendrá dificultades para admirar su trabajo. Quizá la opción más simple sea adquirir alguno de los libros-portafolio o los posters que pueden encontrarse en las tiendas de comics. Entre los libros, los más recomendables son "M.C. Escher The graphic Work" y "El Espejo Mágico de M.C. Escher" de la editorial Taco.

¿infografia & Escher?

En la última etapa de la vida de Escher los ordenadores ya eran algo bien conocido -aunque aún estaba lejos la "revolución Pc" que se produciría después. En cuanto a la Infografía, cuando Escher murió aún estaba en pañales. Pero, ¿qué habría ocurrido si Escher hubiera nacido 20 años más tarde? ¿Habría empleado programas como 3D Studio o POV?

Esta idea puede parecer un sacrilegio a algunos de los admiradores de Escher pero quizá no sea tan descabellada. No hay que olvidar que Escher nunca demostró interés por técnicas como el óleo, la acuarela o el aguafuerte. Lo que Escher buscaba ante todo era representar ciertas ideas, no crear belleza (algo que de todas maneras logró como efecto colateral en muchos de sus trabajos). Por eso este artista apenas utilizó el color excepto cuando fue necesario para ilustrar más fielmente alguna idea dada. Las técnicas escogidas por Escher fueron aquellas con las que podía alcanzar un gran nivel de detalle y crear reproducciones (que persiguiese objetivos propios no significaba que no deseara que su obra fuese conocida por el mayor número posible de personas). Cuando Escher encontraba una técnica nueva o algún otro tipo de material, con los que se pudiera alcanzar una mayor precisión, los adoptaba enseguida. Este amor por el detalle se hace evidente en obras como "Más y más pe-



Aspecto de la escena final sin niebla.

queño", "Límite Circular" y otras donde este paciente artista llegó a emplear una lupa para representar figuras que medían... ¡menos de medio milímetro! Por todo esto el autor de estas líneas tiene la impresión de que probablemente Escher no hubiera rechazado una herramienta con la que puede alcanzarse un nivel de detalle aún mayor (el ordenador, claro).

Por esta precisión y por los trucos que realizaba con la perspectiva, Escher puede ser calificado de "Render humano". Si pensamos en las técnicas empleadas por Escher para conseguir estos resultados, nuestra admiración no puede sino crecer; litografia, grabados en madera, mediatinta, etc.

El tema de la portada

Una vez adoptada la idea de realizar un pequeño homenaje a M. C. Escher en la portada de Rendermanía, el segundo paso fue escoger un tema tratado por él para realizar la escena.

En la obra de este artista hay muchas ideas susceptibles de ser tratadas infográficamente. Tenemos, por ejemplo "Mano con esfera reflejante (donde toda la escena se ve gracias al reflejo producido en una bola de cristal sostenida por la mano del propio Escher), "Envoltura" (donde puede verse una cabeza fragmentada dibujada sobre una corteza pelada procedente de alguna fruta), "Profundidad" (donde se ilustra el tema de la extensión espacial infinita empleando peces voladores) y muchas, muchas composiciones más.

Ilustrar el primer ejemplo hubiera sido sencillo empleando POV o Polyray aunque, eso sí, habríamos necesitado un modelo algo complejo (un robot o algo similar) para sostener la bola. "Envoltura" tampoco habría dado demasiados problemas desde POV, ya que hubiera sido fácil crear una textura con partes transparentes. Y en cuanto a "Profundidad", se podría haber utilizado la sentencia grid de Polyray para representar miles de objetos en el espacio, sin apenas costes en memoria o tiempo. Realmente esta última idea ya se empleó en una escena publicada hace muchos números, aunque no se emplearon peces voladores, sino aviones.

Otras escenas cuya representación hubiera sido igualmente posible, aunque habrían supuesto un trabajo muy superior son "Reptiles" (donde unos reptiles bidimensionales abandonan la malla del dibujo y cobran tridimensionalidad), "Relatividad" y "Casa de Escaleras" (que ilustran el conocido tema de las escaleras y los planos de orientación sin sentido). Para "Reptiles" se podrían haber usado operaciones de escalación en los objetos que abandonan el plano 2D y una textura para el dibujo 2D de cuyo plano salen los reptiles. Aquí la dificultad habría estado en intentar representar el propio dibujo 2D que se ve en la escena, puesto que en él cada reptil se dibuja con los huecos que dejan los demas. Para las otras escenas se podría haber empleado #while y rnd para crear docenas de escaleras, aunque no habría sido tan sencillo conseguir un buen efecto.

La partición cúbica del espacio

Finalmente se optó por "Partición cúbica del espacio". Esta escena fue escogida en parte por la falta de dificultad que entrañaba su realización en POV (admitámoslo), y en parte porque el tema de la partición regular del espacio fue quizá el que más interesó a Escher, el cual llegó a escribir un tratado sobre este particular.

Dicha partición regular de los espacios y superficies fue empleada con gran maestría por Escher en numerosos trabajos en los que multitud de figuras se complementan una a otra y también en las composiciones donde se plasmaba la idea del infinito.

La finalidad de la litografia "Partición cúbica del espacio" es, precisa-



mente, representar una extensión espacial infinita. Para ello Escher dividió el espacio en cubos idénticos a los que conectó con travesaños. La verdad es que este artista realizó trabajos más bellos para ilustrar el tema del infinito. Nuestra portada puede dar una idea engañosa al lector que desconozca la obra de Escher, acerca de la dificultad de la misma.

La portada

Nuestra portada no es, ni pretende serlo, una reproducción exacta de la litografía comentada. Para empezar la escena de Escher tiene una perspectiva menos acusada que la nuestra y está hecha en blanco y negro. Además en su escena Escher sugiere un conjunto infinito de cubos, mientras que en nuestra portada pueden verse un par de zonas donde no los hay, lo cual da a entender que se está representando una estructura muy grande pero no infinita. Finalmente, en la litografía no hay otra cosa que cubos y espacio mientras que en la nuestra se han colocado unos cuantos guerreros con cañones para reforzar la impresión del gran tamaño de la estructura.

En cuanto a la realización, la portada está construida usando una única pieza básica; una unión con 7 objetos cube. El primero sirve para representar a los cubos de la escena y los siguientes seis se usan para crear los travesaños. La unión de un gran número de estas piezas básicas da como resultado la imagen de la portada cuyo render, al no tener que calcular diferencias ni intersecciones de objetos, es bastante rápido.

Hallar la posición de un cubo dado en la escena no es difícil. La pieza básica, travesaños incluidos, mide 700 unidades de largo y está colocada en el espacio comprendido entre <0,0,0> y <700,700,700>. Los cubos forman una malla cúbica que se extiende a lo largo de los lados positivos de los tres ejes, partiendo de <0,0,0>. Por tanto para obtener la posición del quinto cubo horizontal (X) de la segunda fila (Y) en la tercera rejilla (Z) de la malla de cubos, bastará con darle a POV el siguiente cálculo: <(5*700)+350, (2*700)+350, (3*700)+350>. (Las sumas son necesarias para obtener la posición del centro del cubo deseado.)

Teniendo esto en cuenta la colocación de la cámara resulta muy sencilla:

```
location  <(locatX*700)+350+prelX,
           (locatY*700)+350+prelY,
           (locatZ*700)+350+prelZ>
```

Las variables de nombre locat guardan la ubicación espacial de la cámara en distancias "cúbicas". Esta posición puede estar ocupada ya por un cubo. Esto es, podemos decidir colocar la cámara dentro de la malla de cubos. Por ello también se emplea una variable adicional para cada eje (prel) que se encarga de desplazar la cámara para asegurarnos de que ésta no quedará dentro de ningún cubo. En cuanto al objetivo de la cámara lo fijamos de la misma manera, indicando a qué cubo mirar con las variables mircub. De esta manera resulta sencillo colocar y orientar la cámara dentro de la malla de cubos y colocar personajes sobre los mismos.

La escena usa una niebla de tipo constante. Recordemos que fog es la sentencia usada en POV para crear escenas con niebla. El color de esta niebla tenderá a ser el especificado por "colour rgb", el parámetro que sigue a la palabra distance determinará la distancia a la que podrán verse los objetos con un 36% de transparencia y la palabra turbulence servirá para agitar un poco la distribución de la densidad dentro de la niebla. Para dar un poco de contraste a la escena se han empleado dos focos de distinto color en diferentes posiciones. Esta escena es ideal para realizar experimentos con luces, niebla y distintos tipos de cámara.

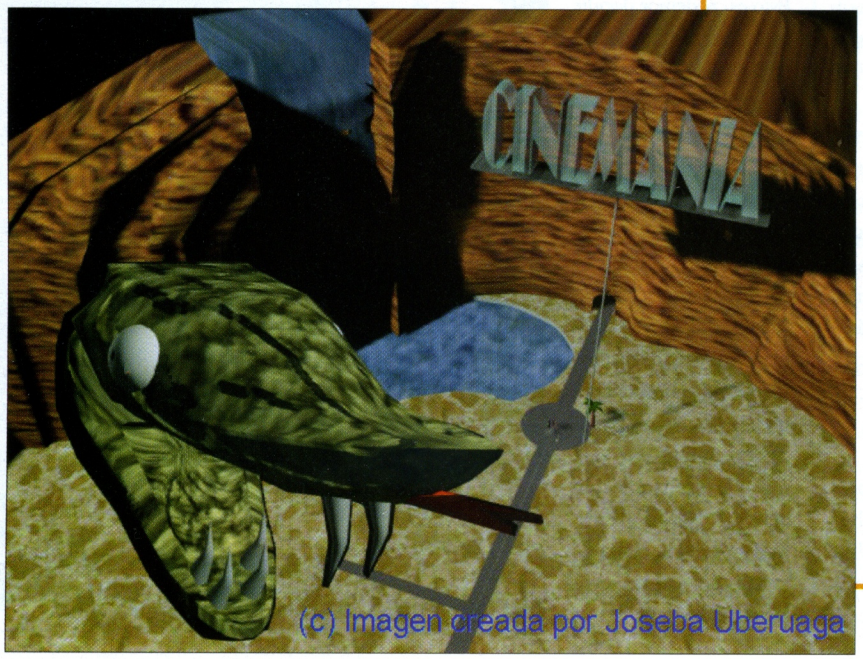


La misma imagen con otros valores para fog.

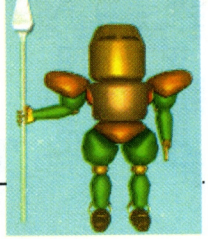
Preguntas con respuesta

Este mes, atendemos en esta sección numerosas preguntas que nos han sido remitidas por distintos medios. Esperamos que las respuestas seas satisfactorias para todos los rendermaniacos.





Joseba Uberuaga nos envía un par de imágenes hechas con 3D Studio sobre las que pide una opinión. Bien, para ser breves (el espacio se acaba), creo que por ahora tu imaginación sobrepasa bastante a tu capacidad técnica. Me gusto la fantasía de ambas imágenes (al principio no me di cuenta del tamaño de la cabeza de la serpiente) pero aún tienes que practicar mucho con el modelador para conseguir las imágenes que quieres. Ánimo, estás en el buen camino.



Nota importante. Podéis remitirnos vuestros trabajos o consultas, bien por carta a la dirección que figura en la segunda página de Pcmanía, o vía e-mail a rendermania.pcmania@hobbypress.es

Un **rendermiaco** que responde al nombre de **Nacho** nos remite la siguiente duda. Está haciendo los sprites de un juego con POV y según afirma "el axis de la nave que estoy moviendo está mal. ¿Cómo lo arreglo? ¿Cómo lo muevo?". Estas preguntas denotan, estimado pov-colega, que eres todavía algo novato con POV. En tu e-mail no explicas si la nave ha sido creada desde un modelador y exportada a POV, o si has creado el modelo entero usando el lenguaje escénico de este raytracer. Si tu caso es el primero deberías leer el artículo "Cómo exportar modelos a POV" del número anterior de Rendermanía. Si por el contrario el modelo está hecho en POV, entonces se trata, simplemente, de errores en el concepto o en el diseño. Es difícil determinarlo ya que no das detalles concretos acerca de cuál es el error ni adjuntas el fichero del modelo. ¿Sabes que las operaciones translate son relativas a la última posición del objeto y no absolutas? ¿Sabes que si el objeto no está centrado las órdenes de escalación desplazarán al objeto además de alterar su tamaño? ¿Sabes que las rotaciones se efectúan siempre con respecto al origen en las coordenadas <0,0,0>? (Si quieres orientarlo en otra dirección y no está en <0,0,0>, tendrás que trasladarlo primero a dichas coordenadas, rotarlo y volverlo a trasladar a las coordenadas primitivas). ¿Tienes en cuenta el orden de las operaciones al construir el objeto? ¿Has comprobado que las medidas entre los objetos componentes del modelo sean congruentes? Si no tienes en cuenta todos estos detalles te sera muy difícil construir un objeto con POV. Te recomiendo que leas los números anteriores de Rendermanía.

Ricardo Crespo nos envía algunas preguntas referentes a la iluminación en POV:

A) "¿Existe algun .inc que incluya distintos tipos de iluminación?" Por supuesto. Están los ficheros de ejemplo que se hallan dentro del directorio lights (dentro de pov3demo).

B) "¿Existe algún tipo de tutorial sobre la iluminación, incluso no referido a POV sino a cine y fotografia?". Tendrías que echar un vistazo a "Raytracing II" publicado por Anaya o bien seguir esta sección, donde últimamente se están estudiando exhaustivamente todos los apartados de POV. De todas maneras y por la forma de tu pregunta he de decirte que existen ciertas diferencias entre la iluminación en el mundo real y la del mundo virtual de POV. Por ello aunque un texto sobre iluminación en general puede ser útil para comprender ciertos principios básicos, muchos de ellos no son aplicables al mundo de POV (y menos aún a programas de generación que usen otros tipos de render menos "cercanos" al mundo real). Toma nota de que…

1) Las fuentes de luz son puntuales.

2) Las fuentes se definen por su color, no por su energía o por otros aspectos típicos de fotografía (y que conste que yo no se nada de fotografía). Los colores finales en la escena dependen de su color propio y del de las fuentes de luz.

3) Lo más parecido a un foco de fotografía son las luces tipo Spotlight.

4) Es posible controlar la distancia a la que llega la luz y su difusión con palabras como "fade_distance" y "fade_power".

Fernando Zuriaga pregunta "¿Existen empresas a las que puedas enviar el fichero fuente de POV y las trayectorias de los objetos y la cámara para realizar animaciones?". La respuesta es que no, que nosotros sepamos, y si existieran ten por seguro que te cobrarían por ello mucho más de lo que cualquiera estaría dispuesto a pagar. Existen, por supuesto, empresas que pueden generarte animaciones, pero habitualmente las han diseñado ellos a petición tuya y te cobran bastante por ello. (Normalmente trabajan para casas de publicidad y otras empresas). De todas maneras hacer animaciones complejas con POV no es tan "fácil" como en 3D Studio y otros programas de tipo profesional , y por ello estas empresas no suelen trabajar con POV. (Aunque yo creo que el uso de POV para escenas sueltas como portadas y similares si está justificado profesionalmente en muchos casos por lo que te ahorras en el trabajo de luces, atmósfera, efectos, etc).

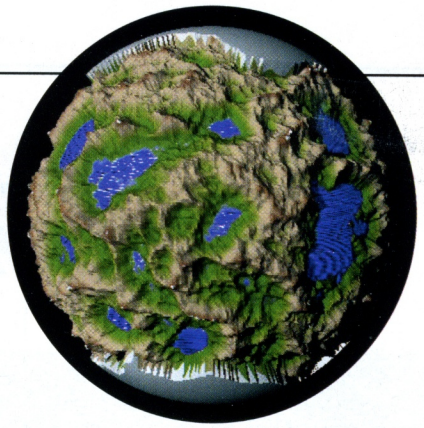
Respecto a tus otras dudas: Terrain Maker es una excelente utilidad de alzado de terrenos y, sinténdolo mucho, no conozco la respuesta en lo concerniente a tu pregunta sobre Mpeg.

Nuestro amigo **Pooky** (ya sabéis, el autor del Poling) ha quedado asombrado al enterarse de que las técnicas necesarias para programar trazado de rayos se estudian en la Universidad en Óptica (en Física) y también en la carrera de Informática. ¿Y de qué te extrañas? La información es de dominio público (como lo es la teoría de la relatividad, la mecánica cuántica, etc.). Ahora bien, ¿cuánta gente es capaz de hacer algo con ella logrando la calidad de POV? Poca, muy poca.

¿Arrays en POV? ¿Te refieres a la palabra matrix? Que yo sepa sirve para componer matrices de transformación para los objetos y como esto ya se hace automáticamente con las operaciones normales (traslación, escalado, rotación), no acabo de ver claro la utilidad de esta nueva palabra. Si te refieres a arrays como los que tiene Polyray, POV (¡Ay!) aún no los tiene. Suerte con la próxima versión de Poling.

Ricardo Villanueva ha enviado unas escenas creadas con POVafx y Ftpov que han llegado machacadas. ¡Envíalas de nuevo please!

Javier Subijana Eixarch pregunta si existe algún libro de Visual Reality en castellano. La respuesta es que no y dudo que la página Web de Micrografx tenga un doctorial sobre su programa (lo que suelen tener los webs de las empresas son pequeños ficheros con cuestiones sobre actualizaciones, bugs, etc). Sorry.



POV vs Polyray

Como sin duda sabéis POV y Polyray son dos programas que comparten muchas características comunes. Casi podría decirse que son primos hermanos. La razón es que Alexander Enzmann, el padre de Polyray, forma parte también del grupo povteam, desarrollador de POV. Sin embargo, con la aparición de la última versión de POV, las diferencias entre ambos programas cobran importancia.

El sistema de orientación de los ejes, las propiedades de la cámara, el funcionamiento del color, las operaciones CSG, etc., son algunas de las características compartidas por ambos programas. Estas semejanzas ocasionaron que, en general, cuando aparecía una nueva utilidad para POV, la correspondiente versión para Polyray no tardase demasiado en publicarse.

Las mayores diferencias estaban en el mejor soporte de animación que implementaba Polyray y en sus primitivas adicionales. POV contrapesaba esto por el hecho de ser freeware (Polyray no lo es, Enzmann pide una pequeña cantidad) y por el mayor número de tools hechas para él. De todos modos podía decirse que, hasta la última versión de POV, éste estaba en desventaja con respecto a su competidor. En cierta manera POV y Polyray mantienen una carrera amistosa en la que suele ganar el equipo que lanza al público la última versión. Por ahora esa última versión la ha puesto el povteam.

La última versión de POV

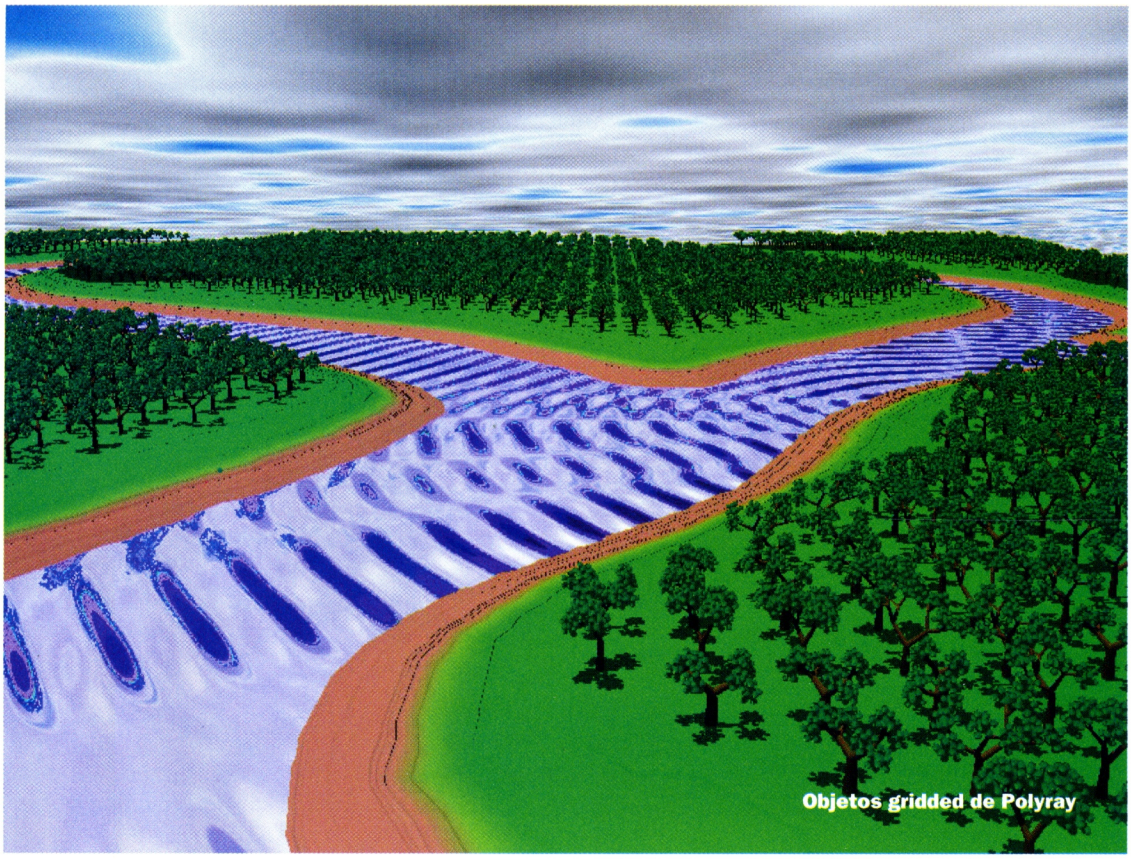
La última versión de POV ha recobrado el terreno perdido en el campo del modelado al incluir los objetos básicos de que disponía Polyray: las superficies de revolución y extrusión. Además las nuevas sentencias de programación (#if , #while, etc), han permitido a POV tomar ventaja sobre su primo. (Polyray también tiene ifs y evaluación de lazos pero parecen estar reservados a la animación). Hay, no obstante, algunos apartados en los que Polyray ofrece ventajas. Por ejemplo, tiene campos de alturas (heightfields) esféricos, lo cual puede servirnos para crear pequeños planetoides. (Los heightfields de POV sólo se extienden sobre cajas). Otras características de Polyray que serían deseables en POV son los objetos grid y los arrays.



Objetos Gridded

En el número anterior nuestra portada consistió básicamente en un castillo construido desde POV. La renderización de este castillo, en el que se emplearon bucles, demandó mucha memoria. ¿Habría sido posible con Polyray? La respuesta es que sí, aunque habríamos tenido que cambiar bastantes cosas en la filosofía del diseño.

Para empezar Polyray no dispone de bucles para crear objetos. Ello nos obligaría a definir las alme-



Objetos gridded de Polyray

nas como uniones donde cada objeto requeriría una línea, lo cual es, aunque algo molesto para las torres más anchas, perfectamente posible ya que podemos definir uniones de uniones. Por otro lado podríamos, quizá, haber usado objetos gridded para la escena del número 0 de Rendermanía, en donde aparecía un pueblo medieval. Los objetos gridded de Polyray permiten representar una matriz bidimensional de objetos sobre el plano X-Z. La distribución de estos objetos estará determinada por un bitmap suministrado como entrada y el único problema es que cada objeto de la matriz deberá ser escalado y trasladado para que ocupe un área espacial que va desde la esquina <0,0,0> hasta <1,1,1>. En nuestro pueblo había casas de distintas dimensiones y ello nos habría forzado a escalarlas de modo que la mayor fuese igual a dicho volumen espacial. (Las demas deberían ser aún más pequeñas).

El tamaño del Gridded está determinado por el número de pixels del bitmap de entrada. Si este tiene 256*256, el número total de objetos puede ser hasta de... ¡65536! Las buenas noticias son que esta matriz de objetos se renderizará sin exigir la descomunal cantidad de memoria que sería de esperar en POV y que la velocidad del render será también muy superior a lo que, en principio, podríamos esperar. (Porque todos los objetos ocupan un volumen predeterminado en la matriz). Veamos el formato de gridded:

```
object {
    gridded "bitmap.tga",
            objeto 1
            objeto 2
            ...
            objeto n

    transformaciones
}
```

Como puede deducirse por las líneas anteriores, el array del gridded puede estar compuesto por diferentes objetos. ¿Cómo se determina cuál se aplica en cada volumen espacial de 1*1*1? Bien, pues ello se determina según el color de cada pixel. Si se está empleando un fichero tga de 16 bits, se toma el segundo byte de color y si la tga es de 24 ó 32 bits, entonces se emplea el componente rojo. Así, si el byte empleado tiene el

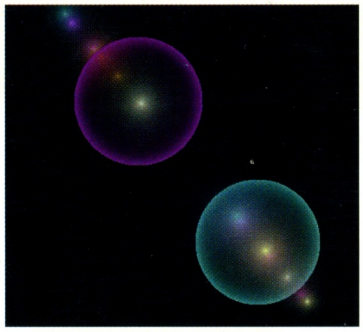
valor cero, entonces el objeto colocado será el primero definido en la declaración del gridded. Un valor 1 indicará que debe usarse el segundo objeto y así sucesivamente. Los valores superiores al número de objetos indicados en el gridded dejaran vacío el correspondiente espacio en el array.

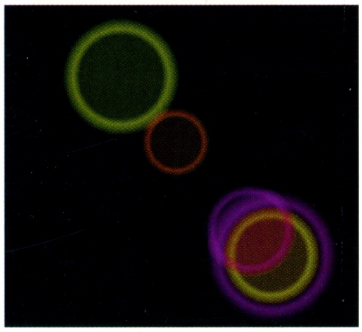
La mayor ventaja del gridded es sus escasos requerimientos de memoria. Ello hace posible preparar escenas con miles de objetos pero, desgraciadamente, todos tendrán que estar situados a la misma altura Y. Esto quiere decir que si, por ejemplo, deseáramos crear una escena bélica con miles de guerreros dispuestos en filas o en desorden, con Gridded podríamos lograrlo, pero tendríamos que prescindir de colinas e irregularidades en el terreno, lo cual quitaría bastante "tono" a la escena. Esto quizá podría solucionarse empleando muchos grids superpuestos, trasladados a diversas alturas pero la cosa tendría bastante trabajo.

Por último hay que hacer notar que la portada de hoy habría sido aún más sencilla con Polyray. Se empleó POV porque inicialmente se iban a usar efectos atmosféricos que al final no se emplearon.

Más diferencias

Otro detalle de Polyray del que carece POV, y que a primera vista no parece de importancia, son los arrays. Estos, como saben los programadores, son una manera de almacenar y representar datos. En Polyray los arrays suelen emplearse para almacenar posiciones de objetos en los distintos frames de una animación pero en POV, si (¡pena!) existieran, podrían emplearse para más cosas. Pensemos, por ejemplo, en la colocación aleatoria de objetos con la función rnd. Si fuera posible almacenar las localizaciones devueltas por rnd en un array de modo que pudieran consultarse podríamos eludir fácilmente el problema de crear un nuevo objeto en una posición ya ocupada y, aún mejor, sería factible preparar animaciones con detecciones entre objetos. Quizá en una próxima versión de POV esto sea posible. Por ahora (¡snif!) no lo es.





Las flares de Polyray poco tienen que ver con los halos de POV.

Otras diferencias pueden encontrarse en el campo de los efectos. Tanto POV como Polyray disponen de halos pero el significado de esta palabra dependerá del programa que vayamos a usar. En POV, los halos crean una distribución de partículas dentro de un volumen espacial. Estas partículas interactuarán con la luz dependiendo de su distribución y de las propiedades del halo. En cambio en Polyray los halos son texturas bidimensionales que se aplican sobre la imagen final ya calculada, y que sirven para simular los efectos que se producen en las cámaras de TV al apuntar hacia focos de luz. Quizá las mayores diferencias estén en el apartado de construcción de texturas procedurales y en las opciones de visualización.

El futuro

Es innegable que Enzmann tendrá que trabajar muy duro si quiere que Polyray vuelva a tener ventajas sobre POV en algunos aspectos. La cosa no parece fácil ya que tendrá que competir con un numeroso grupo de programadores. Es dudoso, de todos modos, que Enzmann opte por hacer que Polyray adquiera las mismas características que acaban de implementarse en POV ya que haciendo esto siempre estaría un paso por detrás. Es más probable que este programador empiece a incorporar en Polyray efectos y características de las que carece POV. De esta manera, y dado que ambos programas se parecen tanto, puede acabar sucediendo que muchos usuarios empleen ambos programas para crear sus trabajos, empleando características de ambos.

Otra posibilidad es que se realice una fusión entre ambos programas pero en ese caso; ¿por qué esto no ha sucedido antes? En fin todo puede ocurrir. Quizá incluso puede suceder que más adelante debamos publicar otro artículo como éste, donde las posiciones de ambos programas estén invertidas. No hay que olvidar que la última versión de Polyray es ya algo antigua. La nueva debería estar ya en vías de lanzamiento y quizá entonces nos llevemos una sorpresa.