|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Java 3D**   * [Texturas en Java 3D](about:blank)   + [¿Qué es el Texturado?](about:blank)   + [Texturado Básico](about:blank)     - [Sencilla Receta de Texturado](about:blank)     - [Sencillos Ejemplos de Programas de Textura](about:blank)     - [Más sobre las Coordenadas de Textura](about:blank)     - [Preview de Algunas Opciones de Texturado](about:blank)     - [Opciones de Textura](about:blank)     - [Texture3d](about:blank)   + [Algunas Aplicaciones de Texturado](about:blank)     - [Texturado de Geométricos Primitivos](about:blank)     - [Texturado de Líneas](about:blank)     - [Usar Texturas Text2D](about:blank)   + [Atributos de Textura](about:blank)     - [Modo de Textura](about:blank)     - [Textura con Color de Mezcla](about:blank)     - [Modo de Corrección de Perspectiva](about:blank)     - [Transformación del Mapeo de Textura](about:blank)     - [API TextureAttributes](about:blank)   + [Generación Automática de Coordenadas de Textura](about:blank)     - [Formato de Generación de Textura](about:blank)     - [Modo de Generación de Textura](about:blank)     - [Cómo usar un Objeto TexCoordGeneration](about:blank)     - [API TexCoordGeneration](about:blank)   + [Múltiples Niveles de Textura (Mipmaps)](about:blank)     - [¿Qué es el Texturado Multi-Nivel (MIPmap)?](about:blank)     - [Ejemplos de Texturas Multi-Nivel](about:blank)     - [Filtros de Reducción para Múltiples Niveles de Textura](about:blank)     - [Modo Mipmap](about:blank)   + [API de Texture, Texture2D, y Texture3d](about:blank)     - [Filtros de Reducción y Ampliación](about:blank)     - [API Texture](about:blank)     - [API de Texture2D](about:blank)     - [API de Texture3d](about:blank)   + [API de TextureLoader y NewTextureLoader](about:blank)     - [API de TextureLoader](about:blank)     - [API de NewTextureLoader](about:blank)     **Texturas en Java 3D**  El aspecto de muchos objetos del mundo real depende de su textura. La textura de un objeto es realmente la geometría relativamente fina de la superficie de un objeto. Para apreciar la superficie del papel las texturas juegan con el aspecto de los objetos del mundo real, consideremos una alfombra. Incluso cuando todas las fibras de una alfombra son del mismo color la alfombra no aparece con un color constante debido a la interacción de la luz con la geometría de las fibras. Aunque Java 3D es capaz de modelar la geometría de las fibras individuales de la alfombra, los requisitos de memoria y el funcionamiento de la renderización para una alfombra del tamaño de una habitación modelada a tal detalle harían dicho modelo inútil. Por otra parte, tener un polígono plano de un solo color no hace un reemplazo convincente para la alfombra en la escena renderizada.  Hasta ahora en el tutorial, los detalles de los objetos visuales los ha proporcionado la geometría. Consecuentemente, los objetos visualmente ricos, como los árboles, pueden requerir mucha geometría que a cambio requiere mucha memoria y cálculo de renderización. A cierto nivel de detalle, el rendimiento puede llegar a ser inaceptable. Este capítulo muestra cómo añadir el aspecto del detalle superficial a un objeto visual sin la adición de más geometría con el uso de texturas  **¿Qué es el Texturado?**  Una alfombra puede ser un ejemplo extremo en términos de complejidad y de densidad de la geometría superficial, pero está lejos de ser el único objeto para el cual percibimos textura. Los ladrillos, el cemento, la madera, los céspedes, las paredes, y el papel son sólo algunos de los objetos que no se representan bien con polígonos planos (no-texturados). Pero, igual que con la alfombra, el coste de representar la textura superficial en los primitivos geométricos para estos objetos sería muy alto.  Una posible alternativa a modelar la fibra de la alfombra es modelar la alfombra como un polígono plano con muchos vértices, asignando colores a los vértcies para darle variaciones de color. Si los vértices están suficientemente cercanos, se puede reproducir la imagen de la alfombra. Esto requiere significativamente menos memoria que el modelo que incluye las fibras de la alfombra; sin embargo, el modelo todavía requiere demasiada memoria para un tamaño de habitación razonable. Esta idea, de representar la imagen del objeto en una superficie plana, es la idea básica de texturado. Sin embargo, con el texturado, la geometría puede ser muy sencilla.  El texturado, también llamado mapeo de textura, es una manera de añadir riqueza visual a una superficie sin la adición de los detalles geométricos finos. La riqueza visual la proporciona una imagen, también llamada textura, que da el aspecto del detalle superficial para el objeto visual. La imagen se mapeda dentro de la geometría del objeto visual en el momento de la renderización. De ahí el término mapeo de textura.  La Figura 7-1 muestra algunas de las texturas usadas en los programas de ejemplo de este capítulo. Como podemos ver, una textura puede proporcionar riqueza visual a objetos de distintos tamaños.  Figura 7-1, Imágenes usadas como Texturas en los Programas de Ejemplo.  **Texturado Básico**  El texturado de polígonos en un programa de Java 3D se consigue a través de la creacción del manojo de apariencia apropiado y cargando la imagen de la textura dentro de él, especificando la localización de la imagen de la textura en la geometría, y fijando los atributos de texturado. Como veremos, especificar texturas puede ser muy complejo. Afortunadamente, hay clases de utilidad para ayudarnos en el proceso y las configuraciones de los valores por defecto para las opciones texturado son las apropiadas para las aplicaciones de texturado básicas.  Para explicar el texturado, la siguiente sección presenta una sencilla receta; y en secciones posteriores se desarrolla un programa de ejemplo basado en la receta, además de explicar el texturado.  **Sencilla Receta de Texturado**  Debido a la flexibilidad del texturado en el API Java 3D, el número de opciones relacionadas con el texturado puede ser un poco fastidioso. Incluso así, el texturado no es necesariamente difícil. Para hacer fácil el trabajo de especificación de textura, seguimos los pasos de la siguiente receta.  La receta solo subraya los pasos relacionados directamente con el texturado. Deberíamos haber observado que la geometría y la apariencia se fijan en un objeto **Shape3D** que se agrega al escenario gráfico.   1. Preparar las Imágenes de Texura 2. Cargar la Textura 3. Configurar la textura en el manojo Appearance 4. Especificar las TextureCoordinates del Geometry   Al igual que muchas de las recetas de este tutorial, algunos de estos pasos se pueden realizar en cualquier orden. De hecho, los pasos de esta receta se pueden realizar en cualquier orden (siempre que los los pasos 2 y 3 se hagan juntos).  **Texturado Paso 1: Preparar las Imágenes de Texturas**  Esta receta comienza con un paso sin programación: "preparar las imágenes de textura". Crear y corregir imágenes de textura es algo que normalmente se hace externamente a los programas de Java 3D. De hecho, la mayoría de las imágenes de textura están preparadas antes de que se comience el programa. Hay dos tareas esenciales en la preparación de la imagen de textura: 1. asegurarnos de que las imágenes sean de dimensiones aceptables, y 2. asegurarnos de que las imágenes se graban en un formato de fichero que pueda ser leído. Por supuesto se puede editar la imagen para alcanzar el color, la transparencia, y las características deseadas.  Por razones de eficiencia , Java 3D necesita que el tamaño de la imagen de la textura sea una potencia matemática de dos (1, 2, 4, 8, 16, ...) en todas las dimensiones. No cumplir esta restricción dará lugar a una excepción en tiempo de ejecución.  Si una imagen no es de las dimensiones aceptables, debe ser modificada (escalada o recortada) para cumplir los requisitos de dimensión antes de que sea utilizada. La edición de la imagen se puede hacer en una gran variedad de programas incluyendo el **API Java Advanced Imaging**. En la Figura 7-1, las dos imágenes más pequeñas son de 128 por 128, el árbol es de 256 por 128, y la tierra es de 256 por 256.  En lo que concierne a los formatos de fichero, se puede utilizar cualquier formato de fichero siempre que proporcionemos un métodos para cargarlo. Los programas de este capítulo cargan texturas usando la clase de utilidad **TextureLoader**. Un objeto **TextureLoader** carga JPEG, GIF, y otros formatos de fichero.  Una palabra más sobre los programas del ejemplo antes de pasar al paso siguiente. Los fragmentos del código y los programas de ejemplo de este capítulo utilizan los nombres del archivo para algunos ficheros de imagen que están incluidos en los ficheros Jar de ejemplo. No hay nada especial en estos ficheros de imagen a excepción de que cumplen con la restricción de las dimensiones. Cualquier fichero de imagen se puede utilizar en los programas simpre que las dimensiones de las imágenes sean potencia de dos . Podemos compilar y ejecutar los programas del ejemplo con nuestros propios ficheros de a imagen. Ahora, con las imágenes de texturas listas, podemos empezar la programación.  **Texturado Paso 2: Cargar la Textura**  El siguiente paso es conseguir la imagen preparada en un objeto imagen. Esto se conoce como cargar la textura. Las texturas se pueden cargar desde ficheros o URLs usando de mismo proceso básico. Cargar una textura se puede lograr con muchas líneas del código, o con dos líneas de código que utilicen un objeto **TextureLoader**. De cualquier forma, el resultado es conseguir la imagen en un objeto **ImageComponent2D**. El [Fragmento de Código 7-1](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-1) muestra un ejemplo de dos líneas que utilizan un **TextureLoader**. El resultado de estas dos líneas es cargar la imagen del fichero stripe.gif en un objeto **Image2DComponent** que se pueda utilizar para crear el manojo de apariencia necesario para el paso 4.  **Fragmento de Código 7-1, usar un objeto TextureLoader Object para Cargar el fichero de imagen STRIPE.GIF**  1. TextureLoader loader = new TextureLoader("stripe.gif", this);  2. ImageComponent2D image = loader.getImage();  Antes de pasar al paso 4 de la receta, echemos una ojeada más cercana el uso del objeto **TextureLoader**. El segundo argumento del constructor especifica un objeto que sirva como image observer. La clase **TextureLoader** utiliza el paquete java.awt.image para cargar las imágenes. Este paquete carga las imágenes de forma asíncrona, lo que es particularmente útil cuando una imagen se carga de una URL. Para facilitar el manejo de cargas asíncronas de imágenes, los componentes de AWT están capacitados para ser observadores de imagen, que es observar el proceso de la carga de la imagen. A un observador de imagen se le puede preguntar por los detalles de la carga de la imagen.  Con el fin escribir programas Java 3D todo lo que necesitamos saber es que cualquier componente del AWT puede servir como un observador de imagen. Puesto que **Applet** es una extensión del componente **Panel** del AWT, el objeto **Applet** de un programa de Java 3D puede ser el observador de imagen para el objeto **TextureLoader**.  **Texturado Paso 3: Crer el Manojo de Appearance**  Para ser utilizada como textura para un objeto visual, la imagen de textura cargada en el paso de 2a se debe asignar como la textura de un objeto **Texture**, que entonces se utiliza en un manojo de apariencia referenciado por el objeto visual. Específicamente, un objeto **Texture2D** contiene la imagen de la textura. La imagen **ImageComponent2D** cargada en el paso 2a es el centro de la creación del manojo de apariencia del paso 2b.  El [Fragmento de Código 7-2](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-2) muestra dos líneas del código del paso 2a seguidas por el código para formar un sencillo manojo de apariencia texturado. Cargando la textura (líneas 1 y 2), la imagen entonces se asigna al objeto **Texture2D** (línea 4). Luego el objeto **Texture2D** se agrega al objeto **Appearance**(línea 6).  **Fragmento de Código 7-2, Crear un Appearance con un objeto Texture.**  1. TextureLoader loader = new TextureLoader("stripe.jpg", this);  2. ImageComponent2D image = loader.getImage();  3. Texture2D texture = new Texture2D();  4. texture.setImage(0, image);  5. Appearance appear = new Appearance();  6. appear.setTexture(texture);  El manojo de apariencia creado en el [Fragmento de Código 7-2](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-2) podría tener otros nodos componentes, la más notable de las posibilidades es el nodo componente **TextureAttributes**. Para este ejemplo, no se utiliza ningún objeto **TextureAttributes**.  **Texturado Paso 4: Especificar TextureCoordinates**  Además de cargar la textura en un manojo de apariencia, el programador también especifica la colocación de la textura en la geometría a través de la especificación de las coordenadas de textura. Las especificaciones de coordenadas de textura se hacen por cada vértice de la geometría. Cada coordenada de textura especifica un punto de textura que se aplicará al vértice. Con la especificación de algunos puntos de la imagen que se aplicarán a los vértices de la geometría, la imagen será rotada, estirada, aplastada, y/o duplicada para hacer que quepa en la especificación.  **TextureCoordinates** se especifica en las dimensiones s (horizontal) y t (verticales) de la imagen de textura según lo mostrado en la Figura 7-3. Esta figura muestra las coordenadas de la textura en el espacio de la imagen de textura.  Figura 7-3, Coordenadas de Mapeo de Textura  El siguiente bloque de referencia muestra sólo uno de los métodos de **GeometryArray** disponible para fijar coordenadas de textura.   |  | | --- | | Método setTextureCoordinate de **GeometryArray**  Las coordenadas de textura se especfican por cada vértice de la geometría mediante uno de los distintos métodos setTextureCoordinate de la clase **GeometryArray**.  void setTextureCoordinate(int index, Point2f texCoord)  Selecciona las coordenadas de textura asociadas con el vértice del índice especificado para este objeto. |   El [Fragmento de Código 7-3](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-3) crea un solo plano usando un objeto de geometría **QuadArray**. Los coordenadas de la textura se asignan para cada vértice. En el código, las líneas tres a once establecen las cuatro esquinas de un cuadrángulo en 3-espacios. Las líneas 13 a 21 establecen la localización de la textura en la geometría. Este fragmento determinado de código crea un plano de 2 metros en una cara y pone la imagen de la textura en la orientación normal (hacia arriba, no invertido) a lo largo de la cara del plano.  **Fragmento de Código 7-3, Aplicar Coordenadas de Texturas a un Quad.**  1. QuadArray plane = new QuadArray(4, GeometryArray.COORDINATES  2. | GeometryArray.TEXTURE\_COORDINATE\_2);  3. Point3f p = new Point3f();  4. p.set(-1.0f, 1.0f, 0.0f);  5. plane.setCoordinate(0, p);  6. p.set(-1.0f, -1.0f, 0.0f);  7. plane.setCoordinate(1, p);  8. p.set( 1.0f, -1.0f, 0.0f);  9. plane.setCoordinate(2, p);  10. p.set( 1.0f, 1.0f, 0.0f);  11. plane.setCoordinate(3, p);  12.  13. Point2f q = new Point2f();  14. q.set(0.0f, 1.0f);  15. plane.setTextureCoordinate(0, q);  16. q.set(0.0f, 0.0f);  17. plane.setTextureCoordinate(1, q);  18. q.set(1.0f, 0.0f);  19. plane.setTextureCoordinate(2, q);  20. q.set(1.0f, 1.0f);  21. plane.setTextureCoordinate(3, q);  La Figura 7-4 muestra la relación entre las coordenadas de vértice y las coordenadas de la textura para el cuadrángulo del ejemplo creado en el [Fragmento de Código 7-3](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-3). La imagen izquierda de la Figura 7-5 muestra la aplicación de la textura **stripe.gif** a la geometría del ejemplo.  Figure 7-5. La orientación de las Coordenadas de Textura en el Espacio de una Imagen de Textura.  Ahora que hemos completado los tres pasos del texturado, el objeto texturado puede añadirse a un escenario gráfico. La siguiente sección presenta una serie de programas de ejemplo que muestra algunas opciones de texturado.  **Sencillos Ejemplos de Programas de Textura**  Siguiendo la receta anterior, se ha desarrollado un sencillo programa de ejemplo de texturado. [SimpleTexturaApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/SimpleTextureApp.java). Este programa es poco más que un programa de visualización de imágenes.  Figura 7-5, Escena Renderizada por los programas SimpleTextureApp (izq.) y SimpleTextureSpinApp (der.).  Se ha creado otro programa de ejemplo [SimpleTexturaSpinApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/SimpleTextureSpinApp.java) utilizando un objeto **RotationInterpolator**. En este programa se hace girar el mismo plano texturado para demostrar la naturaleza 3D del programa. La Figura 7-5 muestra una renderización de este programa a la derecha. Una cosa a observar cuando se ve el programa, la cara posterior del plano está en blanco.  **La Clase NewTextureLoader**  Los programas Java 3D que usan texturas pueden tener una gran cantidad de líneas sólo para cargar las texturas y crear los manojos de apariencia. Se puede ahorrar algo de programación y, más importante, memoria en tiempo de ejecución compartiendo manojos de apariencia cuando sea apropiado. Sin embargo, esto no reduce mucho la cantidad de programación. Se pueden conseguir otras reducciones de programación creando una clase para crear los manojos de apariencia de textura. El desafío de crear esta clase consiste en el requisito del observador de imagen para el objeto **TextureLoader**.  El objeto **Canvas3d** o un **Applet** pueden servir como el observador de imagen, pero tener una referencia a un cierto componente por todas partes en el programa puede ser fastidioso. Para tratar esta inconveniencia, se ha extendido la clase **TextureLoader** que elimina la necesidad de un componente observador de imagen. En su lugar se utiliza un solo método para especificar un observador de imagen para todas las aplicaciones futuras del cargador de textura.  Los constructores de **NewTextureLoader** son iguales a los de **TextureLoader** excepto en que ninguno requiere un componente observador de imagen. Los métodos de **NewTextureLoader** son los mismos de **TextureLoader** con el método adicional para fijar un observador de imagen.  Otro programa del ejemplo,[TexturaPlaneApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TexturePlaneApp.java), carga tres texturas y las visualiza en los planos según lo mostrado en la Figura 7-6. Lo importante de este programa es que las texturas se cargan usando la clase **TexturedPlane** definida externamente al resto del programa que se hace más fácilmente con la clase **NewTextureLoader**. Esta clase **TexturedPlane** no es lo bastante flexible para ser utilizada en muchas aplicaciones, pero sirve como demostración para clases similares.  Figura 7-6, Tres Planos Texturados renderizados por TexturedPlaneApp.java  El [Fragmento de Código 7-4](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-4) es un extracto de [TexturaPlaneApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TexturePlaneApp.java) y es casi todo el código necesario para crear los tres planos texturadps de esta aplicación. El objeto observador de imagen se proporciona al objeto de **NewTextureLoader** del **TexturedPlane**.  **Fragmento de Código 7-4, Añadir tres objetos Texturados a un Escenario Gráfico.**  1. scene.addChild(tg0);  2. tg0.addChild(tg1);  3. tg1.addChild(new TexturedPlane("stripe.gif"));  4.  5. tg0.addChild(tg2);  6. tg2.addChild(new TexturedPlane("brick.gif"));  7.  8. tg0.addChild(tg3);  9. tg3.addChild(new TexturedPlane("earth.jpg"));  **Más sobre las Coordenadas de Textura**  Según lo mencionado en "Texturado Paso 3: Especificar TextureCoordinates", la imagen de la textura se crea para caber en la geometría basándose en la especificación de las coordenadas de textura. El proceso real es asociar los texels de la textura a los pixeles de la geometría cuando es renderizada. Cada pixel de una textura se llama un texel, o un 'elemento de textura'. Éste es el proceso de mapeado de la textura.  El mapeo de textura comienza con la especificación de las coordenadas de textura para los vértices de la geometría. Mientras se renderiza cada pixel del triángulo texturado, se calculan las coordenadas de la textura en el pixel desde los vértices del triángulo. La interpolación Trilinear de las coordenadas de textura de los vertices determina las coordenadas de textura para el pixel y por lo tanto, el texel de la imagen de la textura usada en el color final del pixel.  La Figura 7-7 ilustra el proceso de interpolación trilinear para un pixel del ejemplo. La representación se hace en orden de scan. El pixel, P, para el mapeo de textura está justo en el centro del scan actual en el triángulo de la izquierda de la ilustración. Se han asignado las coordenadas de textura para cada uno de los vértices del triángulo. Se han etiquetado TC1, TC2, y TC3. Estas coordenadas de textura son el punto de partida para la interpolación trilinear (cada una de las interpolaciones lineares se muestra como flechas dos-cabezas en la figura). Las primeras dos interpolaciones lineares determinan las coordenadas de la textura a lo largo de las caras del triángulo en el scan (puntos etiquetados A y B en la figura). La tercera interpolación se hace entre estos dos puntos.  Las coordenadas de textura que resultan para P son (0,75, 0,6). En la derecha de la figura está la textura. Usando las coordenadas de textura calculadas para P se selecciona el texel.  Figura 7-7, Imagen del Mapeo de Textura  La selección del Texel no se explica completamente en el ejemplo anterior. La especificación de la sección de filtración (ver secciones posteriores) da más detalles sobre la selección del texel. Otro detalle todavía no explicado es la interacción entre el color del texel, otras fuentes del color, y el color final del pixel. El modo de valor por defecto es 'substituye' con el cuál se utiliza el color del texel como el color del pixel, pero hay otros modos. Antes de pasar a otros asuntos, ya está en orden la discusión adicional de las coordenadas y el mapeo de textura.  En este punto del capítulo se han utilizado todas las texturas en su orientación ordinaria. La Figura 7-8 muestra planos con algunas de las posibles orientaciones de textura sólo seleccionando las coordenadas de textura en los vértices. El programa de ejemplo [TextureCoordApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TextureCoordApp.java) produce esta imagen.  Figura 7-8, Algunas de las posibles orientaciones de una Textura en un Plano.  Debemos observar que en el programa de ejemplo [TextureCoordinatesApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TextureCoordinatesApp.java) la textura **stripe.gif** se carga solamente una vez. Se crea solamente un manojo de apariencia de la textura que es compartido por los cuatro planos texturados. Esto es posible porque no hay nada en el manojo de la textura que sea único para cualquiera de los planos. Cargar la textura una vez ahorra tiempo y memoria.  Por supuesto, se pueden cometer errores al especificar las coordenadas de textura. Cuando sucede esto, el sistema de renderizado de Java 3D hace que se pregunte por él. Cuando las coordenadas de textura no se especifican para un espacio regularmente mapeado, entonces la triangulación de la geometría llega a ser obvia pues las 'costuras' de la textura se verán a lo largo de los bordes de triángulos.  La Figura 7-9 muestra la imagen renderizada de los planos texturados donde no se especifican las coordenadas de textura para hacer una presentación uniforme de la textura. El programa que genera esta imagen, [TexturaRequestApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TextureRequestApp.java), es solamente un manojo de textura compartido por tres objetos visuales. Las variaciones en el aspecto de los planos sólo son debidas a la especificación de las coordenadas de textura. Ésta es una representación de la frase "en texturado, se consigue lo que se pide".  Figura 7-9, En Texturado, Se Obtiene lo que se Pide.  Este programa muestra algunos de los posibles renderizados para un plano usando la misma textura. Las asignaciones de textura hechas en este programa son ejemplos de posibles errores mientras que todas son aplicaciones legítimas. La imagen de la izquierda es una aplicación de solo una sola fila de texels - las mismas coordenadas de textura se asignan a las parejas de vértices. La imagen de la derecha es la aplicación de un solo texel - las cuatro coordenadas de textura son iguales. Las dos imágenes del centro muestran la asignación de las coordenadas de textura de maneras no uniformes. El cambio de la textura a lo largo de la diagonal es debido a la triangulación del polígono.  El [Fragmento de Código 7-5](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-5) muestra las coordenadas de textura asignadas en la aplicación [TexturaRequestApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TextureRequestApp.java). Estas asignaciones usadas con la textura **stripe.gif** resultan en las imágenes de la Figura 7-9.  **Fragmento de Código 7-5, Asignaciones de Coordenadas de Textura para Planos en TextureRequestApp.**  1. // texture coordinate assignments fof the first plane  2. texturedQuad.setTextureCoordinate(0, new Point2f( 1.0f, 0.0f));  3. texturedQuad.setTextureCoordinate(1, new Point2f( 1.0f, 0.0f));  4. texturedQuad.setTextureCoordinate(2, new Point2f( 0.0f, 0.0f));  5. texturedQuad.setTextureCoordinate(3, new Point2f( 0.0f, 0.0f));  6. // texture coordinate assignments for the second plane  7. texturedQuad.setTextureCoordinate(0, new Point2f( 0.0f, 1.0f));  8. texturedQuad.setTextureCoordinate(1, new Point2f( 1.0f, 0.5f));  9. texturedQuad.setTextureCoordinate(2, new Point2f( 0.5f, 0.5f));  10. texturedQuad.setTextureCoordinate(3, new Point2f( 0.0f, 1.0f));  11. // texture coordinate assignments for the third plane  12. texturedQuad.setTextureCoordinate(0, new Point2f( 1.0f, 0.0f));  13. texturedQuad.setTextureCoordinate(1, new Point2f( 1.0f, 1.0f));  14. texturedQuad.setTextureCoordinate(2, new Point2f( 0.0f, 0.0f));  15. texturedQuad.setTextureCoordinate(3, new Point2f( 1.0f, 1.0f));  16. // texture coordinate assignments for the forth plane  17. texturedQuad.setTextureCoordinate(0, new Point2f( 0.0f, 0.0f));  18. texturedQuad.setTextureCoordinate(1, new Point2f( 0.0f, 0.0f));  19. texturedQuad.setTextureCoordinate(2, new Point2f( 0.0f, 0.0f));  20. texturedQuad.setTextureCoordinate(3, new Point2f( 0.0f, 0.0f));  **Preview de Algunas Opciones de Texturado**  Hay mucho más en el texturado que sólo especificar las coordenadas de textura para los vértices de la geometría. En este punto, la discusión de texturado no ha incluido ninguna de las opciones disponibles en aplicaciones de textura. Por ejemplo, el objeto **Texture2D** se puede configurar para diversos modos de límites y filtros de mapeo. Pero hay inlcuso más que esto.  La configuración adicional de una textura se hace a través de un componente del nodo **TextureAttributes**. La Figura 7-10 muestra un objeto visual con un manojo de apariencia con los componentes **Texture** y **TextureAttributes**.  Figura 7-10, Un Manojo de Apariencia con Componentes Texture y TextureAttributes.  Otras opciones de texturado van más allá de las configuraciones **Texture** y **TextureAttributes**. Por ejemplo, una textura puede ser tridimensional. En secciones posteriores veremos el API para la clase **Texture3d**, que, como **Texture2D**, es una extensión de la clase **Texture**.  **Opciones de Textura**  **Texture2D**, la clase usada en los ejemplos anteriores, es una extensión de **Texture**. Algunas de las opciones básicas para el texturado se implementan en esta clase. Puesto que **Texture** es una clase abstracta, sus configuraciones se harán a través de un objeto **Texture2D** o **Texture3d**. Las configuraciones son el modo de límites, filtros, y el formato de la textura.  **Modo de Límites: Envolver o Abrazar**  En todos los programas anteriores, se han asociado las texturas de una manera tal que una copia de la imagen se ha utilizado para cubrir el plano. El problema es qué hacer cuando una coordenada de textura está más allá del rango 0 a 1 del espacio de la textura y no fue direccionada.  La configuración del modo de límites determina lo que ocurre cuando el mapeo tiene lugar si las coordenadas de textura van más allá del rango 0 a 1 del espacio de la imagen. Las opciones son envolver la imagen, o abrazar la imagen. Envolver, significa repetir la imagen según sea necesario, es el valor por defecto. Abrazar utiliza el color del borde de la imagen en cualquier lugar fuera del rango 0 a 1. Estas configuraciones se hacen independientemente en las dimensiones s y t.  En el programa del ejemplo [BoundaryModeApp](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/BoundaryModeApp.java) la textura se asocia sobre aproximadamente la novena parte de cada uno de los planos. La Figura 7-11 muestra la escena según lo renderizado por este programa. Las variaciones en las imágenes se deben solamente a la configuración de los modos de límites para los planos. De izquierda a derecha las configuraciones son (s luego t) WRAP y WRAP, CLAMP y WRAP, WRAP y CLAMP, y CLAMP y CLAMP.  Figura 7-11, Comparación de las Combinaciones  de Modo de Límites para un Plano Texturado.  Observa que al contrario que las aplicaciones anteriores que comparten el mismo objeto **Texture** entre cuatro objetos visuales, la textura se carga cuatro veces en esta aplicación. Esto es necesario puesto que cada uno de los objetos **Texture** tiene diversas combinaciones de las configuraciones del modo de límites.  **Especificación de Filtrado**  En el cálculo de las coordenadas de textura para cada pixel, raramente hay una correspondencia del pixel directamente a un sólo texel. Normalmente un pixel es del tamaño de varios texels o más pequeño que un texel. En el primer caso se utiliza un filtro de ampliación para asociar varios texels a un pixel. En el segundo caso se utiliza un filtro de reducción para asociar el texel o los texels a un pixel. Hay opciones para manejar cada uno de estos casos.  El filtro de ampliación especifica qué hacer cuando un pixel es más pequeño que un texel. En este caso la textura será ampliada como si se aplicara sobre la geometría. Cada texel aparecerá como varios pixels y es posibles que la imagen resultante exhiba el "texelization" donde se verían los texels individuales para la renderización. Las opciones para el filtro de ampliación son hacer el punto de muestreo, que es seleccionar el texel más cercano y utilizar su color, o interpolarlo entre texels vecinos. El punto de muestreo, o muestreo del vecino más cercano, generalmente tiene menor coste de cálculo; mientras que el muestreo linear de la interpolación cuesta más (en cálculo y por lo tanto en tiempo de renderización) pero reduce el aspecto de cualquier texelization.  El filtro de reducción especifica qué hacer cuando un pixel es más grande que un texel. En este caso los texels deben ser "reducidos" para caber en el pixel. El problema trata en que un pixel puede solamente tener un valor de color y varios texels podrían suministrarlo. Las opciones para el filtro de reducción son hacer el punto de muestreo, que es seleccionar el texel más cercano y utilizar su color, o interpolarlo entre texels vecinos.  No está siempre claro qué filtro será utilizado. Consideremos una textura estirada en una dirección pero aplastada en otra. Dependiendo de qué dimensión see considera, se escogerá un filtro diferente. No hay nada que el programador pueda hacer para determinar cuál será utilizado. Sin embargo, el sistema de ejecución normalmente escoge el filtro que resulte en una imagen mejor.  La selección de un filtro tiene otras implicaciones con respecto al uso del color de límite (siguiente sección). También, las opciones del filtro de reducción son más complejas cuando se usan varios niveles de texturado.  **Color de Limites**  El comportamiento del modo de límites es además configurable con un color del límite. Cuando el modo de límite es CLAMP y se especifica color de límite, se utiliza el color del límite cuando las coordenadas de la textura están fuera del rango 0 a 1. Solo se puede especificar un color de límite para utilizar un color en cada dimensión para la cual el modo de límite se fije a CLAMP. El programa de ejemplo [BoundaryColorApp](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/BoundaryColorApp.java) demuestra esta característica.  En la Figura 7-12 se han fijado colores de límite para los cuatro planos. El plano más a la izquierda no utiliza su color de límite porque sus modos de límite son ambos WRAP. Para el plano siguiente utiliza el color negro de límite sólo en la dimensión vertical debido a los modos de límite. Podemos ver la mezcla entre el azul y el negro a la izquierda; en el lado derecho de la imagen el color negro del límite se mezcla con el borde blanco de la textura. Los colores de límite para los dos planos restantes son verde y rojo. Ambos modos de límite son CLAMP para el plano de la derecha de la imagen.  Figura 7-12, Imagen Producida por BoundaryColorApp.  Observa que el color de límite no se utiliza si el filtro es BASE\_LEVEL\_POINT. Para que el color del límite sea utilizado, el filtro necesita ser por lo menos BASE\_LEVEL\_LINEAR. El corolario es que siempre que el filtro no sea BASE\_LEVEL\_POINT se utiliza el **BoundaryColor**.  También observa que la misma textura se carga cuatro veces en esta aplicación. Un objeto **Texture** no se puede compartir entre los cuatro planos en esta aplicación puesto que cada objeto de textura se configura con una combinación distinta de modos de límite.  **Formato de Textura**  La última configuración de la clase **Texture** es la del formato de textura. El formato de textura es una declaración de cuántos valores hay por texel y cómo esos valores afectan a los pixels. Por ejemplo, una configuración del formato de textura de INTENSITY indica que solo el valor del texel será utilizado para rojo, verde, azul, y los valores de alpha del pixel. Una configuración del formato de textura de RGB indica que los tres valores del texel serán utilizados para los valores rojo, verde, y azul del pixel mientras que el valor de alpha del pixel sigue siendo igual.   |  |  |  |  | | --- | --- | --- | --- | | **Formato de Textura** | **Valores por Texel** | **Modifica Color del Pixel** | **Modifica alpha del Pixel** | | INTENSITY | 1 | si, R=G=B | si, R=G=B=A | | LUMINANCE | 1 (sólo color) | si, R=G=B | no | | ALPHA | 1 (sólo alpha) | no | si | | LUMINANCE\_ALPHA | 2 | si, R=G=B | si | | RGB | 3 | si | no | | RGBA | 4 | si | si |   **Texture3d**  Como el nombre implica, un objeto **Texture3d** contiene una imagen tridimensional de la textura. Puede ser que pensemos en él como un volumen de color. La clase **Texture3d** es una extensión de **Texture**, así que todas las características de la clase **Texture** se aplican a **Texture3d**. La única característica que **Texture3d** tiene y que **Texture2D** no tiene, es una especificación para el modo de límite de la tercera dimensión, o la dimensión r.  **Algunas Aplicaciones de Texturado**  Créamoslo o no, hay muchas más características texturado a explicar. Sin embargo, podemos utilizar las características ya discutidas en muchas aplicaciones. Esta sección hace una parada en la discusión de los detalles de texturado para mostrar dos aplicaciones de texturado. Una aplicación aplica una textura a un primitivo geométrico (véase el [Capítulo 2](http://www.labvis.unam.mx/elio/J3D/cursos/3d/cap_2.html)). Otra textura las líneas de polígonos no-rellenos. Una tercera aplicación utiliza la textura creada por un **Text2D** (véase el [Capítulo 3](http://www.labvis.unam.mx/elio/J3D/cursos/3d/cap_3.html)) a otro objeto visual.  **Texturado de Geométricos Primitivos**  Una forma para simplificar el proceso de presentar una textura es utilizar un primitivo geométrico. Se puede utilizar una bandera para asignar automáticamente las coordenadas de textura al crear primitivos geométricos. El [Fragmento de Código 7-6](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-6) muestra el uso de un constructor para una esfera primitiva con la generación de coordenadas.  **Fragmento de Código 7-6, Crear un Esfera Primitiva con Pre-asignación de Coordenadas de Textura.**  1. objRoot.addChild(new Sphere(1.0f, Primitive.GENERATE\_TEXTURE\_COORDS, appear));  La línea del [Fragmento de Código 7-6](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-6) se utiliza en el programa de ejemplo [PrimitiveTextureApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/PrimitiveTextureApp.java). Este programa textura una esfera con la imagen **earth.jpg**, que también está en el fichero jar de ejemplo, dando por resultado la imagen de al Figura 7-13.  Figura 7-13, Texturado de una Esfera Geométrica Primitiva.  **Texturado de Líneas**  Los polígonos no son los únicos elementos gráficos que pueden ser texturados; las líneas también se pueden texturar. El programa [TexturedLinesApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TexturedLinesApp.java) lo demuestra, usando una textura 1-D para texturar líneas de un objeto visual. En esta aplicación, el objeto visual es una tira torcida creada en el capítulo 2. El único 'truco' para texturar líneas es crear el manojo de apariencia apropiado para visualizar las líneas de la geometría y de los polígonos no rellenos. El [Fragmento de Código 7-7](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-7) muestra las líneas del código para agregar el componente **PolygonAttributes** a un manojo de apariencia para visualizar las líneas.  **Fragmento de Código 7-7, Crear un Manojo de Appearance para Mostrar Líneas en un Geometry Array.**  1. PolygonAttributes polyAttrib = new PolygonAttributes();  2. polyAttrib.setCullFace(PolygonAttributes.CULL\_NONE);  3. polyAttrib.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  4. twistAppear.setPolygonAttributes(polyAttrib);  Una textura unidimensional es realmente un objeto **Texture2D** con una dimensión (generalmente t) con tamaño 1. Para el programa de ejemplo, la textura tiene 16 texels por 1 texel. Las coordenadas de dos dimensiones de la textura se asignan al objeto visual. El valor-t de cada coordenada de textura se fija a 0.0f. Sin embargo, se podría utilizar cualquier valor-t y el resultado sería igual. La Figura 7-14 muestra la geometría de la tira torcida visualizada como líneas texturadas. El código fuente está en [TexturedLinesApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TexturedLinesApp.java)  Figura 7-14, Líneas Texturadas en Java 3D.  **Usar Texturas Text2D**  Los objetos **Text2D** crean texturas de texto especificado y aplican la textura a un polígono. A esta textura se puede acceder fácilmente desde **Text2D** y aplicarla a otro objeto visual. La Figura 7-15 muestra una imagen producida por el programa [Text2DTextureApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/Text2DTextureApp.java), un programa que aplica la textura creada por otro objeto **Text2D** mostrándola en el fondo de la geometría de otro objeto visual.  Figura 7-15, La Textrua de un Objeto Text2D aplicada a otro Objeto visual.  **Atributos de Textura**  En secciones anteriores vimos algunas de las opciones disponibles en texturado. El componente **TextureAttributes** permite la personalización posterior del texturado. Las configuraciones de los atributos de textura incluyen el modo de textura, color de mezcla, modo de corrección de perspectiva, y una correspondencia del mapeo de la textura. Los valores por defecto para estas configuraciones son **REPLACE**, **black**, **FASTEST**, y **NONE**, respectivamente. Además, el método setEnable permite activar y desactivar el mapeo de la textura. Todas las configuraciones se explican en esta sección.  Una ventaja de tener controladas las características del texturado por un componente diferente del nodo es la capacidad de compartir una textura entre objetos visuales pero aún así poder personalizarla para cada objeto visual. La Figura 7-10 muestra dos objetos visuales que comparten un solo objeto **Texture**. Cada uno de los objetos visuales personaliza la textura con el componente **TextureAttributes** en su manojo de apariencia.  Figura 7-16, Dos objetos visuales compartiendo la misma Textura Personalziada por Componentes TextureAttributes.  Los objetos **TextureAttributes** se añaden al escenario gráfico como miembros de una manejo de apariencia. El método setTextureAttributes de **Appearance** se muestra en el siguiente bloque de referencia.   |  | | --- | | Método setTextureAttributes de la Clase **Appearance**  void setTextureAttributes(TextureAttributes textureAttributes)  Selecciona el objeto textureAttributes en un objeto appearance. |   **Modo de Textura**  Para apreciar el rol del modo de textura debemos entender la secuencia de las operaciones implicadas en la determinación del color de un pixel. Indicado brevemente, primero se calcula el color de la no-textura de un pixel, y luego se aplica la textura. El color de no-textura lo determina el color de la geometría por vértice, **ColoringAttributes**, o la combinación de las características materiales y las condiciones de iluminación. Como hay varias formas de determinar el color de la no-textura, hay varias maneras posibles de combinar el color de la no-textura y el color de la textura.  La configuración del modo de textura es un factor importante en la determinación de cómo afecta el valor del texel (color y/o alpha) a los valores del color y del alpha del pixel de la no-textura. La operación de texturado real depende de la combinación del formato de la textura y del modo de textura. Puedes referirte a la especificación de Java 3D API (apéndice E) para más información.  El modo de textura por defecto es **REPLACE**, las otras opciones son **BLEND**, **DECAL**, y **MODULATE**. Cada uno de los modos se describe en las siguientes secciones. También podemos ver la tabla que resume los modos de la textura.  **Blend**  En el modo **BLEND**, el color de la textura se mezcla con el color de la no-textura. El color de la textura determina la cantidad del color de la no-textura a utilizar. La transparencia que resulta es la combinación de la transparencia de la textura y del material. Este modo determinado de la textura tiene la flexibilidad agregada de incluir opcionalmente un color de mezcla.  **Decal**  En el modo **DECAL**, el color de la textura se aplica como etiqueta encima del color de la no-textura. La transparencia de la textura determina la cantidad de color material a utilizar. La transparencia del pixel se deja sin cambios. Esto es totalmente igual que aplicar una etiqueta a un objeto del mundo real. El formato de textura debe ser RGB o RGBA para el modo de textura de **DECAL**.  **Modulate**  En el modo **MODULATE** el color de la textura se combina con el color de la no-textura. La transparencia que resulta es la combinación de la transparencia de la textura y del material. Puesto que el color que resulta depende de la no-textura y de los colores de la textura, este modo es útil en la aplicación de la misma textura a una gran variedad de objetos visuales sin hacer que todos parezcan iguales. Este modo se utiliza a menudo en escenas de iluminación.  **Replace**  En el modo **REPLACE** la textura proporciona el color y la transparencia para el pixel, no haciendo caso del resto de los colores excepto del color specular (si se permite la iluminación). Éste es el modo de textura por defecto incluso cuando no hay componente **TextureAttributes** en el manojo de apariencia.  **Sumario de Modos de Textura**  La Siguiente tabla ofrece un sumario de cada uno de los modos de textura. Esta tabla se ha pensado como una guía general para entender los distintos modos de textura disponibles. Los cálculos del color real se basan en una combinación del modo de textura y del formato de textura.  Cuando se permite la iluminación, los componentes de colores ambiente, difuso, y emisivo se ven afectados por la operación de textura; el color specular se ve afectado. El color specular se calcula basándose en las condiciones del material y de la iluminación, luego después de que la operación de la textura se aplique a otros componentes del color (no-texturados) el color specular se agrega a los otros colores que renderizan el color final del pixel.   |  |  |  |  | | --- | --- | --- | --- | | **Modo de Textura** | **Color de Pixel Derivado desde** | **Determinado por** | **Aplicación a...** | | BLEND | Color de Textura, no-textura, y color mezcla opcional | Color de textura | Escenas Iluminadas con Color de Mezcla | | DECAL | Color de Textura y color no-textura | alpha de la textura | detalles de superficie | | MODULATE | Colores de Textura y de no-textura | n/a | Escenas de iluminación | | REPLACE | Sólo color de textura (modo de textura por defecto) | n/a | Escenas sin iluminación |   **Textura con Color de Mezcla**  El color de mezcla se utiliza en texturado sólo cuando el modo de textura es **BLEND**. El color del pixel resultante es una combinación del color del texel y del color de mezcla. Con el color de mezcla se puede aplicar la misma textura con diferentes sombras a diferentes objetos visuales. El color de mezcla se expresa como un valor **RGBA**. El color de mezcla por defecto es (0,0,0,0) negro con un alpha de 0.  **Modo de Corrección de Perspectiva**  El mapeo de textura ocurre en el espacio de la imagen. Por esta razón los planos texturados pueden parecer incorrectos cuando se ven desde un lateral. Es decir, parecen incorrectos a menos que se haga una corrección de la perspectiva. En Java 3D la corrección de la perspectiva se hace siempre. La única opción es cómo hacer esta corrección de la perspectiva.  Las dos opciones son **FASTEST** y **NICEST**. Obviamente, el dilema es la velocidad clásica contra la calidad de la imagen. Para esta opción, la configuración del valor por defecto es **NICEST**.  **Transformación del Mapeo de Textura**  Dentro del componente **Attributes** de una textura se puede especificar un objeto **Transform3d** para alterar la función de mapeo de la textura. Esta correspondencia de transformación del mapeo de textura se puede utilizar para mover una textura sobre un objeto visual en tiempo de ejecución. La transformación traslada, rota, y escala las coordenadas de textura (s, t, r) antes de que los texels sean seleccionados desde la imagen de textura.  Una traslacción en la transformación de la textura desplazaría la textura a través del objeto visual. Se puede utilizar una transformación de rotación para reorientar la textura en un objeto visual. Se pueden utilizar una transformación de escala para repetir la textura a través de un objeto visual. Por supuesto, como un objeto **transform** puede contener una combinación de éstos, se puede animar la textura de un objeto visual manipulando este objeto.  **API TextureAttributes**  Los siguientes bloques de referencia listan los constructores, métodos y capacidades del componente **TextureAttributes**.   |  | | --- | | Sumario de Constructores de la Clase **TextureAttributes**  Extiende: NodeComponent  El objeto **TextureAttributes** define los atributos que se aplican a un mapeo de textura.  TextureAttributes()  Construye un objeto TextureAttributes con estos valores por defecto:  texture mode : REPLACE, transform : null, blend color : black (0,0,0,0), perspective correction: NICEST  TextureAttributes(int textureMode, Transform3d transform,  Color4f textureBlendColor, int perspCorrectionMode)  Construye un objeto TextureAttributes con los valores especificados. |  |  | | --- | | Constantes de la Clase **TextureAttributes**  Estas constantes se usan en los constructores y métodos para seleccionar los modos de textura y de corrección de la perspectiva.  **Constantes de Modo de Textura**   * **BLEND** Mezcla el color de mezcla con el color del objeto. * **DECAL** Aplica el color de la textura al objeto como una etiqueta. * **MODULATE** Modula el color del objeto con el color de la textura. * **REPLACE** Reemplaza el color del objeto con el color de la textura.   **Constantes del Modo de Corrección de Perspectiva**   * **FASTEST** Usa el método más rápido disponible para la correción de perspectiva del mapeo de textura. * **NICEST** Usa el mejor método (de mayor calidad) disponible para la correción de perspectiva del mapeo de textura. |  |  | | --- | | Sumario de Métodos de la Clase **TextureAttributes**  void getTextureBlendColor(Color4f textureBlendColor)  Obtiene el color de mezcla de la textura para este objeto appearance.  void getTextureTransform(Transform3d transform)  Recupera una copia del objeto transformation de la textura.  void setPerspectiveCorrectionMode(int mode)  Selecciona el modo de corrección de la perspectiva a usar para la interpolación de coorednadas de color y/o textura a uno de :   * **FASTEST** Usa el método más rápido disponible para la correción de perspectiva del mapeo de textura. * **NICEST** Usa el mejor método (de mayor calidad) disponible para la correción de perspectiva del mapeo de textura.   void setTextureBlendColor(Color4f textureBlendColor)  void setTextureBlendColor(float r, float g, float b, float a)  Selecciona el color de mezcla para este objeto TextureAttributes.  void setTextureMode(int textureMode)  Selecciona el parámetro del modo de textura a uno de:   * **BLEND** Mezcla el color de mezcla con el color del objeto. * **DECAL** Aplica el color de la textura al objeto como una etiqueta. * **MODULATE** Modula el color del objeto con el color de la textura. * **REPLACE** Reemplaza el color del objeto con el color de la textura.   void setTextureTransform(Transform3d transform)  Selecciona el objeto transform usado para transformar las coordenadas de textura. |  |  | | --- | | Sumario de capacidades de la Clase **TextureAttributes**   * **ALLOW\_BLEND\_COLOR\_READ | WRITE**  Permite leer (escribir) el color de mezcla de la textura * **ALLOW\_MODE\_READ | WRITE**  Permite leer (escribir) los modos de textura y de correción de perspectiva. * **ALLOW\_TRANSFORM\_READ | WRITE**  Permite leer (escribir) el objeto transform de la textura. |   **Generación Automática de Coordenadas de Textura**  Según se explicó anteriormente, asignar coordenadas de textura a cada vértice de la geometría es un paso necesario en el texturado de objetos visuales. Este proceso puede comsumir mucho tiempo así como es difícil para objetos visuales grandes y/o complejos. Debemos tener presente que esto es un problema para el programador y una vez solucionado, no es un problema que se repita.  A menudo las coordenadas de textura se asignan con el código específico de un objeto visual. Sin embargo, otra solución es automatizar la asignación de las coordenadas de textura mediante algún método. Este método se podía utilizar para cualquier objeto visual tanto si es grande o pequeño, complejo o simple. Este acercamiento es exactamente lo que lo hace un objeto **TexCoordGeneration** (generación de coordenadas de textura). Siempre que un objeto se cargue desde un fichero o sea creado en el código del programa, se puede utilizar un objeto **TexCoordGeneration** para asignar coordenadas de textura.  **TexCoordGeneration** es una clase del corazón del API Java 3D usada para generar coordenadas de textura. Para generar automáticamente coordenadas de textura, el programador especifica los parámetros de la coordenada de textura en un objeto **TexCoordGeneration** y agrega este objeto al manojo de apariencia del objeto visual. Las coordenadas de textura se calculan basándose en los parámetros de especificación de coordenadas en el tiempo de ejecución. Los parámetros se explican en las secciones siguientes.  **Formato de Generación de Textura**  Esta selección simplemente especifica si las coordenadas de textura serán generadas para una textura de dos o tres dimensiones. Las selecciones posibles son **TEXTURE\_COORDINATE\_2** y **TEXTURE\_COORDINATE\_3** que generan coordenadas de textura 2D (S y T) y coordenadas de textura 3D (S, T, y R), respectivamente.  **Modo de Generación de Textura**  Hay dos aproximaciones básicas para la generación de textura: proyección linear y mapeo esférico:  **Poyección Linear**  Con la proyección linear, las coordenadas de textura se especifican con planos. Para las coordenadas de textura de dos dimensiones (s,t), se utilizan dos planos. La distancia desde un vértice a un plano es la coordenada de textura en una dimensión; la distancia desde el otro plano a un vértice es la coordenada de textura en la otra dimensión. Para las texturas tridimensionales, se utilizan tres planos.  Los tres parámetros planos posibles se nombran planeS, planeT, y planeR, donde el nombre corresponde a la dimensión para la cual se utiliza. Cada plano se especifica como 4-tuple (ecuación plana). Los primeros tres valores son el vector normal superficial para el plano. El cuarto valor especifica la distancia desde el origen al plano a lo largo de un vector paralelo al vector normal superficial del plano.  Hay dos variaciones en este método automático de generación de coordenadas de textura. El primero, llamado objeto linear, produce coordenadas de textura estáticas. Con coordenadas de textura generadas linearmente, si el objeto visual se mueve, los coordenadas de textura no cambian. La segunda opción, llamada ojo linear, produce coordenadas de textura relativas a las coordenadas del ojo dando como resultado coordenadas de textura variables. Con coordenadas de textura linear del ojo los objetos que se mueven parecen moverse con la textura.  La Figura 7-17 muestra las imágenes producidas por un programa de ejemplo que utiliza un objeto **TexCoordGeneration** para asignar coordenadas de textura a una tira torcida. En esta aplicación se usa una textura unidimensional. La textura tiene un solo texel rojo en un extremo. Cuando la aplicación se ejecuta, la tira torcida rota.  La imagen de la izquierda de la Figura 7-17 muestra el texturado con modo de generación **OBJECT\_LINEAR**. En este caso la textura rota con el objeto y se puede ver el texel rojo rotar con la tira. La imagen de la derecha de la Figura 7-17 muestra la textura que resulta cuando el modo la generación es **EYE\_LINEAR** para la tira torcida. En este caso, el texel rojo permanece en el centro de la vista mientras que el objeto rota.  Figura 7-17, Comparar los modos de Generación del objeto TexCoordGeneration.  [TexCoordGenApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/TexCoordGenApp.java) es el programa que produce estás imágenes.  **Mapeo Esférico**  Si un objeto brillante está en el centro de una habitación real, probablemente reflejaría la imagen de muchos de los otros objetos de la habitación. Las reflexiones dependerían de la forma del objeto y de la orientación de las cosas en la habitación. El modo de generación de las coordenadas del mapeo esférico está diseñado para asignar coordenadas de textura para aproximar las reflexiones de otros objetos sobre el objeto visual como sucedería para el objeto brillante en el mundo real del ejemplo.  Cuando se usa un objeto **TexCoordGeneration** en el modo de mapeo esférico el cálculo de las coordenadas de textura se basa en las superficies normales y en la dirección de la vista.  La textura usada para este efecto debe estar especialmente preparada. Si el ambiente virtual del objeto brillante existe en el mundo real, una fotografía de la escena tomada con una lente de ojo de pez creará una imagen de textura conveniente. Si no existe la escena, entonces la textura se debe crear para parecer que la imagen es una fotografía tomada con una lente de ojo de pez.  **Cómo usar un Objeto TexCoordGeneration**  Para usar un objeto **TexCoordGeneration**, lo seleccionamos como un componente del manojo de apariencia del objeto visual a texturar. La Figura 7-18 muestra el diagrama de una manojo de apariencia con un objeto **TexCoordGeneration** junto con un objeto **Texture** y otro objeto **TextureAttributes**.  Figura 7-18, Manojo de Appearance con Texture, TextureAttributes, y TexCoodGeneration.   |  | | --- | | Método setTexCoordGeneration de la Clase **Appearance**  void setTexCoordGeneration(TexCoordGeneration texCoordGeneration)  Selecciona el objeto texCoordGeneration al objeto especificado. |   **API TexCoordGeneration**  Los siguientes bloques de referencia listan los constructores, constantes, métodos y capacidades de los objetos de la clase **TexCoordGeneration**.   |  | | --- | | Sumario de Constructores de la Clase **TexCoordGeneration**  El objeto **TexCoordGeneration** contiene todos los parámetros necesarios para generar coordenadas de textura. Está incluido como parte de un objeto **Appearance**.  TexCoordGeneration()  Construye un objeto TexCoordGeneration usando los valores por defecto para todas las edades.  TexCoordGeneration(int genMode, int format)  Construye un objeto TexCoordGeneration con genMode y format especificados.  TexCoordGeneration(int genMode, int format, Vector4f planeS)  TexCoordGeneration(int genMode, int format, Vector4f planeS, Vector4f planeT)  TexCoordGeneration(int genMode, int format, Vector4f planeS, Vector4f planeT, Vector4f planeR)  Construyen un objeto TexCoordGeneration con genMode, format, y las ecuaciones de los planos especificados. |  |  | | --- | | Sumario de Campos de la Clase **TexCoordGeneration**  **Constantes de Modo de Generación**   * **EYE\_LINEAR** Genera coordenadas de textura como una función linear en coordenadas de ojo (por defecto). * **OBJECT\_LINEAR** Genera coordenadas de textura como una función linear en coordenadas del objeto . * **SPHERE\_MAP** Genera coordenadas de textura usando un mapeo de reflexión esférica en coordenadas de ojo.   **Constantes de Formato**   * **TEXTURE\_COORDINATE\_2** Genera coordenadas de textura 2D (S y T) (por defecto) * **TEXTURE\_COORDINATE\_3** Genera coordenadas de textura 3D (S, T, y R) |  |  | | --- | | Sumario de Métodos de la Clase **TexCoordGeneration**  void setEnable(boolean state)  Activa o desactiva la generación de coordenadas para este objeto appearance.  void setFormat(int format)  Selecciona el formato TexCoordGeneration al valor especificado.  void setGenMode(int genMode)  Selecciona el modo de generación de TexCoordGeneration al valor especificado.  void setPlaneR(Vector4f planeR)  Selecciona la ecuación plana de la coordenada R.  void setPlaneS(Vector4f planeS)  Selecciona la ecuación plana de la coordenada S.  void setPlaneT(Vector4f planeT)  Selecciona la ecuación plana de la coordenada T. |  |  | | --- | | Sumario de Capacidades de la Clase **TexCoordGeneration**   * **ALLOW\_ENABLE\_READ | WRITE**  Permite leer/escribir su bandera de enable. * **ALLOW\_FORMAT\_READ**  Permite leer escribir su información de formato. * **ALLOW\_MODE\_READ**  Permite leer su información de modo. * **ALLOW\_PLANE\_READ**  Permite leer la información de componentes planeS, planeR, y planeT. |   **Múltiples Niveles de Textura (Mipmaps)**  Para entender la razón de los múltiples niveles de textura, consideremos una aplicación que contenga un objeto visual texturado que se mueva alrededor la escena (o que lo mueva el espectador). Cuando este objeto visual está cerca del espectador aparecen demasiados pixels en la imagen. Para este caso, se debería utilizar una textura de buen tamaño para evitar la visión de texels individuales; esto es especialmente cierto cuando se utiliza el punto de muestreo como filtro de ampliación.  Sin embargo, cuando este objeto visual se ve en la distancia, la textura será demasiado grande para el objeto visual y la textura se reducirá durante la representación. (Recordamos que el mapeo de textura tiene lugar durante la renderización en la imagen, la pantalla, o el espacio). El punto de muestreo para el filtro de reducción problablemente no dará resultados satisfactorios cuando el objeto visual sea 1/32 o más pequeño (tamaño del pixel) que la resolución de la textura. El dilema es calidad de la imagen contra rendimiento de renderización.  Si en vez de usar un gran mapeo de textura (porque el objeto visual aparecerá grande) se usa uno pequeño para hacer que la vista del objeto sea mejor cuando es pequeño, existe el problema inverso. Para las imágenes de buena calidad el filtro de ampliación implicará la interpolación linear dando por resultado más cálculo. De nuevo, el dilema está entre la calidad de imagen contra el rendimiento de la renderización. La única ventaja de usar un mapeo de textura más pequeño es un menor requerimiento de memoria para almacenar la textura.  Lo que se necesita es un mapeo de textura pequeño cuando el objeto visual aparece pequeño y un mapeo de textura grande cuando el objeto visual aparece grande. Las técnicas de texturado actuales que usan una imagen de textura, llamada texturado base, no pueden hacer esto. Y esto es exactamente lo que proporcionan los múltiples niveles de textura.  **¿Qué es el Texturado Multi-Nivel (MIPmap)?**  Los múltiples niveles de textura se refieren a una técnica de texturado donde se utilizan juntas una serie de imágenes de textura como textura para los objetos visuales. La serie de imágenes es (generalmente) la misma textura en una variedad de resoluciones. Cuando un objeto visual se está renderizando con múltiples niveles de textura, se utiliza la imagen de textura que está más cercana al tamaño de pantalla del objeto visual.  El funcionamiento del renderizador depende de los filtros de reducción y ampliación usados. Sin embargo, con **MIPmaps** tenemos más control sobre la apariencia de los objetos visuales y podemos conseguir objetos visuales de mejor aspecto con un mejor rendimiento.  Usar múltiples niveles de textura es como usar un objeto **DistanceLOD** (véase el [Capítulo 5](http://www.labvis.unam.mx/elio/J3D/pagina35.html)) para aplicar diversas texturas a un objeto visual cuando se ve desde diversas distancias. Las excepciones son que con el **Mipmap** el objeto visual siempre es texturado mientras que con el objeto **DistanceLOD**, el objeto podría no ser texturado a algunas distancias. Y, para los objetos visuales texturados en todas las distancias, el **MIPmap** es más eficiente y ha agregado posibilidades de filtrado con respecto a un objeto **DistanceLOD** usado para una aplicación similar.  Los múltiples niveles la textura son referidos comúnmente como mipmap. El término "**MIPmap**" viene de las siglas del Latin **"multum in parvo"**, que significa muchas cosas en un lugar pequeño. El término **MIPMap** realmente se refiere a una técnica específica de almacenaje para almacenar una serie de imágenes para el uso en texturado de múltiples niveles El término **MIPmap** se utiliza comúnmente para significar texturado de múltiples niveles.  Con la técnica de almacenaje **MIPmap**, el tamaño de una imagen de textura es ¼ del tamaño anterior (½ del tamaño en cada cada dimensión). Esto continúa hasta que el tamaño de la imagen más pequeña es de 1 texel por 1 texel. Por ejemplo, si tamaño máximo de la textura es 16x4, las texturas restantes son 8x2, 4x1, 2x1, y 1x1. La Figura 7-19 muestra los niveles de textura para la textura **stripe.gif**, Cada una de estas imágenes de textura fue preparada usando software de edición de imágenes.  Figura 7-19, Múltiples Niveles de una Textura (Tamaños de imágenes: 128x128, 64x64, 32x32, ..., 1x1)  La Figura 7-20 muestra una imagen de un solo plano texturado con una textura múltiple donde cada nivel de textura tiene un color diferente. El plano se orienta en ángulo al espectador para que el lado izquierdo esté más cercano al espectador que el derecho. Debido a la proyección de la perspectiva el lado izquierdo del plano parece más grande en coordenadas de la imagen que el derecho.  Debido a la orientación y a la proyección del plano, los pixels representan menos área superficial (en el sistema virtual de coordenadas del objeto) a la izquierda y progresivamente más área superficial hacia a la derecha, dando como resultado que el nivel de textura cambia. A la izquierda del plano en la imagen, se utiliza el nivel base de la textura. Los cambios del color en la imagen indican donde ocurrieron los cambios del nivel de textura durante la renderización.  Tener una textura para cada nivel de un color diferene no es la aplicación típica de texturado múltiple. Esta aplicación simplemente iluistra la operación de una textura múltiple.  Figura 7-20, La imagen Generada por un Plano texturado con una Textura MipMap multi-color.  La Figura 7-20 está generada por el programas [MIPmapDemo.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/MIPmapDemo.java).  **Ejemplos de Texturas Multi-Nivel**  En lo que concierte a programación con el API Java 3D, crear una textura de niveles múltiples es casi igual que crearla de un solo nivel, o nivel base. Mirando hacia la receta texturado simple la única diferencia es que se necesitan varias imágenes de textura para la textura de niveles múltiples. Hay dos maneras de crear los niveles múltiples de las imágenes de la textura. Una forma es crear cada imagen a mano con las aplicaciones apropiadas de edición/creacción de imágenes, la otra es utilizar una característica del cargador de textura para crear esas imágenes desde la imagen base.  Las dos técnicas de texturado de nivel múltiple ocupan una cantidad casi igual de código. La de menos cantidad de trabajo es la de generar las imágenes de los niveles desde la imagen base. El [Fragmento de Código 7-8](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-8) presenta el código de carga de las texturas de [MIPmapApp.java](http://www.labvis.unam.mx/elio/J3D/cursos/3d/clases/texture/MIPmapApp.java). Esta aplicación es un ejemplo de como generar niveles múltiples de textura desde una imagen base.  **Fragmento de Código 7-8, Crear Texturado Multi-nivel desde una sola imagen base.**    1. Appearance appear = new Appearance();  2.  3. NewTextureLoader loader = new NewTextureLoader("stripe.gif",  4. TextureLoader.GENERATE\_MIPMAP);  5. ImageComponent2D image = loader.getImage();  6.  7. imageWidth = image.getWidth();  8. imageHeight = image.getHeight();  9.  10. Texture2D texture = new Texture2D(Texture.MULTI\_LEVEL\_MIPMAP,  11. Texture.RGB, imageWidth, imageHeight);  12. imageLevel = 0;  13. texture.setImage(imageLevel, image);  14.  15. while (imageWidth > 1 || imageHeight > 1){ // loop until size: 1x1  16. imageLevel++; // compute this level  17.  18. if (imageWidth > 1) imageWidth /= 2; // adjust width as necessary  19. if (imageHeight > 1) imageHeight /= 2; // adjust height as necessary  20.  21. image = loader.getScaledImage(imageWidth, imageHeight);  22. texture.setImage(imageLevel, image);  23. }  24.  25. texture.setMagFilter(Texture.BASE\_LEVEL\_POINT);  26. texture.setMinFilter(Texture.MULTI\_LEVEL\_POINT);  27.  28. appear.setTexture(texture);  El [Fragmento de Código 7-8](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-8) empieza siguiendo los mismos pasos que son utilizados para cualquier aplicación de textura cargando la imagen base. Una diferencia es que el **TextureLoader** se crea con la bandera **GENERATE\_MIPMAP** (líneas 3-4). Entonces se extrae la imagen base del cargador de la forma usual.  Las dimensiones de esta imagen son necesarias no sólo para crear el objeto **Texture2D**, sino también para calcular los tamaños de las siguientes imágenes. Por esta razón se guardan en dos variables (líneas 7 y 8). Estas variables serán utilizadas durante la generación y carga de las imágenes restantes.  El objeto **Texture2D** se crea usando el modo MIPmap **MULTI\_LEVEL\_MIPMAP** y la dimensión de la imagen base (líneas 10 y 11). El nivel base es el nivel 0. Entonces el número de nivel se graba y se selecciona la imagen base como la imagen para el nivel 0 (líneas 12 y 13).  El bucle itera hasta que el tamaño de la imagen sea de 1 pixel por 1 pixel (línea 15). El número de nivel se incrementa en cada iteración (línea 16) y se calcula la dimensión de la imagen (líneas 18 y 19). La imagen apropiadamente escalada se consigue desde **TextureLoader** (línea 21) y se selecciona para el nivel actual en el objeto **Texture2D** (línea 22).  Cuando se crea un mapeo de textura de múltiples niveles debemos asegurarnos de seleccionar filtros de nivel como se hace en las líneas 25 y 26 del [Fragmento de Código 7-8](http://www.labvis.unam.mx/elio/J3D/cursos/3d/#fragmento7-8). Las selecciones de filtrado desactivan el nivel de texturado múltiple.  Crea las imágenes a mano permite una calidad superior y/o poder añadir efectos especiales. Las imágenes generadas se producen filtrando la imagen base.  **Fragmento de Código 7-9, Múltiples Niveles de Textura Cargados desde Ficheros de Imagenes Individuales.**  1. Appearance appear = new Appearance();  2.  3. String filename = "stripe.gif"; // filename for level 0  4. NewTextureLoader loader = new NewTextureLoader(filename);  5. ImageComponent2D image = loader.getImage();  6.  7. imageWidth = image.getWidth();  8. imageHeight = image.getHeight();  9.  10. Texture2D texture = new Texture2D(Texture.MULTI\_LEVEL\_MIPMAP,  11. Texture.RGBA,imageWidth, imageHeight);  12. imageLevel = 0;  13. texture.setImage(imageLevel, image);  14.  15. while (imageWidth > 1 || imageHeight > 1){ // loop until size: 1x1  16. imageLevel++; // compute this level  17.  18. if (imageWidth > 1) imageWidth /= 2; // adjust width as necess.  19. if (imageHeight > 1) imageHeight /= 2;// adjust height as necess.  20. filename = "stripe"+imageWidth+".gif";// file to load  21.  22. loader = new NewTextureLoader(filename);  23. image = loader.getImage();  24.  25. texture.setImage(imageLevel, image);  26. }  27.  28. texture.setMagFilter(Texture.BASE\_LEVEL\_POINT);  29. texture.setMinFilter(Texture.MULTI\_LEVEL\_POINT);  30.  31. appear.setTexture(texture);  **Filtros de Reducción para Múltiples Niveles de Textura**  Además de los dos filtros de nivel base, hay dos opciones de múltiples filtros para la configuración del filtro de reducción. Estas configuraciones adicionales son **MIPMAP\_POINT**, y **MIPMAP\_LINEAR**. Como con las otras configuraciones de filtro, el filtro de punto es probable que cree imágenes más rápidas pero de una calidad más baja con respecto al filtro linear.  Recuerda, cuando usamos múltiples niveles de textura, debemos seleccionar uno de los filtros de múltiples niveles para el filtro de reducción para utilizar otros niveles distintos del nivel base. Estas configuraciones adicionales de filtro no se aplican a las configuraciones de filtro de ampliación puesto que la ampliación de la textura se haría solamente en el nivel base.  **Modo Mipmap**  El modo **MIPmap** de la clase **Texture** es realmente una elección entre varios niveles de textura y un sólo nivel de textura. Las dos selecciones son **BASE\_LEVEL** y **MULTI\_LEVEL\_MIPMAP**.  **API de Texture, Texture2D, y Texture3d**  Muchas de las secciones precedentes presentan algunas porciones de las clases **Texture**, **Texture2D**, o **Texture3d**. Puesto que estas clases se han descrito en muchas secciones, el API de estas clases se presenta en esta sección.  **Texture** es la clase base para **Texture2D** y **Texture3d**. La clase **Texture** proporciona la mayoría del interfaz para las clases **Texture2D** y **Texture3d** incluyendo texturado multi-nivel. La siguiente tabla presenta un resumen de las características de estas tres clases. Para cada opción de texturado la tabla lista la clase que proporciona al interface, el método (set) para cambiar la configuración y el valor por defecto   |  |  |  |  | | --- | --- | --- | --- | | **Característica/Selección** | **Clase** | **Métodos set** | **Valor Defecto** | | Texture Image | Texture | setImage() | null | | Image Format | Texture | (ver constructores) | none | | Mipmap Mode | Texture | setMipMapMode() | BASE\_LEVEL | | Minification Filter | Texture | setMinFilter() | BASE\_LEVEL\_POINT | | Magnification Filter | Texture | setMagFilter() | BASE\_LEVEL\_POINT | | Boundary Modes | Texture  Texture2D  Texture3d | setBoundaryModeS() setBoundaryModeT() setBoundaryModeR() | WRAP WRAP WRAP | | BoundaryColor | Texture | setBoundaryColor() | black |   **Filtros de Reducción y Ampliación**  Según lo discutido anteriormente hay configuraciones separadas de filtros para reducción y ampliación. Las opciones de ampliación son: **BASE\_LEVEL\_POINT**, **BASE\_LEVEL\_LINEAR**, **FASTEST**, o **NICEST**. El filtro será **BASE\_LEVEL\_POINT** cuando se especifique **FASTEST** y **BASE\_LEVEL\_LINEAR** cuando se especifique **NICEST**.  Las opciones de reducción son: **BASE\_LEVEL\_POINT**, **BASE\_LEVEL\_LINEAR**, **MULTI\_LEVEL\_POINT**, **MULTI\_LEVEL\_LINEAR**, **FASTEST**, o **NICEST**. Las opciones de filtro de nivel base se pueden utilizar para las texturas de un sólo nivel o texturas de varios niveles. Los filtros reales usados cuando se especifica **FASTEST** o **NICEST** se implementan dependiendo de si se elige un filtro multi-nivel o una textura de múltiples niveles.  **API Texture**  Ahora que se han presentado todas las características de la textura, presentamos el API de la clase **Texture**. Como esta es una clase abtracta no hay bloque de referencia sobre sus constructores.   |  | | --- | | Sumario de Campos de la Clase **Texture**  El objeto **Texture** es un componente de un objeto **Appearance** que define las propiedades de la textura usada cuando se activa el mapeo de texturas. El objeto **Texture** es una clase abstracta y todos sus objetos se deben crear usando objetos **Texture2D** o **Texture3d**.  **Constantes de Formato**   * **ALPHA** Especifica la textura que sólo contiene valores Alpha. * **INTENSITY** Especifica la textura que sólo contiene valores Intensity. * **LUMINANCE** Especifica la textura que sólo contiene valores Luminance. * **LUMINANCE\_ALPHA** Especifica la textura que contiene valores Luminance y Alpha. * **RGB** Especifica la textura que contiene valores de color Red, Green y Blue. * **RGBA** Especifica la textura que contiene valores de color Red, Green, Blue y valor Alpha.   **Constantes de modo MIPMap**   * **BASE\_LEVEL** Indica que el objeto Texture sólo tiene un nivel. * **MULTI\_LEVEL\_MIPMAP** El objeto Texture tiene varios niveles - uno por cada nivel mipmap.   **Constantes de Filtro**   * **BASE\_LEVEL\_LINEAR** Realiza interpolación bilinear sobre los cuatro texels más cercanos en el nivel 0 del mapa de textura. * **BASE\_LEVEL\_POINT** Selecciona el texel más cernano en el nivel 0 del mapa de textura. * **MULTI\_LEVEL\_LINEAR** Realiza interpolación tri-linear sobre los cuatro texels más cercanos de los dos niveles de mipmap más cercanos. * **MULTI\_LEVEL\_POINT** Selecciona el texel más cercano en el mipmap más cercano.   **Constantes de Modo de Límites**   * **CLAMP** Encierra las coordenadas de textura para que estén en el rango [0, 1]. * **WRAP** Repite la envoltura envolviendo las coordenadas de textura que están fuera del rango [0,1].   **Constantes del Modo de Corrección de la Perspectiva**   * **FASTEST** Usa el método más rápido disponible para procesar la geometría. * **NICEST** Usa el método de mejor apariencia disponible para procesar la geometría. |  |  | | --- | | Sumario de Métodos de la Clase **Texture**  El objeto **Texture** es un componente de un objeto **Appearance** que define las propiedades de la textura usada cuando se activa el mapeo de texturas. El objeto **Texture** es una clase abstracta y todos sus objetos se deben crear usando objetos **Texture2D** o **Texture3d**.  ImageComponent getImage(int level)  Obtiene el nivel de mipmap especificado.  void setBoundaryColor(Color4f boundaryColor)  void setBoundaryColor(float r, float g, float b, float a)  Selecciona el color límite de la textura para este objeto.  void setBoundaryModeS(int boundaryModeS)  Selecciona el modo de límites para la coordenada S en este objeto texture.  void setBoundaryModeT(int boundaryModeT)  Selecciona el modo de límites para la coordenada T en este objeto texture.  void setEnable(boolean state)  Activa o desactiva el mapeo de textura para este objeto appearance.  void setImage(int level, ImageComponent image)  Selecciona un nivel de mipmap especificado.  void setMagFilter(int magFilter)  Selecciona la función de filtro de ampliación.  void setMinFilter(int minFilter)  Selecciona la función de filtro de reducción.  void setMipMapMode(int mipmapMode)  Selecciona el modo mipmap para el mapeo de textura de este objeto texture. |  |  | | --- | | Sumario de Capacidades de la Clase **Texture**   * **ALLOW\_BOUNDARY\_COLOR\_READ**  Permite leer su información de color de límite. * **ALLOW\_BOUNDARY\_MODE\_READ**  Permite leer su información de modo de límite. * **ALLOW\_ENABLE\_READ | WRITE**  Permite leer/escribir su bandera de enable. * **ALLOW\_FILTER\_READ**  Permite leer su información de filtro. * **ALLOW\_IMAGE\_READ**  Permite leer su información de componente imagen. * **ALLOW\_MIPMAP\_MODE\_READ**  Permite leer su información de modo de mipmap. |   **API de Texture2D**  **Texture2D** es una extensión concreta de la clase abstracta **Texture**. **Texture2D** sólo porporciona un constructor de interés. Todos los métodos usados con objetos **Texture2D** son métodos de **Texture**.   |  | | --- | | Sumario de Constructores de la Clase **Texture2D**  **Texture2D** es una subclase de la clase **Texture**. Extiende la clase **Texture** añadiendo un constructor.  Texture2D(int mipmapMode, int format, int width, int height)  Construye un objeto **Texture2D** vacío con los valores especificados de mipmapMode, format, width, y height. La imagen del nivel 0 la debe seleccionar la aplicación usando el método setImage. Si mipmapMode se selecciona a **MULTI\_LEVEL\_MIPMAP**, se deben seleccionar las imágenes para TODOS los niveles.  Parámetros:   * **mipmapMode** - tipo mipmap para este Texture: Uno de **BASE\_LEVEL**, **MULTI\_LEVEL\_MIPMAP**. * **format** - formato de datos de las texturas grabadas en el objeto. Uno de **INTENSITY**, **LUMINANCE**, **ALPHA**, **LUMINANCE\_ALPHA**, **RGB**, **RGBA**. * **width** - anchura de la imagen del nivel 0. Debe ser una potencia de 2. * **height** - altura de la imagen del nivel 0. Debe ser una potencia de 2. |   **API de Texture3d**  **Texture3D** es una extensión concreta de la clase abstracta **Texture**. **Texture3D** sólo porporciona un constructor de interés. Todos los métodos usados con objetos **Texture3D** son métodos de **Texture**.   |  | | --- | | Sumario de Constructores de la Clase **Texture3d**  **Texture3d** es una subclase de la clase **Texture**. Extiende la clase **Texture** añadiendo una tercera coordenada, un constructor y un método mutador para seleccionar una imagen de textura 3D.  Texture3d(int mipmapMode, int format, int width, int height, int depth)  Construye un objeto **Texture2D** vacío con los valores especificados de mipmapMode, format, width, height y depth. La imagen del nivel 0 la debe seleccionar la aplicación usando el método setImage. Si mipmapMode se selecciona a **MULTI\_LEVEL\_MIPMAP**, se deben seleccionar las imágenes para TODOS los niveles.  Parámetros:   * **mipmapMode** - tipo mipmap para esta Texture: Uno de **BASE\_LEVEL**, **MULTI\_LEVEL\_MIPMAP**. * **format** - formato de datos de las texturas grabadas en el objeto. Uno de **INTENSITY**, **LUMINANCE**, **ALPHA**, **LUMINANCE\_ALPHA**, **RGB**, **RGBA**. * **width** - anchura de la imagen del nivel 0. Debe ser una potencia de 2. * **height** - altura de la imagen del nivel 0. Debe ser una potencia de 2. * **depth** - profundidad de la imagen del nivel 0. Debe ser una potencia de 2. |  |  | | --- | | Sumario de Métodos de la Clase **Texture3d**  void setBoundaryModeR(int boundaryModeR)  Selecciona el modo de límite para la coordenada R de este objeto texture.  Parámetro:   * **boundaryModeR** - el modo de límite para la coordenada R, uno de: **CLAMP** o **WRAP**. |   **API de TextureLoader y NewTextureLoader**  Esta sección lista los bloques de referencia de las clases **TextureLoader** y **NewTextureLoader**.  La clase **NewTextureLoader** extiende la clase **TextureLoader** proporcionando una utilidad de cargador de texturas más fácil de utilizar -- una que no requiere un observador de imagen del AWT para cada constructor.  **API de TextureLoader**   |  | | --- | | Sumario de Campos de **TextureLoader**  GENERATE\_MIPMAP  Bandera opcional - especifica los mipmaps generados para todos los niveles. |   El siguiente bloque de referencia lista algunos constructores de la clase **TextureLoader**. Hay más constructores que no se listan en este bloque de referencia que permiten la carga de imágenes de textura desde otras fuentes. Puedes consultar la especificación del API Java 3D para ver una lista completa de todos los constructores.   |  | | --- | | Lista Parcial de Constructores de la Clase **TextureLoader**  Extiende: java.lang.Object  Paquete: com.sun.j3d.utils.image  Esta clase se usa para cargar una textura desde un objeto **Image** o **BufferedImage**. Se proporcionan métodos para recuperar el objeto **Texture** y el objeto **ImageComponent** asociado o una versión escalada del objeto **ImageComponent**.  El formato por defecto es RGBA.  Otros formatos legales son: RGBA, RGBA4, RGB5\_A1, RGB, RGB4, RGB5, R3\_G3\_B2, LUM8\_ALPHA8, LUM4\_ALPHA4, LUMINANCE y ALPHA  TextureLoader(java.lang.String fname, java.awt.Component observer)  TextureLoader(java.lang.String fname, int flags, java.awt.Component observer)  Contruye un objeto TextureLoader usando el fichero especificado, la opción flags y el formato por defecto RGBA.  TextureLoader(java.net.URL url, java.awt.Component observer)  TextureLoader(java.net.URL url, int flags, java.awt.Component observer)  Construye un objeto TextureLoader usando la URL especificada, opción flags y el formato por defecto RGBA. |  |  | | --- | | Sumario de Métodos de la Clase **TextureLoader**  ImageComponent2D getImage()  Devuelve el objeto ImageComponent2D asociado.  ImageComponent2D getScaledImage(float xScale, float yScale)  Devuelve el objeto ImageComponent2D escalado.  ImageComponent2D getScaledImage(int width, int height)  Devuelve el objeto ImageComponent2D escalado.  Texture getTexture()  Devuelve el objeto Texture asociado. |   **API de NewTextureLoader**  La razón de utilizar **NewTexureLoader** es evitar la necesidad de un observador de imagen para construir un cargador de textura. El siguiente boque de referencia enumera algunos constructores de la clase **NewTextureLoader**. **NewTextureLoader** tiene los mismos constructores que **TextureLoader** excepto en que ninguno requiere un componente del awt para servir como el observador de imagen.   |  | | --- | | Lista Parcial de Constructores de la Clase **NewTextureLoader**  Extiende: com.sun.j3d.utils.image.TextureLoader  Esta clase se usa para cargar una textura desde un fichero o una URL. Esta clase se diferencia de com.sun.j3d.util.image.TextureLoader sólo en la ausencia de un observador de imagen en el constructor y en el método para selecccionar un sólo observador de imagen para todos los usos posteriores.  NewTextureLoader(java.lang.String fname)  NewTextureLoader(java.lang.String fname, int flags)  Construye un objeto TextureLoader usando el fichero especificado, opción flags y el formato po defecto RGBA.  NewTextureLoader(java.net.URL url)  NewTextureLoader(java.net.URL url, int flags)  Construye un objeto TextureLoader usando la URL especificada, opción flags y el formato por defecto RGBA. |   El siguiente bloque de referencia lista los dos métodos definidos en la clase **NewTextureLoader**. Todos los demás métodos están definidos por la clase **TextureLoader**. Para usar un objeto **NewTextureLoader** se debe seleccionar primero un observador de imagen. Esto se hace normalmente cuando se crea el objeto **Canvas3d**.   |  | | --- | | Lista Parcial de Métodos de la Clase **NewTextureLoader**  java.awt.component getImageObserver()  Devuelve el objeto awt.component usado como el observador de imagen para los objetos NewTextureLoader.  void setImageObserver(java.awt.component imageObserver)  Selecciona un objeto awt.component como el objeto a usar como observador de imagen en la construcción de los siguientes objetos NewTextureLoader. | |  |  | |