UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño
y programación de videojuegos

UNLVIRTUAL

MANIPULACION DE OBJETOS EN 2D

Unidad 5: Modelado de la terminación superficial

Docente
Walter Sotil

Tutores
Emmanuel Rojas Fredini, Cristian Yones

CONTENIDOS

1. Introducción 3

2. Texturas 3

2.1. Textura 1D 4

2.2. Textura 2D 4

3. Carga de la textura 5

4. Repetición de la textura 6

4.1. Clamping 6

4.2. Repeating 6

5. Modos de mezcla 7

5.1. Replace 7

5.2. Decal 7

5.3. Modulate 8

5.4. Blend 8

6. Filtrado 9

6.2. Filtros de magnificación 9

6.3. Filtros de minimización 10

7. Mipmapping 10

8. Environment mapping 12

9. Efectos especiales: líneas de perfil 12

10. Efectos especiales: bump mapping 13

BIBLIOGRAFÍA 14

1. Introducción

Hasta el momento hemos dibujado figuras compuestas por polígonos, los cuales se veían en modo wireframe (o render de alambre) o bien estaban pintados en su interior, formando un color relativamente uniforme y que en general los hacía carentes de realismo. De esta manera, por ejemplo, dibujar superficies sencillas como una pared requeriría gran cantidad de polígonos.

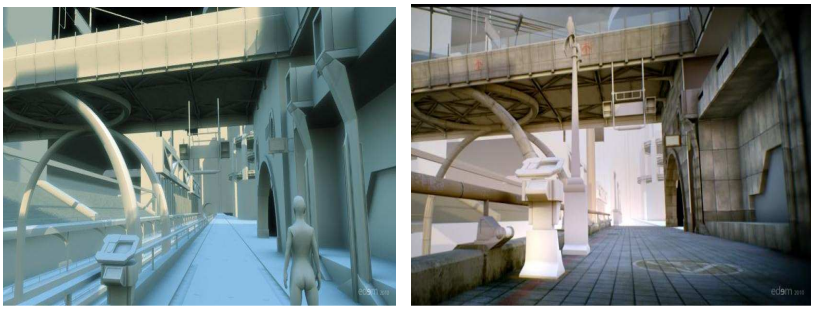


Figura 1. Izquierda: escena sin texturas. Derecha: escena con texturas.

Las texturas solucionan estos problemas, permitiendo simular superficies complejas con baja cantidad de polígonos, suministrando realismo a la escena e incrementando drásticamente el nivel de detalle. En esencia, se busca que cada punto de una superficie se corresponda con un punto de una imagen.

Hay que tener en cuenta que el mapeado de texturas es una ilusión visual, es decir, no se crea geometría nueva, y puede lograr que los objetos sean mucho más realistas.

La principal ventaja del mapeado de texturas es que permite aumentar enormemente la complejidad visual de un modelo sin necesidad de aumentar su complejidad geométrica. En cualquier escena que se aprecie, la iluminación y las texturas suelen tener mayor importancia que la propia geometría que sustenta los modelos. Convertir la visualización de un simple polígono rectangular en un muro de ladrillo, o una sencilla esfera en un planeta, abre nuevas vías de explotación para una multitud de campos, entre ellos, el desarrollo de videojuegos.

Los principales factores que debemos considerar a la hora de implementar el mapeado de texturas son la velocidad y la precisión en la aplicación de los mapas. En ciertos casos, prima un factor sobre el otro. En los videojuegos es mucho más importante la velocidad de representación, para permitir una gran fluidez de juego, que la precisión en los cálculos. De hecho, en este tipo de aplicaciones, es habitual percibir alteraciones en las texturas o incluso en los polígonos.

2. Texturas

En un sentido amplio, se puede definir a la *textura* como un *array* de uno, dos o tres dimensiones de téxels. A cada píxel de la textura se lo denomina *téxel*, dado que la palabra *píxel* se reserva para los fragmentos rasterizados.

Textura
En un sentido amplio, se la puede definir como un array de uno, dos o tres dimensiones de téxels. A cada píxel de la textura se lo denomina téxel, dado que la palabra píxel se reserva para los fragmentos rasterizados.

Según la dimensión del espacio de textura, podemos distinguir tres tipos: 1D, 2D, 3D. Aunque en esta materia sólo veremos texturas unidimensionales y bidimensionales, en la especificación 1.2 de OpenGL ya se permite trabajar con texturas en 3D.

Pegar una textura sobre un objeto consiste en definir una función unívoca (función de mapeo o mapping) que a cada punto de una superficie le hace corresponder un punto de una imagen.

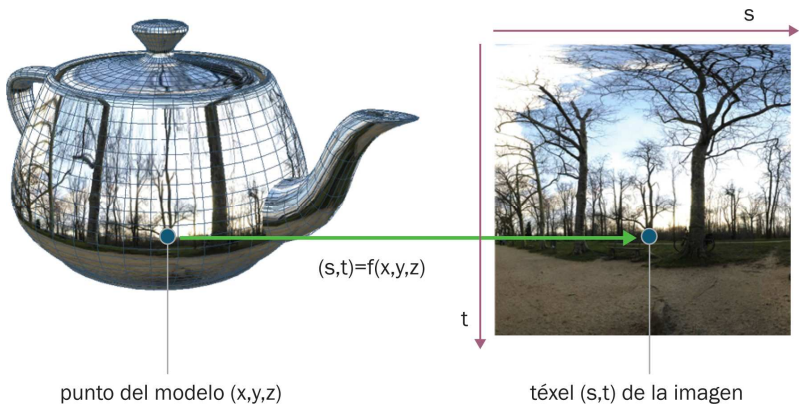


Figura 2. Función de mapeo o mapping.

Es importante mencionar que OpenGL lee e interpreta las texturas con el centro de las coordenadas en el vértice superior izquierdo, es decir con la imagen espejada con respecto al eje *x*.

Luego profundizaremos en esto, pero es necesario saber que las coordenadas de textura se asignan a la primitiva de manera manual o en forma automática a través de OpenGL.

2.1. Textura 1D

Es una imagen que tiene un único téxel de ancho o de alto.

Textura 1D
Es una imagen que tiene un único téxel de ancho o de alto.

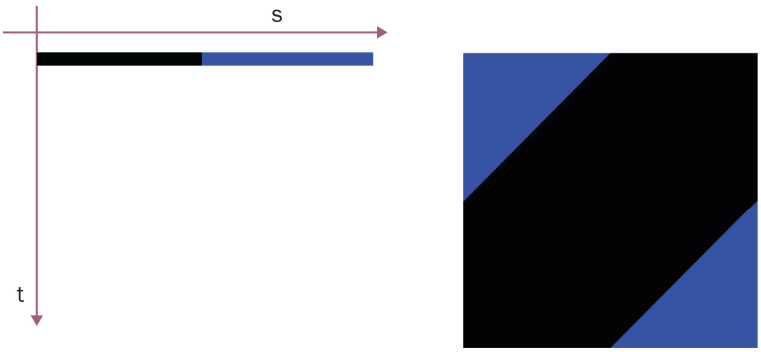


Figura 3. Aplicación de textura 1D.

Tiene la ventaja de poseer rápida generación con la consecuente mejora en la eficiencia.

La función de mapeo es la siguiente: s = f(x,y,z).

2.2. Textura 2D

La textura 2D es una imagen formada por un *array* de téxels de dos dimensiones. Se puede pensar como una foto plana que se pega sobre una superficie en 3D. Es el tipo de textura más utilizado.

Textura 2D
Es una imagen formada por un array de téxels de dos dimensiones.

Al igual que la textura en 1D, se compone de valores de color RGB y puede incluir valores alpha (canal de transparencia).

La función de mapeo es la siguiente: (s,t)=f(x,y,z).

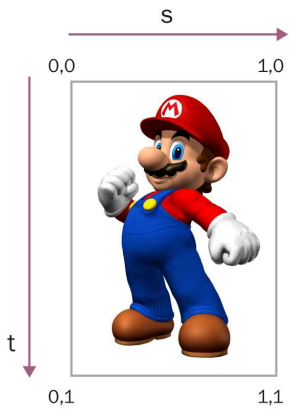


Figura 4. Textura 2D.

Un detalle significativo para mencionar es que las coordenadas del espacio de textura están normalizadas, es decir, varían entre 0 y 1, independientemente del tamaño. Para ello, se realiza una asignación de la textura punto a punto. A cada vértice de cada primitiva a la que se le aplica la textura, se le asignan coordenadas {s,t} entre 0 y 1, definiendo un área de correspondencia entre el modelo y la textura que se mapea, finalmente, por interpolación. Estas áreas no son necesariamente iguales, sino que la textura se expande o contrae para cubrir la primitiva. En el interior de la misma, OpenGL realiza una interpolación de las coordenadas de la textura en forma automática.

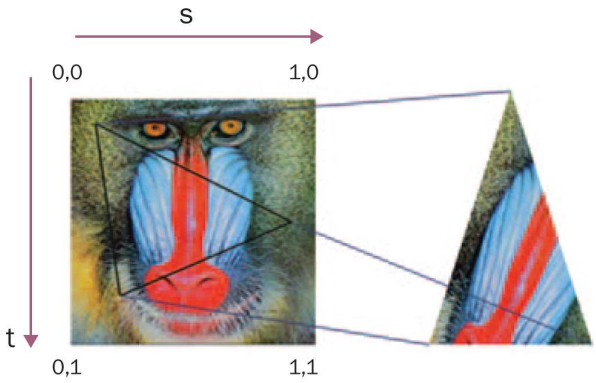


Figura 5. OpenGL realiza la interpolación de las coordenadas de la textura.

3. Carga de la textura

Texturas procedurales
Se crean mediante distintos algoritmos para ser luego asignadas a las primitivas.

La textura se puede leer desde un archivo o generar. En este último caso, estamos hablando de las *texturas procedurales,* las cuales se crean mediante distintos algoritmos para ser luego asignadas a las primitivas. Esto es fácil si la textura es simple, como un tablero de ajedrez.

Los pasos a seguir para la realizar la carga de la textura son los siguientes:

- Habilitar el mapeado de texturas, en este ejemplo 2D, a través de la función glEnable(GL_TEXTURE_2D).

- Alocar memoria con la función glGenTextures.

- Seleccionar una textura a aplicar con glBindTexture.

- Definir distintos parámetros de configuración tales como tipo de filtro a aplicar y modo de mezcla con el color de la primitiva.

- Con glTexImage1D o glTexImage2D, definir el significado de los datos (RGB, RGBA, luminancia, un solo color, sólo alpha, etc), y el puntero al array, donde se encuentra guardada la textura en memoria.

Hay varios formatos para la interpretación de datos, tales como GL_RGB, GL_ALPHA, entre otros, los cuales se eligen de acuerdo al efecto final deseado.

La asignación de las coordenadas de textura a las primitivas se realiza de forma automática a través de OpenGL o de manera manual.

Para la manera manual se deberá asignar a cada uno de los vértices de la primitiva las coordenadas de textura. Estas coordenadas determinan qué téxel en el mapa de texturas es asignado a cada vértice de la primitiva. Para ello es necesario llamar antes de cada glVertex a la función: glTexCoord.

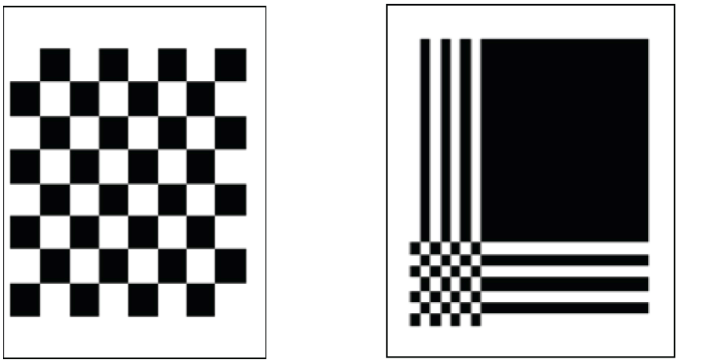
4. Repetición de la textura

En este apartado describiremos qué ocurre cuando se referencian coordenadas de texturas con un valor mayor que 1 o menor que 0. Hay dos alternativas para la apliacación de la textura, denominadas *Clamping* y *Repeating*.

4.1. Clamping

Clamping
Permite repetir los téxels de los bordes de la textura cuando se referencia fuera de ella.

Permite repetir los téxels de los bordes de la textura cuando se referencia fuera de ella. No tiene mucha utilidad, salvo cuando estamos seguros de que no nos vamos a exceder de rango [0,1] en el espacio de textura.



Aplicación de textura original

Aplicación de textura con *clamping*

Figura 6. Utilización de clamping.

En OpenGL se utiliza la función:

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)

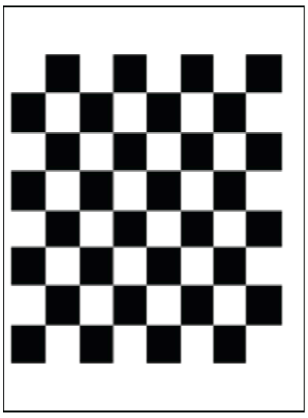
4.2. Repeating

Repeating
Consiste en repetir la textura completa en la primitiva, provocando un efecto de *embaldosado*.

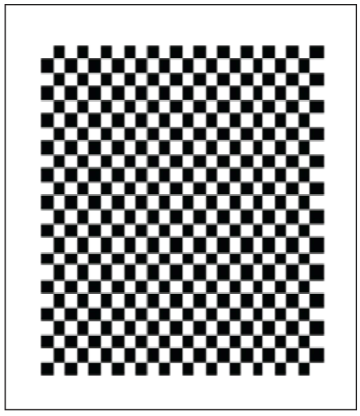
Consiste en repetir la textura completa en la primitiva, provocando un efecto de *embaldosado.* La imagen de la textura está repetida varias veces. Permite trabajar con texturas más chicas y ahorrar memoria.

Asigna las coordenadas de forma que:

s = s-int(s); t = t-int(t)



Aplicación de textura original



Aplicación de textura con *wrapping*

Figura 7. Utilización de repeating.

En OpenGL se usa la función:

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)

GL_TEXTURE_WRAP_S indica repetición de las coordenadas *X* de la textura. GL_TEXTURE_WRAP_T indica repetición de las coordenadas *Y* de la textura.

5. Modos de mezcla

Modos de mezcla
OpenGL brinda cuatro maneras de determinar el valor RGBA final de cada píxel a visualizar, a partir del color de la superficie del objeto a texturizar (fragmento) y del color de la textura (téxel): Replace (sustitución), Decal (sustitución con transparencia), Modulate (modulación) y Blend (mezcla utilizando un tercer color).

Cuando se aplica una textura es necesario especificar cómo queremos que se mezcle con el color del objeto.

OpenGL permite elegir entre distintos métodos con el objetivo de determinar el valor RGBA final de cada píxel a visualizar, a partir del color de la superficie del objeto a texturizar (fragmento) y del color de la textura (téxel). Los cuatro modos de mezcla posibles de elegir son *Replace* (sustitución), *Decal* (sustitución con transparencia), *Modulate* (modulación) y *Blend* (mezcla utilizando un tercer color).

La función OpenGL a emplear es:

glTexEnv(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,(*método*))

Donde *método* indica uno de los recién mencionados.

5.1. Replace

En este modo, el color utilizado para el objeto es el de la textura, es decir, sustituye el color de la superficie por el de la textura.

5.2. Decal

En este modo de mezcla se diferencian dos casos. Si trabajamos con formato de textura RGB, el color utilizado para el objeto es el de la textura.

Base Internal Format	Decal Texture Function
GL_RGB	C = Ct, A = Af
GL_RGBA	C = Cf(1-At) + CtAt, A = Af

Figura 8. Color final en función de los formatos de textura RGB y RGBA.

En cambio, con formato de textura RGBA, el color a utilizar es una mezcla del color de la superficie (Cf) y el de la textura (Ct), donde el predominio de uno sobre otro depende del valor alpha de la textura (At). La fórmula es una interpolación lineal donde el parámetro de interpolación es el valor de alpha, que se encuentra entre 0 y 1.

La aplicación del formato de textura RGBA puede pensarse como la acción de pegar una calcomanía con partes opacas y otras con transparencia.

Es importante destacar que en el *Red Book* se pueden encontrar descriptos más formatos de textura.

5.3. Modulate

Permite ir escalando el color final entre el original de la superficie y el negro, multiplicando a cada color del objeto por el de la textura.

Base Internal Format	Modulate Texture Function
GL_RGB	C = CfCt, A = Af
GL_RGBA	C = CfCt, A = AfAt

Figura 9. Color final en función de los formatos de textura RGB y RGBA.

También puede modular la transparencia alpha.

Multiplicar los colores sirve para realizar ciertos efectos especiales, como *matar* o *dejar vivo* sólo un canal de color.

5.4. Blend

Con esta opción, el color utilizado para el objeto es una mezcla del color de la superficie (Cf) y otro que define el usuario (Cc), donde el predominio de uno sobre otro depende del color de la textura (Ct), que aquí opera como parámetro de interpolación.

Base Internal Format	Blend Texture Function
GL_RGB	C = Cf(1-Ct) + CcCt, A = Af
GL_RGBA	C = Cf(1-Ct) + CcCt, A = AfAt

Figura 10. Color final en función de los formatos de textura RGB y RGBA.

Cc posee componentes R, G, B, A, y se define con:

glTexEnv(GL_TEXTURE_ENV,GL_TEXTURE_ENV_COLOR,(*puntero*))

mediante un puntero al color que define el usuario: {Rc,Gc,Bc,Ac}.

6. Filtrado

Cuando el tamaño de los píxeles no coincide con el de los téxels aparecen defectos visuales que disminuyen la calidad del resultado.

Uno de estos problemas es el de la *magnificación.* Surge cuando el objeto se acerca demasiado a la cámara, debido a que los téxels proyectados sobre el objeto aparecen mayores que los píxeles de la imagen, y la textura se ve pixelada. Es decir, cuando un téxel resulta más grande que un píxel.

Para poder suavizar estos defectos y adaptar las texturas, se aplica una serie de operaciones o filtros.

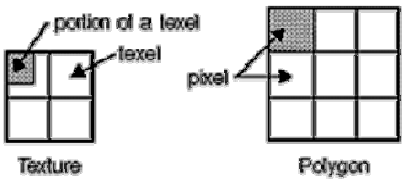


Figura 11. Magnificación.

Otro efecto desagradable también aparece cuando los téxels proyectados son más pequeños que los píxeles, por lo que la textura parece cortada. Este problema se denomina *minimización* y se da cuando el objeto se aleja demasiado del observador.

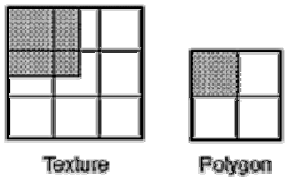


Figura 12. Minimización.

OpenGL tiene implementados distintos filtros, que se activan mediante glTexParameter, con la finalidad de mitigar las falencias producidas por la magnificación y la minimización.

6.1. Filtros de magnificación

Filtros de magnificación
Indican cómo calcular el color de un píxel de la imagen cuando los téxels ocupan varios píxeles.

Indican cómo calcular el color de un píxel de la imagen cuando los téxels ocupan varios píxeles.

OpenGL define dos filtros de magnificación:

- *Filtro puntual (GL_NEAREST):* se elige el punto de la textura más cercano a las coordenadas del píxel. Es el filtro por defecto y el menos costoso, pero produce el efecto de dentado.

- *Filtro lineal (GL_LINEAR):* se realiza una interpolación bilineal (promedio ponderado otorgándole mayor peso a los téxels más cercanos al píxel) entre los colores de una matriz de 2x2 téxels (los que están más cerca del centro del píxel evaluado), de forma que aparece la imagen suavizada, produciendo un resultado borroso, pero que elimina el dentado.

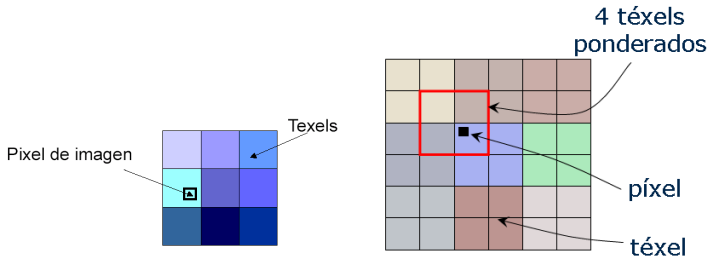


Figura 13. Izquierda: filtro puntual. Derecha: filtro lineal.

6.2. Filtros de minimización

Filtros de minimización
Indican cómo calcular el color de un píxel de la imagen cuando los téxels son más pequeños que los píxeles.

Indican cómo calcular el color de un píxel de la imagen cuando los téxels son más pequeños que los píxeles.

OpenGL especifica dos filtros:

- *Filtro puntual (GL_NEAREST):* se elige el téxel situado en el centro del píxel. El problema es que si la posición cambia ligeramente, se producen los efectos de dentado y fluctuación de la textura.

- *Filtro lineal (GL_LINEAR):* como ocurre en el filtro de magnificación lineal, se realiza una interpolación bilineal (o promedio ponderado) entre los colores de una matriz de 2x2 téxels cercanos al centro del píxel evaluado. Elimina un poco el dentado.

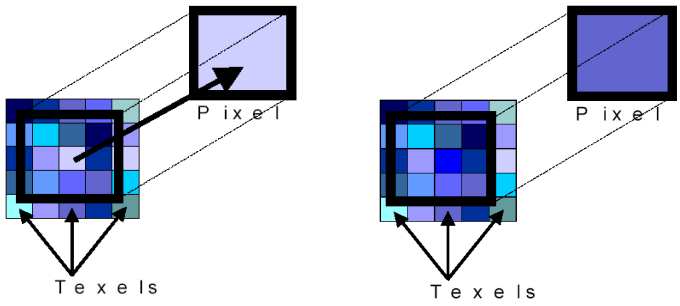


Figura 14. Izquierda: filtro puntual. Derecha: filtro lineal.

7. Mipmapping

Mipmapping
Es una técnica que permite guardar distintas versiones de la misma textura, con diferentes resoluciones.

Los objetos sobre los que se han aplicado texturas pueden ser vistos, al igual que cualquier otro objeto, a distintas distancias según la posición del observador.

En una escena dinámica, por ejemplo, cuando un objeto se aleja de la posición del observador, la textura debería disminuir en tamaño en relación con la imagen proyectada.

Un procedimiento utilizado para causar este efecto consiste en obtener un conjunto de texturas de resoluciones decrecientes, denominadas *mipmaps*. Con la técnica mipmapping, se guardan distintas versiones de la misma textura, con diferentes

resoluciones. Por lo tanto, al utilizar versiones de menor resolución de la textura a medida que nos alejamos del objeto, se mitigan los problemas de minimización.

El procedimiento tiene un consumo mayor de memoria (aunque menor al 1.5 de la textura original).

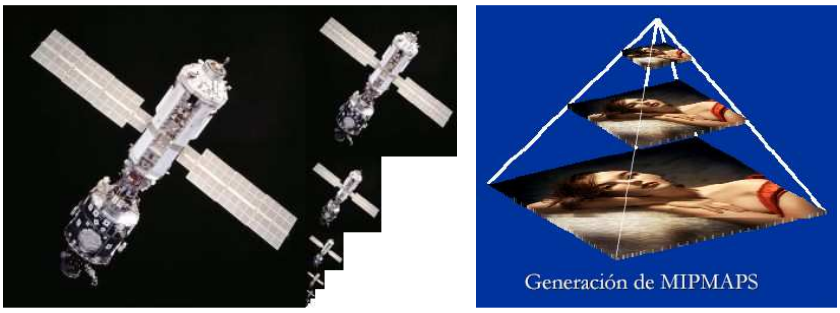


Figura 15. Izquierda: una imagen a distintas resoluciones. Derecha: resoluciones en función a la distancia con la cámara.

Para utilizar mipmapping con OpenGL se deben proporcionar texturas de todos los tamaños en potencias de 2, de modo que los tamaños estén comprendidos entre la máxima resolución y la resolución1x1, mediante la función glTexImage2D.

Para facilitar el uso de mipmaps, GLUT proporciona dos funciones denominadas gluBuild1DMipmaps y gluBuild2Dmipmaps, las cuales, dada una única textura, generan automáticamente texturas para resoluciones menores. Durante la visualización, se elige dinámicamente cuál es el nivel del mipmap que hay que aplicar en cada caso (aquel en el que el tamaño del téxel proyectado sobre el objeto sea aproximadamente el de un píxel).

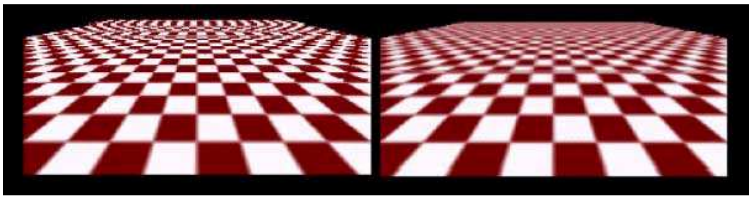


Figura 15. Izquierda: mipmaps desactivados. Derecha: mipmaps activados.

Con la incorporación de los mipmaps se usa la función glTexParameter para establecer más opciones en los filtros de minimización, según se detalla en la siguiente tabla:

Filtro	Descripción
GL_NEAREST_MIPMAP_NEAREST	Utiliza la imagen más cercana a la resolución de la pantalla. Usaremos el filtro GL_NEAREST cuando usemos este mapa
GL_LINEAR_MIPMAP_NEAREST	Utiliza la imagen más cercana a la resolución de la pantalla. Usaremos el filtro GL_LINEAR cuando usemos este mapa
GL_NEAREST_MIPMAP_LINEAR	Interpola linealmente entre las dos imágenes más cercanas a la resolución de la pantalla. Usaremos el filtro GL_NEAREST cuando utilicemos este mapa.
GL_LINEAR_MIPMAP_LINEAR	Interpola linealmente entre las dos imágenes más cercanas a la resolución de la pantalla. Usaremos el filtro GL_LINEAR cuando utilicemos este mapa.

El último filtro es el más costoso en tiempo de ejecución.

8. Environment mapping

Environment mapping
Es un método para añadir realismo a una escena, en la que un objeto parece un espejo de su entorno.

Es un método para añadir realismo a una escena, en la que un objeto parece un espejo de su entorno. Por ejemplo, si observamos un objeto de plata en una habitación, veremos las paredes, el piso y otras cosas del lugar reflejados en ese objeto.

Para desarrollar el environment mapping, debemos crear una textura y OpenGL generará el entorno automáticamente. Es decir que aquí las coordenadas de textura se asignan de manera automática a través de OpenGL.

El mapeo de entorno esférico es el método más popular de environment mapping y el mismo es soportado por OpenGL. Las coordenadas de la textura se calculan por reflexión en las caras de la primitiva, según un mapeo esférico.

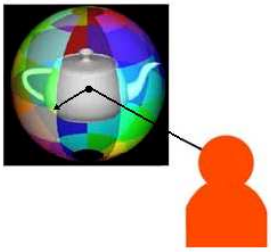


Figura 16. Mapeo de entorno esférico.

Para implementar el mapeo de entorno esférico con OpenGL hay que crear un mapa de la textura apropiado (mapeo de textura esférico) y lograr que OpenGL genere las coordenadas de la textura de forma automática.

Se utiliza la función:

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP)

glEnable(GL_TEXTURE_GEN_S)

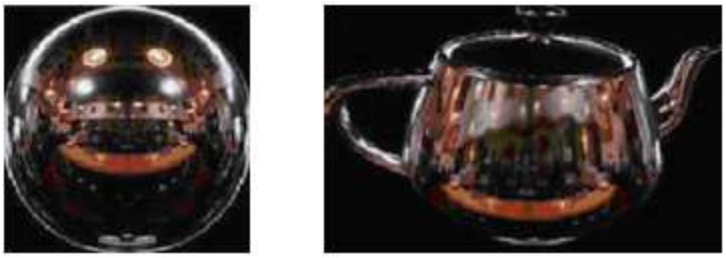


Figura 17. Izquierda: textura. Derecha: mapeo de entorno esférico.

9. Efectos especiales: líneas de perfil

Líneas de perfil
Constituyen un método mediante el cual se corta al objeto con un plano y se distribuye la textura en función de la distancia al plano.

Por medio de este método se corta al objeto con un plano y se distribuye la textura en función de la distancia al plano. Nuevamente, aquí las coordenadas de textura se asignan de manera automática a través de OpenGL.

La normal al plano indica la orientación de la textura.

OpenGL genera las coordenadas de textura automáticamente con glTexGen.

El plano queda definido por un punto $\mathbf{P_0}(x_0,y_0,z_0)$ y un vector $\mathbf{n}(a,b,c)$ ortogonal al plano.

La ecuación vectorial del plano está dada por $(\mathbf{P} - \mathbf{P_0}) \cdot \mathbf{n} = 0$.

En tanto que $d = ax_0 + by_0 + cz_0 = \mathbf{n} \cdot \mathbf{P_0}$.

Se deben especificar a OpenGL las tres componentes del vector ***n*** ortogonal al plano y la "distancia al plano" *d.*

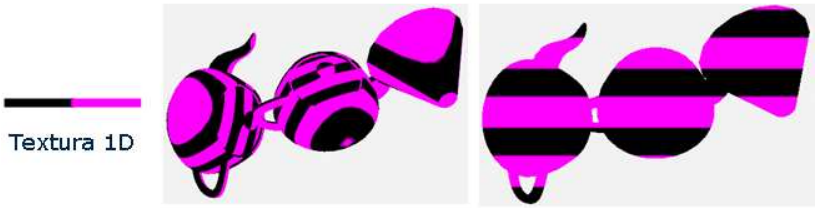


Figura 18. Izquierda: textura. Derecha: mapeo con repetición de textura activa.

10. Efectos especiales: bump mapping

Bump mapping
Es una técnica según la cual la geometría del objeto no es afectada, sino que sólo se modifican sus vectores normales, alterando su magnitud o dirección para simular depresiones y picos en su superficie.

Aunque aquí no veremos la implementación de esta técnica, su importancia nos obliga a mencionar algunas de sus características principales.

Con esta técnica, la geometría del objeto no es afectada, sino que sólo se modifican sus vectores normales, alterando su magnitud o dirección para simular depresiones y picos en su superficie.

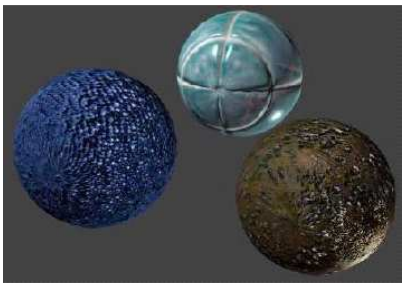


Figura 18. Aplicación de bump mapping.

Utiliza un espacio de textura 2D, que codifica vectores normales en lugar de color.

Al mapear esta textura sobre una superficie, el vector normal de esta última se modifica según el valor encontrado en la textura.

Existen distintas maneras de implementar bump mapping; a continuación mostramos una:

Figura 19. Aplicación de bump mapping a la tetera.

En este ejemplo, podemos observar cómo se mapea en relieve la letra *D* sobre la superficie de la tetera. Cuando en la textura se encuentra la posición donde hay una transición de blanco al marrón (o del marrón al blanco), se debe rotar un determinado ángulo el vector normal a la superficie en ese punto de transición.

BIBLIOGRAFÍA

Cátedra Computación Gráfica, FICH - UNL-, 2010 [en línea] [CIMEC] www.cimec.org.ar/cg

Green D. Drexel University. , 2011 [en línea] [Drexel] http://www.cs.drexel.edu/~david/Classes/ICG/Lectures/Lecture7.6.pdf

Kilgard M. J. "The OpenGL Utiliy Toolkit (GLUT) Programming Interface". Silicon Graphics, Inc, 1996 [en línea] [OpenGL] http://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf

Neider J.;Davis,T.; Woo M. "OpenGL Programming Guide: The Official Guide to Learning OpenGL". Addison-Wesley Publishing Company, 1997 [en línea] [OpenGL] http://www.opengl.org/documentation/red_book/

OpenGL [en línea] http://www.opengl.org

The Blue Book: The OpenGL Reference manual [en línea] [OpenGL] http://www.opengl.org/documentation/blue_book/

Wolfe R. "Teaching Texture Mapping Visually". Siggraph, 1999 [en línea] [OpenGL] http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm