Tutorial de POV-Ray (II)

Por Eric Albrecht

En el número anterior aprendimos a crear un render POV-Ray, comenzando con un archivo LDraw creado en cualquier editor, abriéndolo en LDView, y exportándolo a POV-Ray. También aprendimos cómo decirle a LDView que utilice automáticamente la biblioteca LGEO de piezas de POV-Ray para crear mejores renders. Una vez que todo esto ha sido establecido según el último tutorial, todo funciona sin demasiada intervención por parte del usuario. Abrir, exportar, abrir, renderizar. Si queremos llegar más lejos, sin embargo, vamos a tener que hacer algunas cosas manualmente y eso significa comprender lo que realmente está dentro del archivo POV-Ray. Vamos a empezar con el sencillo archivo que creamos la última vez, que contiene un único ladrillo rojo. Si no tienes ese archivo, simplemente exporta cualquier cosa desde LDView para empezar.

```
// Generated By: LDView 4.2 Beta 1 Copyright (C) 2008 Travis Cobbs & Peter Bartfai
// See: http://ldview.sourceforge.net/
// Date: Sun Mar 03 19:50:43 2013
// Input LDraw File: brick.ldr
//
// This file was automatically generated from an LDraw file by the program
// LDView.  See comments above for further details.

#declare LDXQual = 3;   // Quality (0 = Bounding Box; 1 = No Refraction; 2 = Normal; 3 = Stud Logos)
#declare LDXSW = 0.5;   // Seam Width (0 for no seams)
#declare LDXStuds = 1;  // Show studs? (1 = YES; 0 = NO)
#declare LDXRefls = 1;  // Reflections? (1 = YES; 0 = NO)
#declare LDXShads = 1;  // Shadows? (1 = YES; 0 = NO)
#declare LDXFloor = 1;  // Include Floor? (1 = YES; 0 = NO)

// Model bounds information
#declare LDXMinX = -40;
#declare LDXMinY = -4;
#declare LDXMinZ = -20;
#declare LDXMaxX = 40;
#declare LDXMaxY = 24;
#declare LDXMaxZ = 20;
#declare LDXCenterX = 0;
#declare LDXCenterY = 10;
#declare LDXCenterZ = 0;
#declare LDXCenter = <LDXCenterX,LDXCenterY,LDXCenterZ>;
#declare LDXRadius = 46.8615;

// Camera settings
#declare LDXCameraLoc = < 856.136169,-357.441711,-534.972961 >; // Camera Location vector
#declare LDXCameraLookAt = < 2.6927083672914023,8.8443190579598649,-1.682618580794383 >;  //Camera look-at point vector
#declare LDXCameraSky = < -0.29004934789986447,-0.93969269365205865,0.18124297856689955 >; //Camera sky vector(<0,-1,0> will

#declare LDXFloorLoc = LDXMaxY; // Floor location. (Dependent on floor axis; MAX_Y is bottom of model)
#declare LDXFloorAxis = y;      // Floor axis (x, y, or z)
#declare LDXFloorR = 0.8;       // Floor Red
#declare LDXFloorG = 0.8;       // Floor Green
#declare LDXFloorB = 0.8;       // Floor Blue
#declare LDXFloorAmb = 0.4;     // Floor Ambient
#declare LDXFloorDif = 0.4;     // Floor Diffuse
#declare LDXAmb = 0.4;
#declare LDXDif = 0.4;
#declare LDXRefl = 0.08;
#declare LDXPhong = 0.5;
#declare LDXPhongS = 40;
#declare LDXTRefl = 0.2;
#declare LDXTFilt = 0.85;
#declare LDXIoR = 1.25;
#declare LDXRubberRefl = 0;
#declare LDXRubberPhong = 0.1;
#declare LDXRubberPhongS = 10;
#declare LDXChromeRefl = 0.85;
#declare LDXChromeBril = 5;
#declare LDXChromeSpec = 0.8;
#declare LDXChromeRough = 0.01;
#declare LDXIPov = 1;   // Use inline POV code from LDraw file? (1 = YES; 0 = NO)
#declare LDXBgR = 1;    // Background Red
#declare LDXBgG = 1;    // Background Green
#declare LDXBgB = 1;    // Background Blue
#declare LDXOrigVer = version;  // DO NOT MODIFY

// Camera
#ifndef (LDXSkipCamera)
camera {
        #declare LDXCamAspect = 1280/720;
        location LDXCameraLoc
        sky LDXCameraSky
        right LDXCamAspect * < -1,0,0 >
        look_at LDXCameraLookAt
        angle 8.876736
}
#end
```

Lo primero que notarás es que hay varios colores en la pantalla. Cada color representa algo específico en POV-Ray, así que vamos a hablar de cada uno por separado.

Verde está reservado para los comentarios. POV-Ray ignora totalmente los comentarios, por lo que sólo están ahí para ayudarte a ti, el usuario. LDView hace un trabajo bastante bueno comentando sus datos de salida para decirte lo que está haciendo, y es una buena idea crear comentarios para ti mismo cuando cambies el archivo, para que puedas realizar un seguimiento de lo que está pasando. Este archivo de ejemplo es bastante sencillo y ocupa sólo un par de pantallas, pero si se trata de miles de piezas, tu archivo será de muchos Mb de texto sin formato y puede ser bastante difícil encontrar algo si no lo has comentado correctamente.

Hay dos maneras de designar una línea de comentario en POV-Ray. La forma más común es simplemente poner dos barras invertidas al principio de la línea. Adelante, pruébalo. Teclea dos barras al comienzo de una línea y verás que se vuelve verde en el editor. A veces es posible que desees convertir un gran bloque en un comentario. Por ejemplo, puedes tener dos conjuntos diferentes de iluminación o dos cámaras diferentes, y puedes "comentar" uno o el otro para desactivarlos. Para convertir múltiples líneas en un comentario, pon /* al principio y */ al final. Puedes abarcar tantas filas como desees, incluso la totalidad del fichero. Inténtalo.

El texto morado se utiliza para los comandos. Hay cientos de comandos diferentes en POV-Ray, pero sólo vas a utilizar un pequeño número de ellos. Puedes ver una muestra de los utilizados por la exportación de LDView en el archivo.

El texto en azul se utiliza para los valores. Por ejemplo, podrían ser coordenadas o cualquier otro tipo de número.

El texto de color rojo se utiliza para los operadores como +, -, * y así sucesivamente. Puedes utilizarlos para realizar operaciones matemáticas dentro de tu archivo. Esto se vuelve muy importante cuando estás programando animaciones, por ejemplo. El rojo también se usa para nombres de archivo si haces referencia a archivos externos, lo que harás muchas veces.

Por último, el texto negro es para todo lo demás, aunque esto principalmente equivale a nombres de variables.

Ahora vamos a empezar por el principio del archivo y ver lo que allí hay. Las primeras filas son comentarios de LDView simplemente diciéndote que este es un archivo de LDView y dando crédito a Travis Cobbs por su excelente trabajo al escribir el programa.

Primero viene la definición de algunas variables.

Las seis variables siguientes se pueden utilizar para cambiar el comportamiento del render:

· La primera variable, LDXQual, cambia la calidad general del render. Si pones un 0 aquí tus piezas serán reemplazadas por rectángulos simples que te permitirán hacer un render muy rápidamente y comprobar que los ángulos de la cámara y la iluminación son correctos. Si colocas un 1 aquí conseguirás las piezas reales, pero la luz no se refracta a través de las piezas transparentes lo que consume mucho tiempo. El 2 es el ajuste estándar. Por último, el 3 habilita los logos en los studs, lo que significa que un pequeño logo de LEGO® será representado en cada stud. Los logos en los stud agregan una gran cantidad de detalle a tu modelo, pero por supuesto esto se produce a expensas de memoria y tiempo de procesamiento.
· El siguiente es LDXSW que controla un ancho de juntura. Esta es una manera de añadir un poco de espacio entre las piezas a fin de que no todo se vea perfectamente a ras, que es lo que obtendrás por defecto utilizando la geometría LDraw. Un valor de 0,5 (esto significa ½ de una unidad de LDraw) funciona muy bien aquí. Una advertencia es que si el modelo tiene pegatinas esto las hará desaparecer porque las pegatinas tienen menos de 0,5 LDU de espesor! Me llevó bastante tiempo averiguar porqué estaba sucediendo.
· LDXStuds se encarga de representar o no los studs. Yo nunca lo apagaría. Lo fundamental de hacer un render es que se vea real y LEGO sin studs simplemente no lo hace. La única vez que lo desconecté fue cuando hice un render tan grande (más de un millón de piezas) que no se podían ver los studs de todos modos y quitarlos disminuyó masivamente los requisitos de memoria.
· LDXRefls enciende o apaga las reflexiones. Como en el punto anterior. Yo siempre las dejo encendidas.
· LDXShads es para sombras. Igual que el anterior.
· Por último, LDXFloor enciende o apaga un suelo. Ten en cuenta que LDView asume que tu modelo se construye con la dirección -y hacia arriba que es lo que MLCAD utiliza. Para colocar el suelo, por lo tanto, encuentra el valor más positivo para "y" en el modelo y crea un plano XZ en esa ubicación. Puedes o no querer verlo, e incluso si lo quieres, puede no ser el lugar donde deseas que esté. Por supuesto, se puede mover y su comportamiento se puede modificar.

Después hay una sección llamada Model Bounds. Esta define la extensión máxima del modelo en x, y, y z. Puede que no veas la importancia de esto, pero la disponibilidad de estas variables es increíblemente útil más adelante. Las extensiones máximas también permiten calcular el centro. Hay también un radio que asume que el prisma rectangular asignado por los otros valores es una esfera. Esto se utiliza para colocar la cámara y las luces.

A continuación viene la sección que define algunas variables utilizadas por la cámara. Estas incluyen la posición de la cámara, la dirección a la que está apuntando, y una definición de qué dirección es arriba. Estos son LDXCameraLoc, LDXCameraLookAt, y LDXCameraSky.

Luego vienen un montón de otras definiciones de las variables. No pasaré a través de cada una en detalle, pero incluyen la posición del suelo, el color del suelo, algunas propiedades del suelo, algunas de las variables utilizadas para definir los colores, y el color de fondo. En general, no vamos a meternos en ellas, pero por supuesto cualquiera puede modificarse. Una cosa a observar aquí es que los colores se definen como porcentajes de rojo, verde y azul. Así cada parte del valor RGB puede estar entre 0 y 1. Si estás convirtiéndolo desde otro sistema como el hexadecimal en el que color pleno es FF o 255, tendrás que hacer algunos cálculos para convertirlos a porcentajes.

Ahora se define la propia cámara. Hay muchos tipos de cámaras en POV-Ray, pero la cámara por defecto de LDView siempre será una cámara perspectiva. Si deseas utilizar cualquier otro tipo especial (como el ojo de pez), tendrás que definirla por ti mismo. Los parámetros para la cámara provienen de las variables definidas anteriormente. Estos, a su vez, provenían de cómo tenías la vista configurada en LDView. La cámara simplemente coincide con lo que vimos en LDView. Una cosa a tener cuidado aquí es LDXCamAspect que define la relación de aspecto. Esta se basa en la forma de tu ventana en LDView. Cuando

se procesa la imagen, es necesario utilizar esta misma relación de aspecto o tu render resultará torcido. No es necesario que coincida con los valores de píxeles, sólo la relación. Es una buena idea asegurarte de que tu ventana en LDView es una relación estándar como 4:3 o 16:9 antes de exportar por primera vez.

```
// Lights
#ifndef (LDXSkipLight1)
light_source {  // Latitude,Longitude: 45,0,LDXRadius*2
        <0*LDXRadius,-1.414214*LDXRadius,-1.414214*LDXRadius> + LDXCenter
        color rgb <1,1,1>
}
#end
#ifndef (LDXSkipLight2)
light_source {  // Latitude,Longitude: 30,120,LDXRadius*2
        <1.5*LDXRadius,-1*LDXRadius,0.866026*LDXRadius> + LDXCenter
        color rgb <1,1,1>
}
#end
#ifndef (LDXSkipLight3)
light_source {  // Latitude,Longitude: 60,-120,LDXRadius*2
        <-0.866025*LDXRadius,-1.732051*LDXRadius,0.5*LDXRadius> + LDXCenter
        color rgb <1,1,1>
}
#end

#macro LDXSeamMatrix(Width, Height, Depth, CenterX, CenterY, CenterZ)
#local aw = 0;
#local ah = 0;
#local ad = 0;
#local ax = 0;
#local ay = 0;
#local az = 0;
#if (Width != 0)
#local aw = 1-LDXSW/Width;
#end
#if (Height != 0)
#local ah = 1-LDXSW/Height;
#end
#if (Depth != 0)
#local ad = 1-LDXSW/Depth;
#end
#if (Width != 0 & CenterX != 0)
#local ax = LDXSW/(Width / CenterX);
#end
#if (Height != 0 & CenterY != 0)
#local ay = LDXSW/(Height / CenterY);
#end
#if (Depth != 0 & CenterZ != 0)
#local az = LDXSW/(Depth / CenterZ);
#end
#if (aw <= 0)
#local aw = 1;
#local ax = 0;
#end
#if (ah <= 0)
#local ah = 1;
#local ay = 0;
#end
#if (ad <= 0)
#local ad = 1;
#local az = 0;
#end
matrix <aw,0,0,0,ah,0,0,0,ad,ax,ay,az>
#end


background { color rgb <LDXBgR,LDXBgG,LDXBgB> }

#declare lg_quality = LDXQual;
#if (lg_quality = 3)
#declare lg_quality = 4;
#end

#declare lg_studs = LDXStuds;


#include "lg_defs.inc"

#include "lg_color.inc"
```

Ahora se definen las luces. Las luces por defecto son 3 luces puntuales blancas. Esta sección, más que cualquiera otra, es la que vamos a cambiar más tarde para hacer la escena más realista. Se puede ver que las luces están dispuestas según un concepto de latitud y longitud. Cada luz tiene una posición y un color. El color se define como un vector con los valores RGB. rgb <1,1,1> es blanco.

A continuación viene un montón de matemática que define una subrutina usada para hacer las juntas entre las piezas. Definitivamente no quieres liarte con esto.

A continuación, se define el color de fondo. Éste también es blanco de forma predeterminada. Puedes apagar por completo el fondo “comentando” esta línea.

Los archivos POV-Ray no suelen contener toda la información requerida en el mismo archivo. En cambio, llama a otros archivos llamados archivos “include”. Un archivo grande de POV-Ray puede llamar a cientos o incluso miles de otros archivos. Esto mantiene el tamaño del archivo relativamente pequeño, pero tienes que tener en cuenta al planificar los requisitos de memoria, que todo tiene que estar en la memoria cuando procese el render. A veces es útil abrir algunos de estos archivos sólo para entender cuánto trabajo está haciendo POV-Ray en realidad.
Los dos primeros archivos que se cargan están relacionados con la biblioteca LGEO. El lg_defs contiene la definición de muchas variables utilizadas por archivos de piezas de LGEO. Esto incluye cosas como el tamaño de un stud, cómo es el logo, la anchura de un plate, y cientos de otras cosas. Sigue adelante y échale un vistazo. Simplemente haz clic en el título rojo del archivo “include” y lo puedes abrir. El lg_color no es ningún misterio. En él se definen los colores utilizados por LGEO. Hay un número sorprendentemente grande de colores de LEGO®!

```
#include "lg_defs.inc"

#include "lg_color.inc"

#ifndef (LDXColor4) // Red
#declare LDXColor4 = #if (version >= 3.1) material { #end
        texture {
                lg_red
        }
#if (version >= 3.1) } #end
#declare LDXColor4_slope = #if (version >= 3.1) material { #end
        texture {
                lg_red
                #if (LDXQual > 1) normal { bumps 0.3 scale 25*0.02 } #end
        }
#if (version >= 3.1) } #end
#end

#ifndef (LDXColor7) // Light Gray
#declare LDXColor7 = #if (version >= 3.1) material { #end
        texture {
                lg_grey
        }
#if (version >= 3.1) } #end
#declare LDXColor7_slope = #if (version >= 3.1) material { #end
        texture {
                lg_grey
                #if (LDXQual > 1) normal { bumps 0.3 scale 25*0.02 } #end
        }
#if (version >= 3.1) } #end
#end

#include "lg_3001.inc" // Brick  2 x  4

#declare LDX_brick_dot_ldr = union {
// Brick Render
        object {
                lg_3001
                matrix <0,0,-25,-25,0,0,0,-25,0,0,0,0>
                LDXSeamMatrix(80, 28, 40, 0, 10, 0)
                matrix <1,0,0,0,1,0,0,0,1,0,0,0>
                #if (version >= 3.1) material #else texture #end { LDXColor4 }
        }
#if (LDXRefls = 0)
        no_reflection
#end
#if (LDXShads = 0)
        no_shadow
#end
}

// brick.ldr
object {
        LDX_brick_dot_ldr
        #if (version >= 3.1) material #else texture #end { LDXColor7 }
}

// Floor
#if (LDXFloor != 0)
object {
        plane { LDXFloorAxis, LDXFloorLoc hollow }
        texture {
                pigment { color rgb <LDXFloorR,LDXFloorG,LDXFloorB> }
                finish { ambient LDXFloorAmb diffuse LDXFloorDif }
        }
}
#end
```

A continuación se presentan algunas definiciones de color. Sólo encontrarás las definiciones de los colores que utilices. En este caso, es sólo rojo. ¿Por qué está el color gris claro? El conjunto de nivel superior se considera siempre gris claro, por lo que este siempre estará ahí. Si abres el archivo lg_colors, verás que los colores se definen no sólo con un color, sino con algunas otras propiedades de la superficie como la reflectividad. También terminaremos haciendo muchos cambios aquí en tutoriales posteriores.

Finalmente, es tiempo para las piezas reales. En este caso, no tenemos más que una sola pieza: un ladrillo 2x4. Esta pieza ha sido reemplazada con una pieza LGEO, lg_3001. Así que todo lo que POV-Ray tiene que hacer, es llamar al archivo “include” lg_3001.inc. Una vez más, si haces clic en ese texto puedes abrir ese archivo. En la página siguiente puedes ver lo increíblemente complicada que es una sola pieza!

Esto comenzará a darte una idea de la labor que POV-Ray está haciendo por ti. Recuerda que esto es sólo la definición de un tipo de pieza, y tú puedes tener cientos de tipos de piezas en el modelo.

Algunas piezas no tienen un sustituto LGEO. En esos casos, verás una definición extensa de la pieza en el archivo principal POV-Ray generado por LDView. Esta definición puede abarcar cientos de páginas para una pieza compleja como un panel Technic curvado o una cabeza de Fabuland.

Nuestra escena sólo incluía un archivo de una pieza, pero un modelo más grande puede incluir muchos. Cada uno sólo se incluye una vez, pero puede haber muchos casos en que se llama a esa parte cuando el modelo está montado. Eso es lo que encontramos en nuestro próximo archivo. Ahora que POV-Ray sabe el aspecto de una pieza lg_3001, podemos poner algunos ladrillos en el espacio. Esta es la sección que comienza con “object”. En este caso, se crea un objeto en el espacio a partir de lg_3001. La línea más importante es la siguiente:

matriz <1,0,0,0,1,0,0,0,1,0,0,0>

```
/*****************************************************************************/
/*                                                                           */
/* LGEO Library Include File     (C) lgeo@digitalbricks.org (Lutz Uhlmann)   */
/*                                                                           */
/* 19970623 Lutz Uhlmann                                                     */
/* 20071225 Lutz Uhlmann fixed stud orientation                              */
/*                                                                           */
/* This file is in no way related to the LEGO(tm) Group.                     */
/* It is provided for private non-commercial use only.                       */
/*                                                                           */
/* lg_3001: Brick 2 x 4                                                      */
/*                                                                           */
/*****************************************************************************/
#ifdef(lg_3001)
#else
#declare LENGTH = 4;
#declare WIDTH = 2;
#declare lg_3001 =
union {
 sphere {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <(LENGTH*LG_BRICK_WIDTH-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>,
  LG_CORNER_SPACE
 }
 sphere {
  <(LENGTH*LG_BRICK_WIDTH-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>, LG_CORNER_SPACE
 }
 cylinder {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 sphere {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>, LG_CORNER_SPACE
 }
 cylinder {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 sphere {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>,
  LG_CORNER_SPACE
 }
 sphere {
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>,
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 sphere {
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>, LG_CORNER_SPACE
 }
 cylinder {
  <LG_CORNER_SPACE, LG_CORNER_SPACE, (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  <LG_CORNER_SPACE, ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), (LG_BRICK_HEIGHT-LG_CORNER_SPACE)>,
  LG_CORNER_SPACE
 }
 cylinder {
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE, LG_CORNER_SPACE>,
  <((LENGTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), ((WIDTH*LG_BRICK_WIDTH)-LG_CORNER_SPACE), LG_CORNER_SPACE>,
```

¿Recuerdas cuando aprendimos un poco sobre matrices de transformación en la última lección? Las primeras nueve cifras aquí son la rotación en los planos x, y, y z nada más sacarlo de nuestro archivo LDraw. Las tres siguientes son la posición en el espacio, en este caso 0,0,0. Este es un ejemplo muy simple, pero este término puede ser realmente complicado. Puede haber casos en los que desees mover manualmente o girar una pieza cambiando estos valores.

A continuación puedes ver que se asigna el color rojo a esta pieza (LDXColor 4). De nuevo, es bueno ser capaz de cambiar estos colores manualmente sabiendo dónde buscar.

Si tuviéramos un modelo más grande, hecho de submodelos, veríamos cada submodelo definido como una “unión” o una combinación de piezas, y luego los submodelos unidos en una combinación más grande al final. El último objeto en el archivo antes del suelo es siempre el ensamblaje de nivel superior. En este caso, el montaje es sólo nuestro ladrillo. Puedes ver lo que te dije antes sobre cómo este conjunto superior se define como gris claro (LDXColor 7). Esto no quiere decir nada. Las piezas ya tienen un color y esto no va a sobrescribirlo. Sólo los colores de las piezas son importantes, no los colores de montaje.

La última cosa que se define es el suelo, y utiliza las variables del principio del archivo. Una vez más, vamos a hacer cambios en éste más tarde.

Puede parecer que la mayor parte del archivo eran “generalidades” definiendo variables, cámaras y luces, pero no mucho del modelo. Esto es cierto para un modelo pequeño de una sola pieza. Cuando tengas un modelo grande, lo descubrirás.

Habiendo recorrido el archivo, deberías tener una comprensión básica de lo que todo eso significa. Este es un pequeño archivo que es relativamente fácil de seguir. Ahora te animo a simplemente probar a cambiarlo todo y ver qué pasa. Cambia solo una de las variables cada vez y observa el efecto. Esta es la mejor manera de aprender lo que hacen. La próxima vez te voy a enseñar cómo cambiar algunas de las variables y definiciones más importantes para mejorar la calidad de tus renders.
#