SUPLEMENTO MENSUAL DEDICADO A LA IMAGEN SINTÉTICA

Rendermanía

Número 14



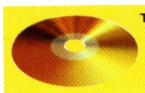
Cómo...
C rear ejércitos de orcos

Taller Virtrual
Creamos posturas para modelos-mesh

En el CD-Rom
Actualización de sPatch y conversor 3DWin

Foro del Lector
Belleza virtual

pcmanía



Todo los ficheros de inclusión y los ejemplos podéis encontrarlos en el directorio **PCMANIA\RENDER64** del CD-Rom.



José Manuel Muñoz

Crear regimientos para



Aunque se suponía que en el número 9 de Rendermanía iba a tener lugar la tan prometida batalla entre orcos y humanos, al final ésta quedó reducida a una misérrima escaramuza entre zombis y humanos. La sustitución de los orcos por los zombis se debió a que por entonces no disponíamos de ningún modelo decente de orco, detalle este que ya fue subsanado en el número 11. Más grave fue, sin embargo, el problema del brutal consumo de memoria que «POV» parecía requerir para representar escenas con cientos de modelos (y que redujo la "batalla" a sólo tres pares de luchadores). Pero, aunque parezca increíble, este problema se debía tan sólo a la ignorancia y no a una limitación de «POV». Sabed pues, oh Rendermaníacos y Povadictos... ¡que Mesh nos permitirá crear primorosas escenas compuestas por cientos de modelos complejos!

Antes de empezar, claro está, debemos pedir disculpas a los lectores por no haber hablado antes de esta importantísima sentencia del lenguaje escénico de «POV». Hace bastante tiempo, cuando empezamos a leer el doctorial de «POV», pasamos rápidamente por Mesh y nos centramos sobre todo en las sentencias de programación, pensando erróneamente que Mesh sería una optimización sin importancia. Solamente hace

batallas virtuales



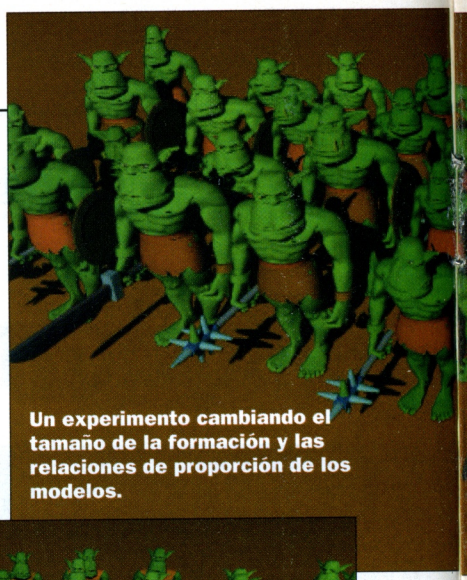
unos meses, al preparar el especial sobre «Rhinoceros», comprendimos el enorme potencial de esta sentencia gracias a que «Rhino» empleaba Mesh para exportar sus escenas al formato de «POV». Intrigados por ello estudiamos esta instrucción con algo más de atención y... bueno, ahora veréis. Nuestra única disculpa posible –aunque pobre– es que nadie más parece haber dado a Mesh el valor que merece: al menos nosotros no hemos encontrado aún una sola escena –ni en la red ni en el foro– en la que se haya empleado a Mesh para crear fantásticas escenas con cientos de modelos. ¿Por qué? En fin, puede ser que no hayamos buscado lo suficiente. Si este es el caso esperamos que algún amable rendermaníaco nos saque del error... También queremos añadir que en este artículo hemos supuesto que el lector comprende el funcionamiento de #if, #while y de otras "sentencias de programación" ya explicadas hace mucho y sobre cuya importancia no nos cansaremos de insistir.

Exportación de modelos a «POV»

Como ya saben los usuarios experimentados de «POV», nuestro raytracer dispone, desde hace mucho, de un par de sentencias –triangle y smooth_triangle– que le permiten usar objetos construidos desde modeladores poligonales como «Imagine» o «3D Studio». Para ello ha de seguirse el siguiente proceso:

1) El usuario crea su modelo empleando su modelador preferido («3D Studio», «Imagine», «trueSpace», etc.). Con ello obtendrá un modelo que no será directamente digerible por «POV» (la excepción está en el uso de algún modelador como «Moray», expresamente creado para «POV», y de otros programas del mismo tipo). En efecto: normalmente el modelador guardará la malla de triángulos que forma al modelo como un fichero binario empleando un formato propio ininteligible para «POV» (3DS en el caso de «3D Studio», obj en el de «Imagine», etc.). Aunque también es frecuente que estos modeladores ofrezcan además la opción de guardar el modelo usando el formato DXF o bien un formato ASCII sencillo, a fin de facilitar las exportaciones de objetos a otros programas.

2) Seguidamente, el usuario deberá emplear alguna herramienta de conversión para "traducir" el archivo generado por el modelador (en números anteriores ya hemos hablado de algunas de estas herramientas: 3ds2pov, 3dto3d, Wcvt2pov, etc.). El resultado será un nuevo fichero en formato ASCII y comprensible para «POV» en el que los polígonos del modelo del archivo suministrado como entrada serán traducidos a sentencias triangle y smooth triangle. Estas sentencias fueron implementadas en el lenguaje escénico de «POV» por el Povteam pensando precisamente en el caso de aquellos usuarios que deseasen emplear modelos no creados desde el mismo «POV».

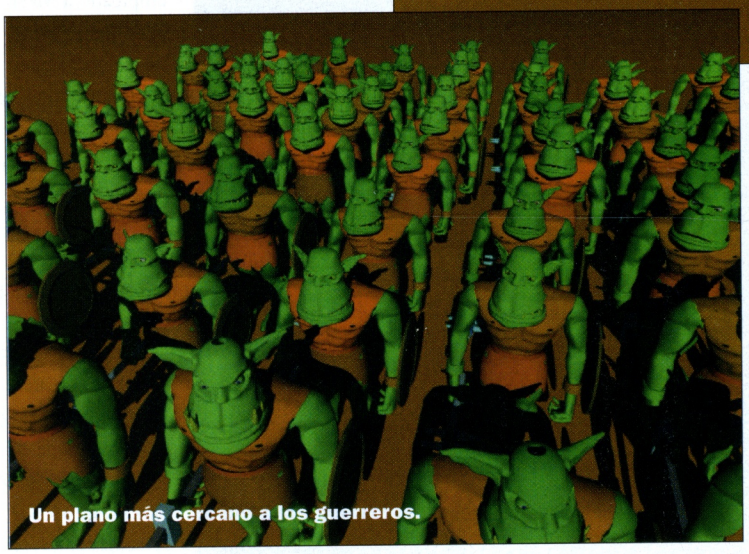


Un experimento cambiando el tamaño de la formación y las relaciones de proporción de los modelos.

Teóricamente podrían ser empleadas por el usuario para crear objetos a mano, empleando un procesador de textos, pero esto, a menos que el objeto sea algo tan sencillo como un cubo, no resulta práctico, ya que habría que calcular a ojo un montón de coordenadas de vértices.

3) Después, simplemente, habremos de incluir el archivo .POV o .INC resultante de la "traducción" en nuestro fichero escénico. Para ello escribiremos una o más órdenes #include en el archivo donde se describe nuestra escena. Esto no siempre será necesario ya que puede ocurrir que todos los elementos de la escena ya hayan sido establecidos desde el modelador y que no deseemos cambiarlos desde «POV». En estos casos las utilidades como 3ds2pov nos vendrán de perlas, ya que generan archivos .POV en los que se "traduce", además de a la malla con el modelo, a las luces, la colocación y orientación de la cámara, los colores de los objetos y sus propiedades de superficie, etc. De esta manera bastará con invocar a «POV» y pedirle que renderice el archivo .POV generado por la herramienta de traducción. Así obtendremos una imagen muy similar a la que podríamos renderizar desde el modelador que hayamos usado.

Para mayor comodidad del usuario, las mejores utilidades de conversión (como 3ds2pov) generan 2 o más archivos para «POV». Suelen crear un fichero .POV en el que se incluye la "traducción" de la cámara y las fuentes de luz, y uno o más archivos .INC donde irán las sentencias triangle y smooth_triangle con la forma del modelo. Para nuestros propósitos actuales, o sea para generar escenas complejas con cientos de modelos, lo mejor será colocar cada uno de estos modelos en la escena aprovechando las magníficas capacidades del lenguaje escénico de «POV». Por ello



Un plano más cercano a los guerreros.

prescindiremos de los ficheros .POV generados en las conversiones, quedándonos únicamente con los archivos que describen la geometría de los modelos.

Sabido esto ya sólo nos queda recordar algunos de los problemas que fastidiaron a nuestra batallita. Uno de ellos es que los objetos creados con modeladores poligonales se almacenan como mallas de caras compuestas por vértices, requiriendo cada uno de los cuales tres valores en flotante para describir su posición espacial (y aún puede ser necesaria más memoria para describir los vectores de las normales, si éstas son también almacenadas). Esto quiere decir que frecuentemente los archivos .INC que habremos de manejar serán muy grandes, con lo cual «POV» precisará mucha memoria para incluirlos en nuestras escenas. Sin embargo, lo peor es que, para cada copia del modelo empleada en la escena, «POV» requerirá (usando union) la misma cantidad de memoria. Así, si por ejemplo, un orco requiere 6 Megas de RAM, la inclusión de 6 monstruos de este tipo en una escena precisará 36 Megas. Este problema fue el mayor obstáculo para la tan ansiada batalla orco-humana.

Mesh: un milagro bien sencillo

Este problema de la excesiva demanda de memoria se debe al empleo de la sentencia union. Las utilidades de conversión algo antiguas como 3ds2pov y 3dto3d usan esta sentencia para englo-



bar a las sentencias triangle y smooth_triangle que describen a los modelos importados. Como recordaréis, union se usa para indicar a «POV» que un conjunto de objetos individuales (en este caso un conjunto de triángulos) forman un único objeto. Antes de la versión 3.0, el uso de union para los objetos importados resultaba imprescindible ya que de esta manera bastaba con especificar una única textura que se aplicaba sobre un grupo de triángulos (en vez de hacer lo propio para cada triángulo). Además, usando esta sentencia podíamos colocar sentencias de transformación que afectaban a todo el grupo de triángulos englobados por la unión. Sin embargo, union no fue creada para agrupar triángulos, sino objetos creados desde el propio «POV». La sentencia mesh, en cambio, sí ha sido diseñada exclusivamente con esta finalidad y funciona exactamente igual que union. La única diferencia radica en que con mesh los grupos de triángulos englobados se almacenan en memoria una única vez. Así, en la siguiente ocasión en que invoquemos al objeto-mesh dentro del fichero de escena, «POV» únicamente precisará una pequeña cantidad de RAM para generar al objeto, el cual será considerado como una copia del primero. Únicamente hay que recordar que mesh sólo podrá ser usada para englobar triángulos (si usamos una sentencia mesh para englobar a varias mesh o para englobar a objetos normales, el parser nos dará el error "No triangles in triangle mesh").

Dado que las utilidades como 3ds2pov o 3dto3d no generan aún ficheros con mesh sino con union (al menos en las versiones más modernas que hemos podido encontrar), deberemos –si usamos estas herramientas y queremos aprovechar la potencia de mesh– emplear un editor de textos para sustituir las sentencias union por mesh. Otros programas más modernos como «3Dwin 3.0» de Thomas Baier (que hemos incluido en el CD) sí generan conversiones mesh.

Seguidamente comentaremos la forma más adecuada de usar mesh y estudiaremos la lista de experimentos que llevamos a cabo para usar mesh con modelos antropomórficos y que desembocó en las diversas escenas que ilustran este número.

Creando nuestro primer regimiento de orcos

El modelo de orco empleado para crear las escenas básicas de nuestro regimiento de ejemplo es el que presentamos en el número 11 de Rendermanía. La cabeza de este orco fue creada con «Rhinoceros», el cual sí hace exportaciones con mesh, por lo que no fue demasiado difícil preparar a la mencionada cabeza para usarla en los primeros experimentos hechos para comprobar el funcionamiento de esta sentencia. Lo mismo cabe decir del cuerpo, el cual está basado en una malla que hallamos por la Red. La cabeza original de dicha malla fue eliminada y las proporciones del cuerpo alteradas desde «Rhino» para adecuarlas a las formas exageradas y monstruosas de los orcos. En cuanto al faldellín y a las armas se hicieron también en «Rhino» o se importaron de «Imagine». Por último, el modelo completo se exportó a «POV» en varios archivos; uno para la cabeza, otro para el cuerpo y otro para las armas... (al menos así fue en las primeras pruebas).

Ahora veamos la forma más sencilla de emplear Mesh. Después de utilizar una herramienta de conversión lo más probable será que obtengamos uno o más archivos .INC, cada uno de los cuales estará formado por varias sentencias union o mesh (dependiendo de si la herramienta usada es un poco antigua o no). Luego, para cada fichero, sustituiremos –si las hay– a las sentencias union que engloben a los triángulos por instrucciones mesh. También deberemos eliminar, si las hubiera, las definiciones de la cámara y de las fuentes de luz. (En el caso de 3ds2pov, todo esto se pone aparte pero no todas las utilidades de conversión funcionan igual). Por último, y en caso de que no la hubiera, deberemos crear una union que englobe a todas las mesh del fichero .INC sobre el que estemos trabajando. Este último paso será imprescindible para referenciar luego a las mallas desde el fichero de escena con sentencias como las que podéis ver en el **cuadro 1**.

```
                    ...
                    ...
#declare cuerpo1=object{#include "body1.inc"}
                    ...
                    ...
object{cuerpo1}
                    ...
                    ...
```

Cuadro 1

Así, cuando «POV» esté compilando la escena, al encontrar el include, el programa irá cargando el archivo y pasando los datos de las mallas a la RAM. Más tarde, el modelo podrá ser invocado cuantas veces deseemos usando sentencias object y «POV» tan solo precisará una pequeña cantidad de memoria para los datos propios de cada referencia al objeto inicial. Ni que decir tiene que sería una estupidez usar sentencias como...

```
object{#include "body1.inc"}
```

...cada vez que deseásemos crear un nuevo modelo en la escena, ya que entonces «POV» cargaría nuevamente el archivo cada vez que hallara una de estas sentencias en el fichero de escena. Examinad el listado de la siguiente figura.

"Ejemplo de creación de un cuadro de guerreros orcos"

```
#declare orco1=object{#include "orco.inc"}
#declare nfilas=8
#declare xpos=0
#declare zpos=0
#while(colum!=0)
   #declare xpos=0
   #declare fila=8
   #while(fila!=0)
        object{orco1 translate <xpos,0,zpos> }
        #declare xpos=xpos+120
        #declare fila=fila-1
   #end
   #declare zpos=zpos+120
   #declare nfilas=nfilas-1
#end
```

Este ejemplo corresponde a una de las primeras pruebas con mesh (cuando el orco aún no había sido dividido en piezas) y en él se crea a un cuadro de 8x8 guerreros orcos. En esta y en las siguientes pruebas las "sentencias de programación" fueron imprescindibles para crear grandes grupos de guerreros.

Estadísticas

Para hacernos una idea cabal de lo conveniente que es el uso de mesh vale la pena citar algunas estadísticas recogidas tras las primeras pruebas con esta sentencia. Las mallas del orco completo tienen muchos triángulos y por ello, tras realizar el primer render para una sola de estas criaturillas, pudimos ver en la pantalla de estadísticas la nada desdeñable cifra de 14.612.203 bytes. ¡Casi 14 Megas de RAM para un solo orco! Sin embargo, al añadir 63 orcos más a la escena, el gasto de memoria tan sólo ascendió a unos 2 Megas más. Por otro lado, para otros modelos menos complejos como el del lancero, el gasto de memoria fue aún menor. A priori resulta imposible predecir la cantidad total de memoria que precisaremos para crear un grupo de objetos-copia (así los llamaremos desde ahora). Sólo podemos estar seguros de una cosa, el ahorro de memoria será impresionante y además podremos manipular a los objetos-copia de maneras sumamente interesantes.



El primer cuadro de orcos.



Lo mismo haciendo variaciones de tamaño.



Añadiendo pigmentación aleatoria, cabezas adicionales, más armas y descolocación aleatoria.

Perfeccionando el regimiento de orcos

En la primera prueba se creó una formación cuadrada de 8x8 orcos. Este cuadro de guerreros no resultaba demasiado atractivo y el mayor problema fue, por supuesto, que todos los orcos eran iguales. Este problema sólo tenía una solución posible: crear más orcos. Pero claro, cada nuevo monstruo hubiera requerido (de emplear la misma densidad poligonal y detalle que en el primero) otros 12 ó 13 Megas de memoria. Además, esto sólo nos hubiera



Resultado final añadiendo un portaestandarte.

dado una variación de 2 tipos de personajes diferentes dentro del cuadro de 64 orcos, con lo cual no hubiéramos conseguido gran cosa. Finalmente, se tomaron las siguientes medidas para mejorar el resultado inicial:

1) Se desglosó el modelo completo del orco en varias partes: el cuerpo, la cabeza y las armas. Luego se alteró el código del fichero escénico de manera que según un factor aleatorio se colocase la cabeza creada en el número 11 o las diseñadas en el 13 con Spatch. Además se hizo lo mismo con las armas y el estandarte (como estos objetos se empuñan de la misma manera y están colocados en la misma posición espacial son fácilmente intercambiables).

2) Los objetos componentes del orco se englobaron dentro de una union, de manera que pudieran variarse las proporciones globales de altura y anchura de cada personaje.

3) Como la alineación del cuadro de guerreros resultaba excesivamente perfecta, se introdujeron pequeñas alteraciones aleatorias de posición en cada orco.

4) Se introdujo un pequeño factor aleatorio de rotación lateral en la cabeza de cada personaje.

5) Se alteraron los ficheros .INC para que el color básico de la piel de cada orco sufriese pequeñas variaciones aleatorias.

El resultado final –visible en la escena final de la serie de pruebas–, aunque aún dista bastante de la perfección, es muy superior al obtenido en la primera imagen de la misma. Sin embargo, lo importante es comprender que cada uno de los modelos que aparecen en la escena final de la serie ha ocupado –a pesar de sus leves variaciones con respecto al modelo original– tan solo una pequeña cantidad de memoria. Viendo esto uno no puede por menos de preguntarse si no será posible conseguir un resultado aún mejor, y lo cierto es que sí que es posible. En el siguiente artículo explicaremos cómo esto puede lograrse siguiendo un curioso y sencillo método. Por ahora echaremos un vistazo en detalle a los trucos que se han empleado para la última escena de la serie.

Trucos de programación

Las llamadas "sentencias de programación" del lenguaje escénico de «POV» son vitales para crear escenas con cientos o miles de modelos. Por ejemplo para crear el cuadro de guerreros se empleó un bucle anidado dentro de otro. El bucle más interno crea los guerreros de cada fila y usa para ello el contador fila. El bucle más exterior se encarga —usando el contador nfilas— de controlar el número de filas que tendrá la formación. Así, por ejemplo, unos valores de 4 y 10 para nfila y fila respectivamente crearían una formación de 4 filas de profundidad con 10 guerreros en cada fila. Pero, por supuesto, podríamos haber dado valores para crear un cuadro de 400 o más orcos. ¿Os imagináis situándolos en la escena a mano? (Gracias al povteam por #while).

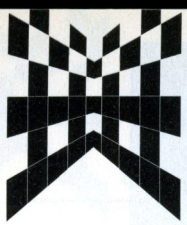
Pero sigamos: el guerrero "original" (el cargado en memoria con #include) está centrado en los ejes X y Z y tiene los pies sobre Y=0. Así que, para desplazar cada objeto-copia a la posición adecuada dentro de la formación, podemos usar una sentencia parecida a esta

```
object{orco1 translate <xpos,0,zpos> }
```

Naturalmente dentro del bucle interno la variable xpos se irá actualizando en cada pasada del mismo con una sentencia como…

```
#declare xpos=xpos+120
```

donde el valor sumado a xpos es la separación espacial en el eje X que el próximo orco tendrá con respecto al último creado en la escena (cuanto más pequeña sea la suma para xpos y zpos, más apretados estarán los guerreros dentro de la formación).



Cómo...

Lo malo de esto es que la formación resultante es excesivamente perfecta. Conseguiremos un resultado más real si los guerreros se salen levemente de la posición ideal de cada uno. Para conseguir esto lo mejor será emplear la sentencia rand para añadir una suma aleatoria a los valores de xpos e zpos. Echad un vistazo al código de la siguiente figura.

"Ejemplo tomado del fichero de generación de una de las escenas de la serie"

```
#declare cuerpo1=object{#include "body1.inc"}
#declare cabeza1=object{#include "head1.inc"}
#declare cabeza2=object{#include "head2.inc"}
#declare cabeza3=object{#include "head3.inc"}
#declare R=seed(8534)
#declare colum=1
#declare xpos=0
#declare zpos=0
#while(colum!=0)
   #declare xpos=0
   #declare fila=8
   #while(fila!=0)
      union{
      object{cuerpo1}
      #declare a=int(rand(R)*3)
      #switch (a)
         #case (0)
         object{cabeza1 rotate <0,-25 +(rand(R)*50),0>}
      #break
         #case (1)
         object{cabeza2 rotate <0,-25 +(rand(R)*50),0>}
      #break
         #case (2)
         object{cabeza3 rotate <0,-25 +(rand(R)*50),0>}
      #end
      #declare alto=.8 + rand(R)*.15
      #declare ancho=.90 + rand(R)*.2
      scale <1*ancho,1*alto,1*ancho>
      #declare sumx=-15 +(rand(R)*30)
      #declare sumz=-15 +(rand(R)*30)
      translate <xpos+sumx,0,zpos+sumz>
      }
      #declare xpos=xpos+120
      #declare fila=fila-1
   #end
   #declare zpos=zpos+120
   #declare colum=colum-1
#end
```

En este ejemplo puede verse cómo se añaden a xpos y zpos los valores sumx y sumz respectivamente. Estos valores representan una variación aleatoria de 30 unidades con respecto a la posición ideal. Para lograrla se restan 15 unidades a dicha posición y se suma el valor aleatorio devuelto por rand. Como recordaréis rand devuelve un valor aleatorio entre 0 y 1 por lo que multiplicar el valor devuelto por 30 equivale a obtener un valor aleatorio entre 0 y 30. Como se resta 15 al valor de estas variables, el resultado es que cada orco puede ser colocado dentro de un rectángulo espacial de 30x30 unidades cuyo centro es su posición ideal. Lógicamente, si cambiamos el valor de la semilla dado a R (en "#declare R=seed(8534)"), el "desorden" obtenido en la formación será diferente.

Otro detalle a tener en cuenta es que podemos variar la escala de un objeto-copia, diferenciándolo así un poco del modelo original. Esta posibilidad ha sido usada para alterar la altura y el grosor de los orcos, logrando así orcos bajos y anchos y otros altos y delgados. Esto se ha conseguido ordenando la siguiente transformación para la union del modelo completo:

```
#declare alto=.8 + rand(R)*.15
#declare ancho=.90 + rand(R)*.2
scale <1*ancho,1*alto,1*ancho>
```

Si exageramos los valores dados a alto y ancho obtendremos una variedad de proporciones aún mayor aunque, por supuesto, seguimos sin poder alterar las formas básicas del modelo. Por eso, el resultado es un cuadro de guerreros orcos que parecen todos miembros de un clan fuertemente endogámico.

Y ya solamente nos queda dar cuenta de dos detalles. Como véis en el ejemplo de la figura, se escoge entre una y otra cabeza dependiendo (otra vez) de un valor aleatorio. En caso de que hubiéramos podido disponer de más cabezas hubiera sido sencillo ampliar el switch para incluirlas así como variar la sentencia que asigna su valor a "a" (nótese el uso de int para forzar a «POV» a devolver sólo valores enteros para el switch).

En cuanto a la pigmentación también se ha usado un valor aleatorio para alterar un poco la intensidad del verde con respecto al color ideal de cada orco. Lo mismo se ha hecho con el color de la faldita y de la cota de mallas. Sobre esto último hay que decir que realmente no hay cota de mallas, sino tan solo un color distinto al verde de la piel, pero desde lejos podemos dar el pego con un simple cambio de color.

Creación de modelos-copia diferentes al modelo original

En las páginas precedentes hemos visto cómo puede emplearse la sentencia mesh para crear copias de modelos en «POV». Como dichas copias requieren tan sólo una pequeña cantidad de memoria –en contraste con la gran cantidad de RAM que puede requerir el modelo original–, esto quiere decir que podremos crear escenas con cientos o miles de formas complejas. Sin embargo, esto no es todo lo que podemos hacer: existe una manera sencilla de crear modelos-copia y darles posturas diferentes a la original. Es decir, si tenemos el tiempo suficiente, podemos crear docenas de posturas de un modelo para una escena y emplearlas sin que cada nueva postura precise más espacio del que requeriría un objeto-copia idéntico al modelo original.



Antes de empezar conviene aclarar que nuestro orco no es el modelo más adecuado para ilustrar la técnica que vamos a describir en el presente artículo. Ello se debe a que el cuerpo del modelo carece de objetos que sirvan como ejes de articulación. Por esta razón, en muchas de las posturas pueden apreciarse huecos en las rodillas, los codos o los hombros, con lo que queda de manifiesto que el cuerpo del orco no es un modelo sólido sino tan solo un conjunto de formas huecas. Este problema ha quedado sin solución por ahora, ya que no había tiempo para construir las formas necesarias e incorporarlas al modelo (bueno, lo cierto es que hemos añadido unas esferas en las rodillas ya que los huecos que aparecían eran demasiado reveladores, pero esto no pasa de ser un apaño provisional).

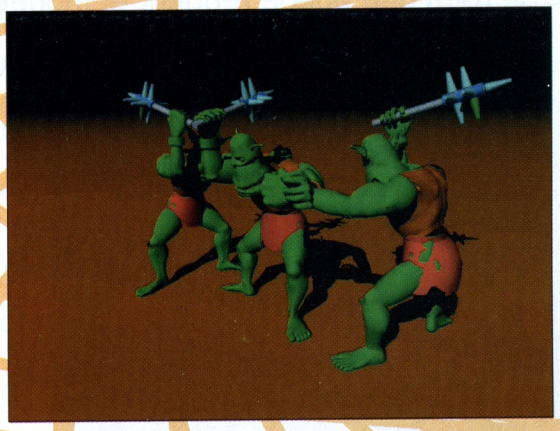
Sin embargo, lo realmente importante para nuestros propósitos era demostrar la validez del método que vamos a presentar a continuación, y para esto el orco resultaba mucho más conveniente que, por ejemplo, un mech.

La idea básica

En el artículo anterior las últimas escenas fueron creadas dividiendo el modelo de orco en tres partes: el cuerpo completo, las cabezas y las armas. De esta manera, cada una de estas partes era fácilmente intercambiable, lo cual daba algo de variedad a las escenas ¡¡lástima no haber tenido tiempo suficiente para preparar más cuerpos!!

Hecho esto, nos preguntamos si no sería aún mejor almacenar cada parte del cuerpo separadamente, a fin de poder aplicar cambios individuales en cada pieza. ¿Y qué tipo de cambios? Pues no, evidentemente, cambios anatómicos, puesto que entonces las proporciones del modelo habrían quedado estropeadas, pero sí cambios de orientación y de colocación. Así, como cada parte estaría almacenada usando mesh y como cada objeto-copia podría ser transformado independientemente, el resultado sería que podríamos crear orcos-copia con posturas diferentes a la del conjunto de piezas del orco original.

El que esto funcione bien o no dependerá, como en el caso del artículo anterior, de que estemos empleando una herramienta de traducción que respete la colocación que tenían los objetos en el espacio virtual del modelador donde fueron creados. Decimos esto porque hay algunos traductores 3D que olvidan dicha colocación y que además reescalan los objetos a su capricho al hacer la "traducción", con lo cual no nos servirán para nada. En el caso del orco, este fue modificado con «Rhinoceros», programa desde el cual se exportaron todos los objetos al formato de «POV». Luego se comprobó, al crear una escena de prueba desde «POV», que «Rhino» había respetado la escala, colocación y orientación de cada pieza al hacer la grabación, por lo cual el resultado fue un orco completo.





Preparando posturas para el orco.

Ahora bien; ¿cómo crear las posturas? Como imaginaréis, crear cada postura desde «POV» estableciendo los valores de rotación de cada objeto es algo muy trabajoso (al menos al principio, hasta que se acaba cogiendo práctica. Recordad que algunos esforzados rendermaníacos lo han hecho. Este es el caso de Óscar Alvárez Díaz, con su soldado imperial de asalto, y de Carlos Monterde Escude-



El ejército orco en formación de batalla.

ro, con su horda orca). Por esta razón, y después de exprimirse un rato las neuronas, el autor de estas líneas ideó el siguiente método para crear cada nueva postura:

1) «Imagine» es el programa donde más fácil y rápidamente podemos crear nuevas posturas de un modelo antropomórfico. Para aprovecharnos de esta útil característica de «Imagine» cargaremos desde este programa el modelo cuyas posturas deseemos crear.

Para ello deberemos grabar el modelo desde el modelador que estemos empleando y traducirlo luego a un formato que pueda ser leído desde «Imagine» (al suyo propio o al DXF). Otra posibilidad es, simplemente, emplear un modelo antropomórfico similar al del orco ya que lo que nos interesa es únicamente tomar nota de los ángulos de rotación de las distintas partes del cuerpo. Esto último fue lo que se hizo (para evitarnos el trabajo de la conversión a «Imagine») y empleamos para esto al modelo del no-muerto que apareció en Rendermanía número 9, usando la postura básica del mismo que es igual a la básica del orco (o sea, de pie y en postura de firmes salvo por las manos cerradas en puños). Esto sólo tiene un inconveniente: puede darse el caso que tengamos que hacer ajustes desde «POV» dadas las posibles diferencias entre el modelo antropomórfico empleado en «Imagine» y el modelo usado en «POV».

2) Desde «Rhinoceros» o «Imagine», o desde el programa que hayamos empleado para crear nuestro modelo, tomaremos nota de la posición de cada eje local de giro. Llamaremos eje local al punto espacial donde pivota una pieza corporal o un conjunto de piezas. Por ejemplo, habrá que definir un eje local para la mano (situándolo en la muñeca), otro para el antebrazo (situándolo a la altura del centro del codo) y otro para el brazo completo (situándolo en el centro del hombro). Así pues tomaremos nota de los valores <x,y,z> de la posición de cada eje local. Atención: si establecemos mal la posición de estos ejes, luego desde «POV» pueden sucedernos cosas tales como que un brazo o una pierna se salgan del cuerpo, al intentar definir su posición.

3) Hecho esto crearemos las jerarquías del modelo –si es que no estaban ya creadas– y procederemos a crear la postura deseada tomando nota del valor del ángulo para cada pieza y del eje en que dicha rotación va a llevarse a cabo. Aquí, naturalmente, lo mejor será que nos hayamos molestado en disponer previamente al modelo con una orientación tal que los ejes de rotación correspondan a la que normalmente tienen nuestros modelos en «POV» (recordad que aquí solemos exportar los modelos a «POV» de tal manera que el personaje quede de pie sobre el plano X-Z, con los pies a la altura de Y=0 y con los ojos mirando en la dirección -Z. Esta es la orientación de nuestro orco y también la del zombi, por lo cual no fue preciso reorientar a éste en «Imagine»). Hecho esto iremos rotando las distintas partes del modelo para crear la postura deseada y, al mismo tiempo, tomaremos nota de los grados (visibles en la esquina superior derecha de la pantalla de «Imagine») de cada rotación. También hay que recordar estipular en nuestras notas dónde se efectúa la rotación ya que si ésta se lleva a cabo en el hombro, por ejemplo, habrá de afectar al brazo completo.

4) El último paso será crear uno o más ficheros plantilla usando el lenguaje escénico de «POV» (luego veremos cómo). En nuestro caso, un proyecto que describa una escena en la que participen regimientos de orcos se compone al menos de 3 tipos de ficheros: uno con la extensión .«POV» que describe la escena e invoca a los demás archivos, otros que incluyen los valores de rotación de cada estructura jerárquica del modelo y otro donde se crea una jerarquía de uniones que imita a las jerarquías del modelo en «Imagine» (o a las del modelo usado en su lugar desde este programa). Aunque, por supuesto, esto podría hacerse siguiendo otro sistema.

Otro detalle importante que conviene no olvidar es que muchas utilidades de traducción incluyen en las mallas

traducidas una textura similar a la que tenía el modelo en el programa donde fue creado. Pues bien, si queremos realizar ciertos trucos como una variación aleatoria de color en la piel o cambiar los colores de otras partes del modelo, lo mejor será eliminar las texturas traducidas. De esta manera, las piezas asumirán la textura indicada en orco.inc, la cual puede ser distinta a la usada en el modelo-copia anterior, puesto que dicha textura puede ser alterada en el fichero de escena.

Simulación de jerarquías en «POV»

Hemos dicho en las líneas anteriores que hay que preparar un archivo que establezca un sistema de jerarquías en «POV» que imite al que tendría el mismo modelo en «Imagine». De esta manera, podremos hacer girar por ejemplo una mano o un brazo completo, según lo que nos propongamos. Quizá el lector se extrañe ante esto y se pregunte si verdaderamente es posible y la respuesta es que sí que lo es, si seguimos unas reglas bien sencillas.

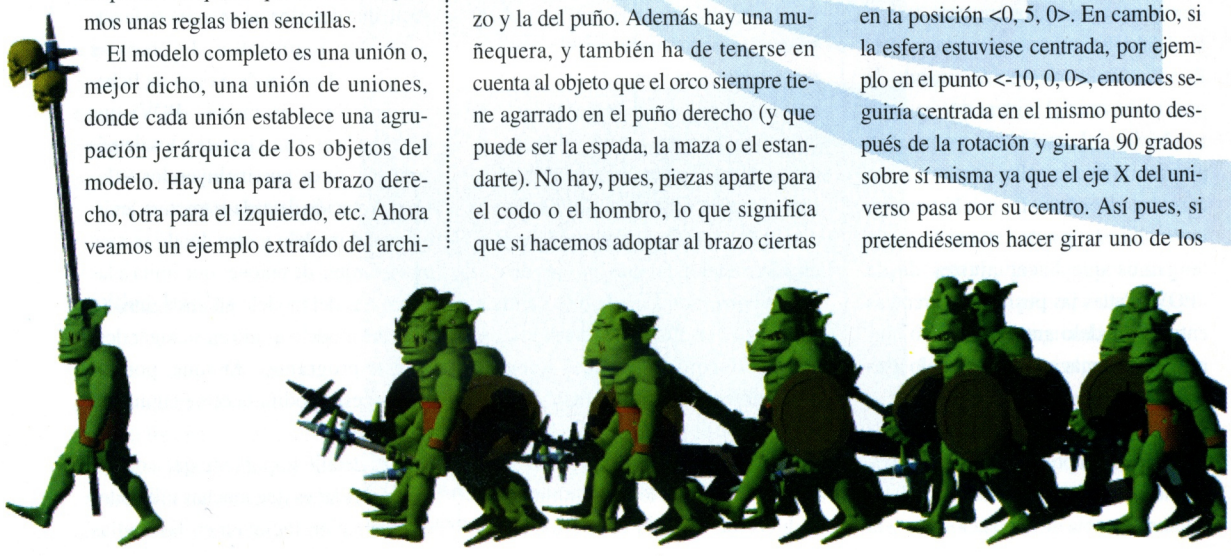
El modelo completo es una unión o, mejor dicho, una unión de uniones, donde cada unión establece una agrupación jerárquica de los objetos del modelo. Hay una para el brazo derecho, otra para el izquierdo, etc. Ahora veamos un ejemplo extraído del archivo orco.inc, donde hemos definido las relaciones jerárquicas de nuestro orco:

Ejemplo de relación jerárquica para el brazo derecho del orco

```
...
...
//brazo derecho
      unión{
        unión{object{antebd}
              object{munned}
//la mano y su arma cogen también el giro del antebrazo
              unión{object{manod} object{arma}
                translate<39.5,—75,—2.5>
                        rotate <rmanodX, rmanodY, rmanodZ>
                        translate<—39.5,75,2.5>
                        }
              translate<38,—104,—12>
              rotate <rantebdX, rantebdY, rantebdZ>
              translate<—38,104,12>
             }
        object{brazod}
        translate<25,—136,—9>
        rotate <rbrazodX, rbrazodY, rbrazodZ>
        translate<—25,136,9>
        }
...
```

En este ejemplo se define la jerarquía para el brazo derecho de nuestro orco. Este brazo está compuesto por las siguientes piezas: la que define la forma del hombro y biceps, la del antebrazo y la del puño. Además hay una muñequera, y también ha de tenerse en cuenta al objeto que el orco siempre tiene agarrado en el puño derecho (y que puede ser la espada, la maza o el estandarte). No hay, pues, piezas aparte para el codo o el hombro, lo que significa que si hacemos adoptar al brazo ciertas posturas algo forzadas comprobaremos que las formas del brazo están huecas. Lo mismo es válido para el resto de los miembros del modelo.

Ahora veamos cómo funciona todo esto. Como recordarán los «POV»adeptos, los objetos en «POV» giran en torno al eje en que se establece la rotación y los tres ejes, X, Y y Z, se cruzan en el centro del universo virtual de «POV», en el punto <0, 0, 0>. Así pues, si por ejemplo creamos una esfera centrada en la posición <0, 0, -5> y le efectuamos una rotación de 90 grados en el eje X, entonces la esfera girará en torno a este eje quedando centrada en la posición <0, 5, 0>. En cambio, si la esfera estuviese centrada, por ejemplo en el punto <-10, 0, 0>, entonces seguiría centrada en el mismo punto después de la rotación y giraría 90 grados sobre sí misma ya que el eje X del universo pasa por su centro. Así pues, si pretendiésemos hacer girar uno de los





brazos del orco, no bastaría con indicar la operación de rotación, sino que antes deberíamos trasladar el brazo entero de manera que su eje local coincidiese con el centro de coordenadas <0, 0, 0>. Si no obrásemos así, el brazo se saldría del hombro al efectuarse la rotación.

Por tanto, y siguiendo con el mismo ejemplo, para girar el brazo (o cualquier otro objeto) tendremos que seguir tres pasos: trasladar el brazo de modo que su eje local coincida con el eje del universo en torno al cual queremos girar, efectuar la rotación y por último aplicar al brazo una traslación inversa a la primera, de manera que el brazo vuelva a quedar encajado en el hombro.

Todo esto no sería preciso si «POV» dispusiera de una instrucción para efectuar rotaciones en torno a un eje local, pero dicha instrucción no existe (o al menos es desconocida por nosotros). Ahora bien; ¿qué valores daremos a las operaciones de traslación que enmarcan a cada operación de rotación?

Evidentemente, los valores que correspondan a los puntos donde hemos determinado que se hallen los ejes locales (recordad que previamente deberemos haber establecido desde el modelador los ejes locales de todas las piezas que puedan girar; en nuestro orco se tomó nota de dichos ejes desde «Rhino»). Por ejemplo, hemos decidido que el eje local del brazo derecho completo de nuestro orco esté en la posición <25, -136, -9> (este punto está localizado, a ojo, en el centro del hombro de nuestro modelo). Por tanto, para girar el brazo derecho de nuestro orco, hemos escrito las siguientes sentencias:

```
translate<25,—136,—9>
rotate <rbrazodX, rbrazodY, rbrazodZ>
translate<—25,136,9>
```

Probablemente tanta prolijidad os resultará molesta pero hay que asegurarse de que estos puntos quedan claros. Las variables rbrazodX, rbrazodY y rbrazodZ son las que especifican la cantidad de grados que el brazo va a girar en el eje local declarado. Hay variables con nombres parecidos para cada articulación y los valores de rotación para ellas se establecen en otro fichero donde, gracias a dichos valores, se indica la postura que deseamos para el orco. Y con esto ya hemos explicado cómo giran los miembros del cuerpo del orco.

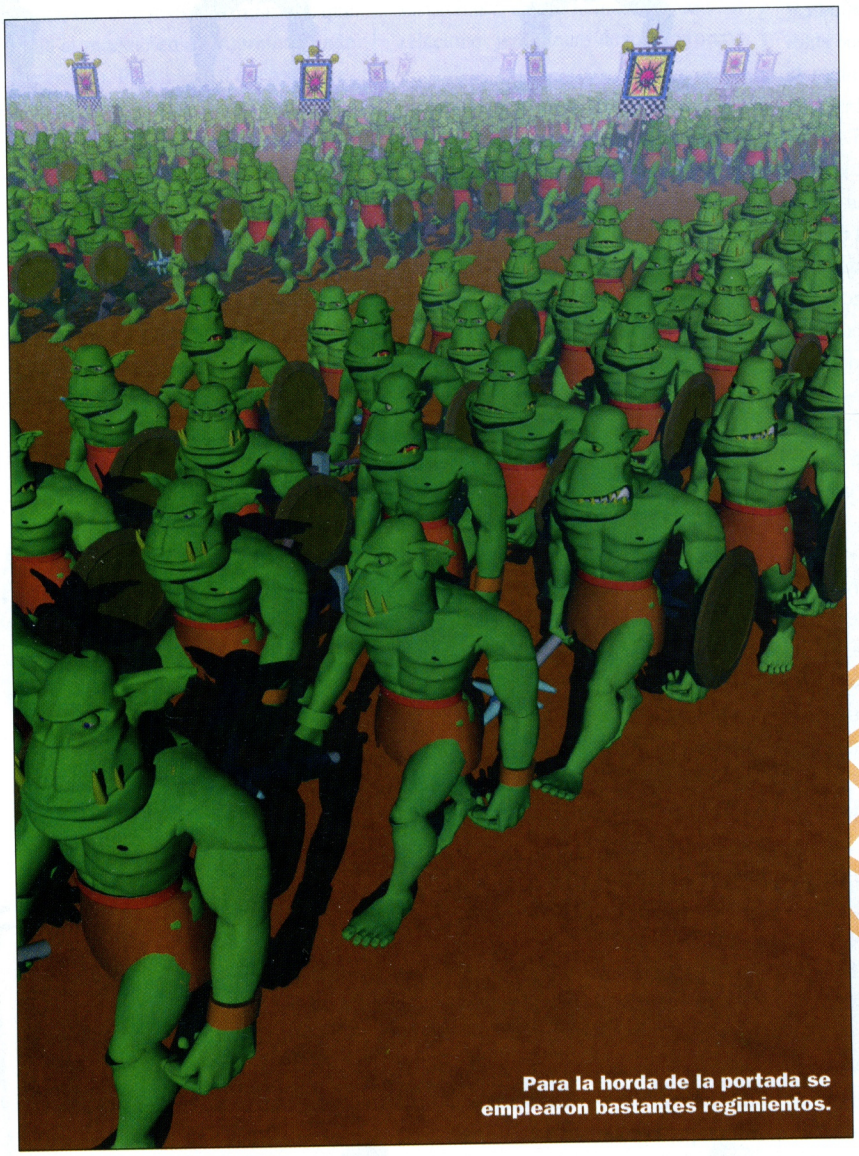
Ahora pasaremos al siguiente tema: ¿Cómo podemos simular jerarquías en «POV»?

Echad un nuevo vistazo al listado de ejemplo anterior. En la unión más interna están los objetos manod y arma, los cuales giraran en torno al eje local definido para la mano (si la mano derecha del orco no empuñase un arma no sería preciso declarar una unión aquí, y bastaría con incluir las transformaciones dentro del objeto manod). Luego, siguiendo hacia afuera, veremos que la unión manod–arma forma parte de otra unión compuesta también por los objetos antebd (antebrazo derecho) y munned (muñequera derecha). Al final de la unión encontraremos las sentencias de traslación y rotación que afectarán a todo el antebrazo permitiéndole girar en torno al eje local situado a la altura del codo. Esta unión forma parte de otra, más exterior, de la que el objeto brazod es el otro componente –hemos llamado brazod a esta pieza para indicar que su giro afecta al brazo completo, aunque el objeto en sí solo se refiere a la parte superior del brazo. Esta zona incluye al hombro y acaba a la altura del codo–. Como ya imaginaréis, las sentencias de transformación para esta unión afectan a todo el brazo.



Otra prueba de horda cambiando la semilla de rand.

Cuando «POV» vaya a procesar el trozo de código del brazo completo, comenzará a trabajar con los objetos o uniones más internos, efectuando primero las rotaciones para la mano y el arma asida por ésta (en caso de que hayamos dado valores a las variables de esta zona). Luego, se pro-



Para la horda de la portada se emplearon bastantes regimientos.

cesarán las rotaciones para el antebrazo completo y finalmente se ejecutarán las del brazo entero. Por tanto, cuando vayamos a crear un archivo-plantilla para otro modelo antropomórfico, deberemos tener presente que lo más sencillo es escribir el conjunto desde dentro hacia afuera, correspondiendo las piezas de la unión más interna a las más extremas del miembro tratado.

Para la pierna derecha, por ejemplo, el orden sería pie, pantorrilla-pie, pierna-pantorrilla-pie, siendo el pie el objeto más interno de la estructura (echad un vistazo a orco.inc). Todo esto es aplicable a todas las piezas del modelo completo y la única dificultad real estará en decidir sobre la jerarquía de ciertas piezas (y por tanto sobre su colocación dentro del archivo).

Es fácil ver que la mano tiene un nivel jerárquico más interno que el antebrazo pero... ¿dónde situaremos la pelvis o la cabeza? En el caso de nuestro orco hemos ordenado esto de manera que hay varias uniones con el mismo nivel jerárquico. Los brazos (enteros) y la cabeza, por ejemplo, tienen el mismo nivel jerárquico y ambos forman parte de la unión que incluye al tórax y que tiene su eje local a la altura del cinturón del modelo. Luego hay otra unión del mismo nivel que la anterior y que incluye a las piernas (enteras), la falda, la pelvis y el cinturón. Ambas uniones principales están englobadas dentro de la unión principal (el cuerpo entero) cuyas operaciones afectan a todo el conjunto.

De esta manera, un valor dado, por ejemplo, a rantebdZ hará doblarse el brazo mientras que otro dado a rtoraxZ hará doblarse a todo el orco de cintura para arriba. Y con esto, ya deberíais estar en condiciones de crear vuestras propias plantillas jerárquicas para tratar a otros modelos, ya sean antropomórficos o no.

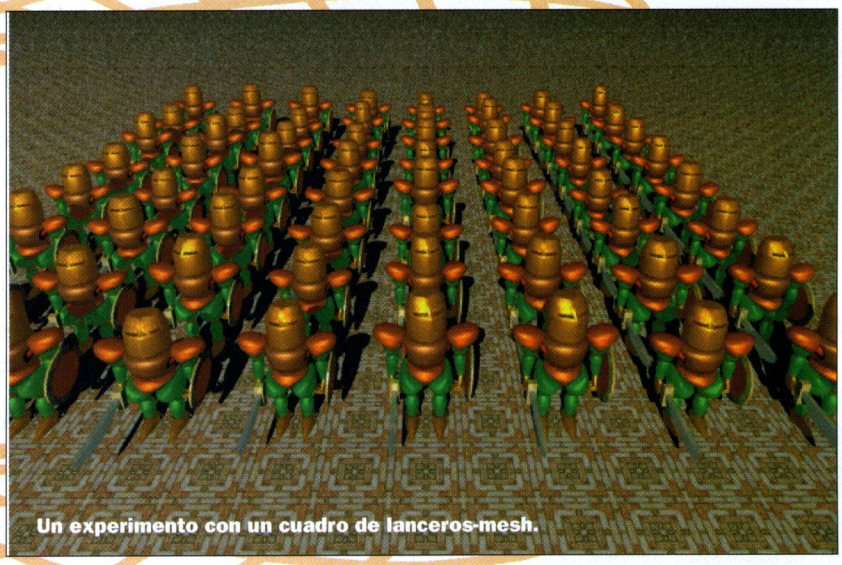
Creación de ficheros-postura

Llamaremos ficheros-postura a los archivos donde se dan valores a las variables de rotación que definen las posturas de los modelos-copia. Algunos de estos ficheros son orcanda1, orcombat, orcasca1, etc. Los números impares (cuando los hay) corresponden a posturas nuevas y los pares a posturas donde se han intercambiado los valores de rotación de los brazos y piernas. Estudiadlos y empleadlos como plantilla para crear los vuestros propios.

Una vez que se tiene práctica, pueden crearse posturas directamente usando un editor de textos. Si ya estáis pensando en preparar vuestros propios modelos usando este sistema, tened presente que convendrá comprobar desde «POV» si cada nueva postura es correcta. Para ello lo mejor será generar una escena con al menos tres copias de la nueva postura y darles distintas orientaciones con respecto a la cámara. Esto es especialmente recomendable si hemos estado utilizando en «Imagine» otro modelo antropomórfico –en vez del original– como en el presente caso.

Finalmente comentaremos una última cosa. Si el lector examina la escena del ejército orco avanzando y su fichero de generación correspondiente (marcha.«POV»), se dará cuenta de un curioso detalle: la escena sólo utiliza dos posturas, orcanda1 y orcanda2, pero todos los orcos parecen algo diferentes entre sí; unos tienen las piernas más levantadas o las rodillas más dobladas que otros, los hay que adelantan más el brazo, etc. Esto se debe sencillamente a que hemos incorporado en los dos ficheros-postura una pequeña variación aleatoria para algunas de las jerarquías del orco. En efecto, cada vez que el fichero de escena incluye a uno de los archivos orcanda, éstos llaman repetidas veces a un ficherito-función creado por nosotros donde se da un valor aleatorio a la variable "variacion", la cual es luego sumada a las variables de rotación. Antes de llamar a este ficherito de nombre funcal.pvm, hay que dar un valor a la variable-parámetro "signo". Normalmente, pondremos el valor 255 en esta variable, indicando así que "variacion" puede ser positiva o negativa, pero en ciertos casos, como por ejemplo en el de una pierna, no nos interesará que la variación tenga un sentido determinado, ya que entonces podría suceder que el miembro adoptase una postura anatómicamente indeseada. En estos casos pondremos a "signo" con el valor +1 o -1 según el sentido del giro que deseemos impedir. Pero esto no es todo.

Desde el fichero escénico daremos también un valor a la variable "varirand", la cual es usada en funcal.pvm para establecer el rango de variaciones que deseamos para "aleatorizar" los grados originales de la postura (en marcha.pov hemos usado el valor 30. Lógicamente cuanto mayor sea este valor, mayor será el rango de variaciones aleatorias para las posturas originales). Podemos aplicar este truco a todas nuestras posturas para conseguir así rehuir un poco la impresión de que la escena está compuesta sólo por unas pocas posturas pero, atención, no podemos usar esta idea siempre. En las piernas, sobre todo, corremos el riesgo de que éstas se hundan en el suelo si permitimos cualquier variación. Otra posibilidad de fallo es que uno de los brazos del orco se hunda en su propio cuerpo así que…¡cuidado!



Un experimento con un cuadro de lanceros-mesh.



Una escena cualquiera con cientos de actores

Para visualizar mejor cómo funciona todo esto vamos a repasar cómo se genera una cualquiera de estas escenas. Por ejemplo, en marcha.pov, se especifica primero la pigmentación "ideal" de los orcos. Luego se incluyen todas las piezas que pueden aparecer en la escena (del cuerpo, las cabezas, las armas...). Después, asignamos valores a una serie de variables que se emplearán en ficheros posteriores y luego se entra en un bucle anidado con el que crearemos un ejército construyendo bloques de regimientos. Cada regimiento será creado al invocar al fichero regiorc.inc, archivo que utilizará las variables anteriores para decidir el ancho del regimiento, su número de filas, si llevará un portaestandarte al frente, etc. La separación entre regimientos y otros detalles también se especificarán antes del bucle.



Una vez en regiorc.inc, «POV» hallará otro doble bucle con el que construirá el regimiento. En cada caso se decidirá si el orco llevará escudo, su tipo de arma, su inclinación lateral de cabeza, etc. Pero lo más importante es que –según el valor de la variable tipoej– se decidirán cuáles serán los ficheros–postura a emplear. Si tipoej vale 1, se creará al ejército usando las posturas de los archivos orcanda, con lo cual obtendremos un ejército marchando. En cambio, si tipoej tiene otro valor, entonces se usará el archivo orcombat.inc, el cual creará poses de combate. En este último caso obtendremos un ejército a punto de entrar en batalla, con nuestros personajes exhibiéndose en diferentes posturas de provocación hacia el enemigo. Puede parecer mentira el que todos los horcos de esta horda hayan sido creados tan sólo con este único archivo, pero en realidad el truco es muy simple.

Ante todo se crearon dentro de este archivo una serie de poses individuales para los miembros del orco. Hay tres posibles colocaciones para el brazo derecho, dos para el brazo izquierdo y una (por ahora) para las piernas. Además de esto cada una de estas posibles colocaciones puede sufrir variaciones aleatorias dentro de ciertos rangos calculados para impedir que el personaje se atraviese con su propia arma (pues, sí, esto sucedió en las pruebas iniciales). En resumen: en cada pasada de orcombat.inc se establece aleatoriamente el brazo que va a colocarse y las variaciones aleatorias correspondientes. El resultado se guarda en las variables de rotación que más tarde se pasarán a orco.inc desde regiorc.inc.

Todo esto puede parecer muy complicado al principio pero desengañaos; no lo es. Lo realmente importante es que entendáis cómo pueden crearse jerarquías simuladas en «POV». Lo demás (la creación de posturas, regimientos o ejércitos enteros) podéis hacerlo si lo preferís de cualquier otra manera.

Futuras escenas con mesh

Visto esto es fácil imaginar muchas posibles escenas interesantes que son posibles con mesh. Con esta sentencia ya no sería imposible crear ciudades gigantescas o grandes flotas espaciales o incluso algo como un hormiguero.

Como mesh puede ser usada en conjunción con las llamadas sentencias de programación, las posibilidades son casi ilimitadas. Tan sólo vemos un problema: aún no hay manera de determinar –utilizando colocaciones aleatorias– si dos modelos se están superponiendo. Esto puede ser bastante molesto (en las primeras pruebas muchos de los orcos se ensartaban unos a otros). Contra esto, por ahora, la única solución es controlar el espacio de separación de cada elemento, usar bucles de colocación que eviten que se trate dos veces una misma zona espacial y esperar que haya suerte.

ACERCA DE SPATCH

El mes pasado incluimos en el CD-ROM un programa llamado «sPatch» que se distribuía gratuitamente a través de la red Internet y que, según lo que podía leerse en el sitio Web del autor, nunca caducaba. A pesar de esta afirmación, «sPatch» ha caducado inesperadamente con el año 97, así que incluimos ahora la nueva versión. Dada la imprecisión de la versión anterior, no podemos asegurar que ésta funcione cuando se publique la revista. Para más información os remitimos al Web del autor:

http://www.aimnet.com/~clifton/spatch.html

El Foro del Lector

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

Mundos Fascinantes

Como bien dice Jesús Angel Lanza Salcines, aunque sepamos muy bien que el corazón de nuestro ordenador es un frío chip de silicio, al otro lado de la pantalla de nuestros monitores pueden existir mundos fascinantes que tan sólo esperan a que nosotros sepamos hacerlos realidad. En ese otro lado puede haber poesía, belleza, fantasía e incluso horror. Lo que hallemos dependerá, como siempre, de nuestro esfuerzo y de la riqueza de nuestro propio mundo interior. Y es que algunos autores (volviendo a citar a Jesús) prefieren considerar a la pantalla del ordenador no como frío hardware, sino como la puerta a un mundo mágico.



Probablemente cuando echéis un vistazo a las maravillosas escenas de **Jesús Angel Lanza Salcines** os quedaréis tan asombrados como yo mismo. Y es que las escenas de este maestro del «Caligari Truespace» no son simples imágenes renderizadas. Más bien parecen puertas a universos donde se están desarrollando historias como las que cuentan Tolkien, Zelazny, Moorcock y otros maestros de la fantasía. Son escenas soberbias, fantásticas, alucinantes, donde las chicas de Jesús no parecen mallas poligonales sino personajes vivos y adorables.



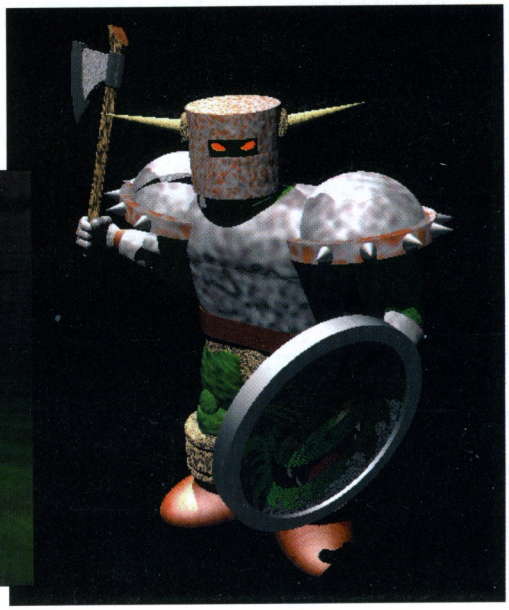
El Foro del Lector

Seguimos hablando de la obra de **Jesus Angel Lanza Salcines.** En cuanto a los datos técnicos, poco es lo que podemos decir sobre la chica. Los datos incluidos por Jesús dicen que Pamela tiene unas 6200 caras (cuesta creerlo, ¿verdad?). Y tampoco podemos explicar nada sobre el propio autor, ya que no nos ha enviado carta alguna. Tan sólo contamos con unos interesantes archivos html, donde podremos ver escenas y animaciones y leer algunos divertidos textos relacionados con las imágenes. Podéis encontrar todo esto –junto con otras imágenes que venían en otra carta y que no son referenciadas en los archivos html– en el directorio lanza del CD. Amigo Jesús, tu chica es extraordinaria y podría en mi opinión competir incluso contra Kyoko o Rina (otras dos fantásticas chicas 3D que ya vimos en el número 8 de Rendermanía). Estoy seguro de que todos los rendermaníacos desearán tanto volver a verte en estas páginas como yo. Hasta entonces dile a Pamela que me considere su más ferviente admirador.





Como podéis ver en esta página, el orco de **Carlos Monterde Escudero** ha progresado mucho. Las manos están muy mejoradas con respecto a las del modelo del número 11 y Carlos ha preparado bastantes posturas de este orco creado con el lenguaje de «POV». Además, Carlos incluye un brujo (también completamente modelado desde «POV») y ha preparado nuevas armas para su orco. Bien por ti pov-colega, pero te recomiendo que utilices el método que expongo en el presente número. ¡Tardarás menos en crear posturas y grupos de personajes! Por otro lado el hermano de Carlos, CESAR, nos ha enviado una pequeña animación de la que espera una crítica que por una vez no voy a hacer (la dejaré para tu siguiente animación). En cuanto al reto de vuestro clan de orcos, puede ser que decida aceptarlo y quizá, quizá, os llevéis una sorpresa.







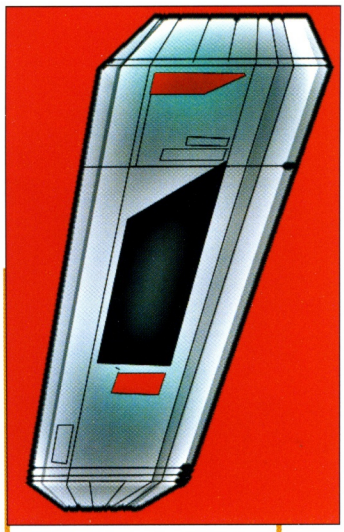
Otro par de hermanos: **Alberto y Oscar Otero Leal** nos envían un par de mechs creados con «MAX». Ambos hermanos incluyen una carta explicando los detalles de la construcción de sus modelos y las características técnicas de cada mech y me piden que diga cuál de los dos me ha gustado más (y además aclaran que no aceptarán un empate). Ambos hermanos aducen las diferentes ventajas de sus mechs y Oscar, por ejemplo, afirma que cree al suyo capaz de cualquier misión. ¡Pues bien, no lo creo! Dudo que, por ejemplo, fuese capaz de tomar el té en el salón de mi casa y por dicha razón le descalifico y doy la victoria al de Alberto. Bueno, ahora en serio, os habéis olvidado de especificar qué imagen corresponde a cada mech aunque creo que me gusta más el blanco ya que su forma me parece mas original que la del negro. Por supuesto que aceptaré a vuestros modelos para la escena de mechs, que como veis se está retrasando un poco (problemas de conversiones).



Luis Pérez Monpeán ataca de nuevo con dos imágenes creadas con «POV». Esta vez se trata de una habitación y de una réplica aproximada del escenario empleado por U2 para su gira "Pop Mart". Pues bien, ya que pides una crítica te diré que la pega principal es que tus imágenes podrían haber ganado bastante si hubieras usado esas sentencias de programación de «POV» que aún no has tenido tiempo de aprender. No temas: no voy a enviarte un Mech trazado con rayos a que te tire de las orejas para que las estudies, pero sí quiero hacerte notar que si las hubieras usado probablemente tus escenas habrían ganado mucho con muy poco esfuerzo adicional. Por ejemplo, habrías podido emplear #while para las vallas del estadio o para dar algo de detalle a las gradas del mismo e incluso para el mismo armario. Tomo nota de tu sugerencia para la ciudad-portada pero antes habría que especificar algunas reglas de construcción ¿no? A fin de cuentas un chalet o un templo romano no pegan en medio de una ciudad de rascacielos moderna. Hasta la próxima.



Francisco J. Serrano Rey nos envía una imagen de su tanque escorpión que desea vender y solicita una opinión. Bueno pues, sin ánimo de desanimarte y sin decir en modo alguno que tu modelo sea malo, he de señalarte que incluso los mejores infografistas lo tendrían difícil para vender un único modelo. Además, tu tanque está restringido a un campo muy limitado (un buen modelo humano lo tendría algo más fácil). Mi consejo es que, si tienes algo visualizable de ese videojuego estratégico del que hablas, te acerques a alguna casa de soft. Otra posibilidad es trabajar como infografista en alguna casa de juegos. Pero mientras buscas, mi consejo es que sigas practicando, ya que sólo cuando se alcanza un nivel de habilidad verdaderamente espectacular hay esperanzas de lograr un trabajo en este mundillo. ¡Suerte!



Benjamin Albares Moreno recibe mi más sentido pésame por la disolución de tu grupo y también mi más sincera felicitación por tu estupendo Mech. Me alegra tu intención de participar en la convocatoria de la portada sobre Mechs (que como ves, se va a retrasar un poco) pero lo que dices es bien cierto; tu mech emplea muchas texturas que yo habría de aplicar pieza a pieza desde «POV» ya que no conozco ningún conversor que "traduzca" también la aplicación de las texturas. ¿Qué hacer? Aún no lo sé. En cuanto a tu sugerencia para el collage, desde luego es lo que menos trabajo me costaría pero la verdad es que prefiero preparar una portada normal, aunque no voy a descartar tu idea.

Por último quiero felicitaros a ti y a David Lozano Lucas por vuestra revista electrónica "Euro 3d Design" (leed los detalles en la carta de Benyi). Para ella, como dice Spock; ¡Larga y próspera vida!



A&SP STUDIO

VOLCADO DE ANIMACIONES • PRODUCCION Y POSTPRODUCCION DIGITAL DE VIDEO

¿Te gustan tus trabajos 2D/3D?



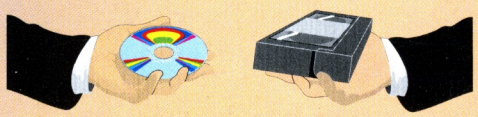
Entonces, ¿por qué los terminas en formato AVI, FLI, o con tarjetas MPEG de baja calidad, cuando puedes obtener resultados verdaderamente profesionales a un coste razonable?

Si eres usuario de AutoCAD, 3D Studio, Animator Pro, LightWave, POVRay, etc. y quieres tener tus propias animaciones volcadas a vídeo en calidad Broadcast Betacam, así como toda una serie de servicios adicionales, llámanos y te informaremos.

¿Te imaginas?

Poder ver en video cualquiera de tus animaciones.

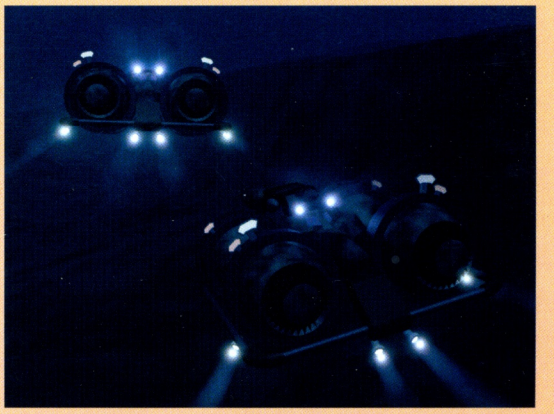
¡Increible!



No lo pienses

LLamanos

571 3142



Fax : 571 3501
E-mail : a-sp@infornet.es
Web: http://www.a-sp.com
C/Ávila 11, 1º E 28020 MADRID

Recogida y envío de trabajos a toda España concertada con SEUR Servicio Urgente de Transportes