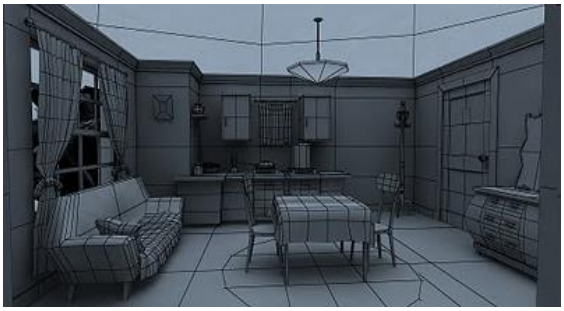
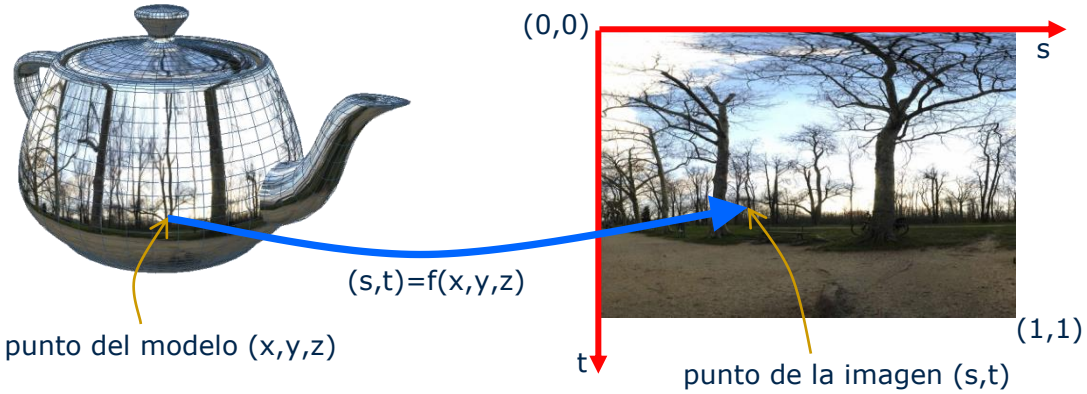
Texturas

La palabra “textura” tiene la misma raíz que “textil” y se refiere a la trama de un tejido o tela, ya sea en lo táctil como en lo visual. En CG una textura es una imagen, la imagen de una textura compleja que se aplica sobre una superficie simple.



Una escena en CG se compone de objetos formados por aproximación, mediante polígonos simples. Si hubiese que representar una superficie de césped, modelando hoja por hoja, la escena sería inmanejable; en su lugar, se puede recurrir a un único rectángulo, con una imagen “pegada” que aparente ser césped. Lo mismo sucede para una superficie lisa de colores variables, como un mármol o una madera; para definir distintas regiones con distintos materiales habría que subdividir en primitivas que cubran cada región por separado; allí también el problema se resuelve aplicando una imagen sobre una superficie sencilla. En síntesis, las texturas (CG) son imágenes que permiten representar objetos de textura (castellano) compleja pero sobre una geometría sencilla.



Una imagen o foto digital es un *array* bidimensional de WxH pixeles de ancho y alto, que se almacena comenzando desde el borde superior izquierdo y continúa por filas. Los píxeles de la imagen de textura suelen denominarse téxeles. Primero se hace un mapeo lineal: a la imagen se le asignan coordenadas (s,t) reales entre 0 y 1; de modo que (s,t) = (0,0) corresponda al borde superior izquierdo del primer téxel y (1,1) al inferior derecho del último. Un dado par $(s,t) \in [0,1]^2 \in \mathbb{R}^2$ cae en algún punto de la imagen, dentro de un cuadradito de 1x1 que representa el espacio ocupado un téxel, con un dado color.

Los pasos básicos consisten en: 1) Leer la imagen. 2) Definir cómo se pega o mapea (*texture mapping*) la textura sobre cada primitiva; es decir: una función $(s,t) = f(x,y,z)$ que asigne coordenadas de textura a cualquier punto de la superficie del modelo. Hay varios métodos; el básico consiste en asignar coordenadas (s,t) a los vértices de las primitivas y que se interpolen al rasterizar (cada fragmento recibe color, normal y coordenadas de textura interpolados, en forma hiperbólica, de los asignados a los vértices de la primitiva). 3) Las coordenadas (s,t) asignadas o las calculadas, son un par de reales cualesquiera, pero siempre habrá que definir un mecanismo para que terminen entre 0 y 1. 4) En ese punto de la imagen, con $(s,t) \in [0,1]^2$, se lee el color o se interpola de los téxeles más cercanos. 5) Finalmente, ese “color de textura”, se asigna al fragmento o se mezcla con el color del sombreado, que traía el fragmento.

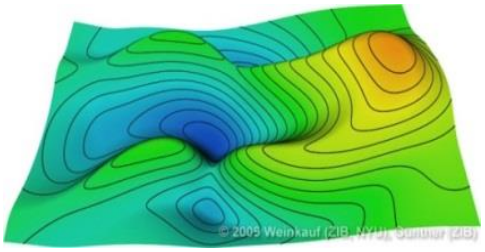
Cada téxel contiene un vector de hasta cuatro componentes, que normalmente se asumen como color y alfa (R,G,B,A); pero también pueden interpretarse como al programador le convenga. Esto se aprovecha en los *shaders* programables, para trasmitir cualquier variable vectorial 4D a cada fragmento, puede ser para modificar la normal, el *z*, el material o para decidir el descarte del fragmento.

Dimensiones del *buffer* de textura

El uso estándar involucra imágenes bidimensionales, pero se pueden utilizar texturas de entre uno y tres dimensiones, aunque con cuatro coordenadas, que se denominan (s,t,r,q) en OpenGL y (s,t,p,q) en GLSL (shaders), donde la r se reserva para rojo. La cuarta, q, se utiliza para indexar o para proyectar texturas.

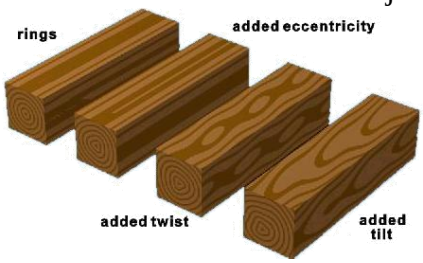
Una textura unidimensional es una línea de color y/o alfa (imagen de Wx1) que se utiliza en mapas de color para visualización científica, en mapas de isolíneas o para hacer invisibles partes del objeto.

Supongamos una superficie con una temperatura (o cualquier variable real) en cada vértice. Una textura 1D como la de arriba; haciendo $s = (t-t_{min})/(t_{max}-t_{min})$, es decir: identificando el azul con la mínima temperatura y el rojo con la máxima; permite visualizar las temperaturas. Con otra textura de un solo téxel con alfa 1 (0,0,0,1) y el resto con alfa 0 (0,0,0,0); asignando s = t/10-int(t/10) se logra marcar una isoterma que se repite cada 10º. En la figura se han aplicado dos texturas unidimensionales para mostrar la altura z, en lugar de t.



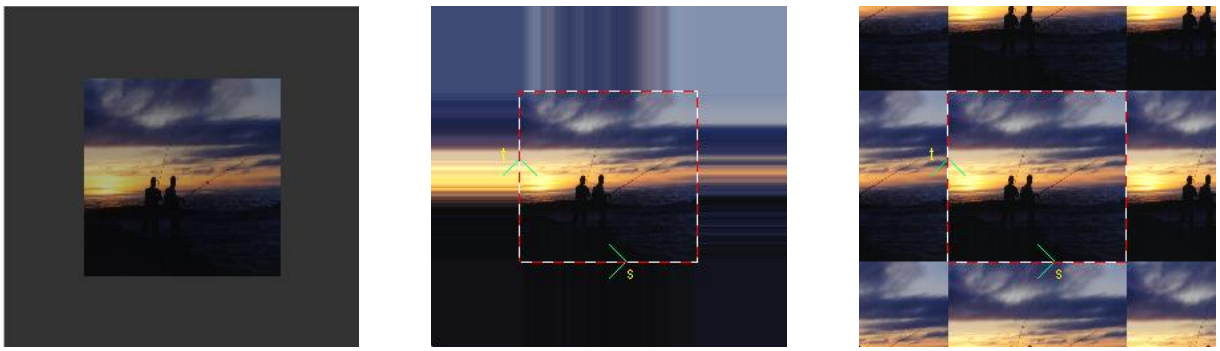
En otro uso, con algunos téxeles de alfa nulo y usando el test de alfa, se definen zonas invisibles.

Las texturas tridimensionales o volumétricas se utilizan casi siempre sin deformar, mediante un mapeo afín de cada coordenada de objeto a textura: $s = (x-x_{min})/(x_{max}-x_{min})$ y lo mismo para el resto. La imagen



3D, es decir el *buffer* de datos RGBA en cada "vóxel" (píxel 3D), se puede generar calculando el color de cada vóxel en un procedimiento (**textura procedural**) que genera imágenes como el veteado de la madera o el mármol. También puede provenir de algún equipo de tomografía o resonancia magnética y se utiliza para visualizar órganos, o provenir de mediciones, para visualizar estratos rocosos, composición del suelo, etc.

Tratamiento de las coordenadas fuera del rango (*wrapping*)



Las coordenadas en la imagen (s,t,r,q) varían entre cero y uno, pero se definen o calculan con cualquier otro valor e internamente se reasignan al intervalo [0,1] por medio de uno de varios métodos alternativos de *wrapping* (envoltorio), los dos clásicos son: *clamp* y *repeat*, el método se define para cada coordenada de textura y puede ser distinto para cada una. Se definen del siguiente modo:

- ***Clamping***: s = max(0,min(s,1)). Lo mismo para las otras coordenadas. Esto es: cualquier valor menor que cero se toma como cero y cualquier valor mayor que uno se toma como uno. Solo tiene sentido usar *clamping* si se garantiza que la textura cubre bien al objeto; en caso contrario, la primera o la última línea de la imagen se verá repetida del lado que exceda los límites. Cuando se quiere mapear una imagen simple, una sola vez en un objeto grande (ej.: brillo reluciente en una copa o calcomanía en un tanque de moto) también se usa *clamping*, pero el borde de la imagen debe ser invisible (normalmente, con alfa nulo) y de ese modo la repetición del borde no influye sobre el resto del objeto.

- ***Repeat***: s = s−int(s). Se conserva solo la parte fraccionaria del valor. La imagen se repite en la dirección en que se aplica ese método. Se utiliza, obviamente, para texturas repetitivas como césped, ladrillos o diseños de telas y en las líneas de nivel que se mostraban arriba en 1D. Existe un modo alternativo llamado ***mirrored repeat*** que actúa como reflejando la imagen en los bordes.

Métodos de mapeo

Mapear una textura en un objeto consiste en asignar coordenadas de textura a los puntos del objeto. La asignación se realiza mediante un algoritmo, que debe responder cuales son las coordenadas de textura para cualquier punto, normalmente: el centro de un fragmento. El resultado es distinto si las coordenadas espaciales se dan en el espacio del modelo (textura fija al objeto) o el del ojo (textura deslizante). La función debe definirse de forma analítica o procedural, pero con unos pocos parámetros adecuados.

Normalmente, las coordenadas de textura se definen, manual o automáticamente, pero sólo en los vértices de las primitivas y luego es el software en el hardware gráfico, al rasterizar la primitiva, quien interpola las coordenadas de los vértices y calcula las coordenadas que corresponden a cada fragmento interior. En casos especiales o para realizar trucos avanzados, la asignación puede hacerse en un programa especial (*fragment shader*) que se interpone en el *pipeline* gráfico y se ejecuta en la GPU.

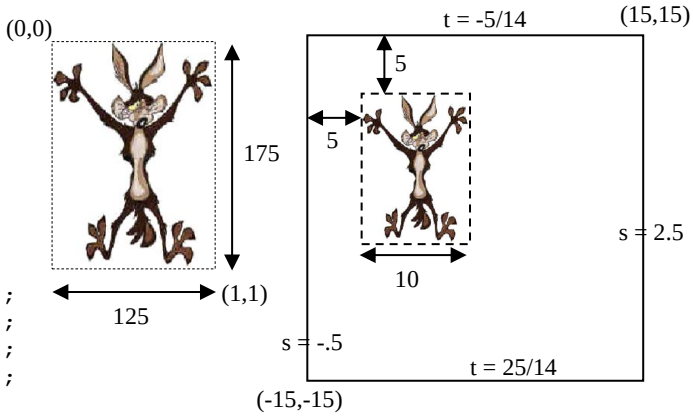
Mapeo Manual:

El método más primitivo consiste en la asignación manual de coordenadas de textura a cada vértice de cada primitiva. Este método se utiliza solo en los casos más sencillos, como un piso de césped, una pared de ladrillos, una etiqueta o una calcomanía, sobre una superficie analítica o simple.

Cuando se mapea una textura estilo calcomanía, que debe quedar en un lugar determinado del interior del objeto, hay que asignar coordenadas de textura a los vértices de modo que el rango [0,1] quede donde se pretende. Veamos el caso más sencillo: para pegar la imagen del coyote (125x175 píxeles) en un cuadrado de 30x30 unidades del modelo, con ancho 10 y posición (5,5) como se indica:

$\Delta x = 10 \Rightarrow \Delta s = 1$
$\Delta x = -5 \Rightarrow \Delta s = -5/10 = -.5$
$\Delta x = 25 \Rightarrow \Delta s = 25/10 = 2.5$
$\Delta y = 10*175/125 = 14 \Rightarrow \Delta t = 1$
$\Delta y = -5 \Rightarrow \Delta t = -5/14$
$\Delta y = 25 \Rightarrow \Delta t = 25/14$

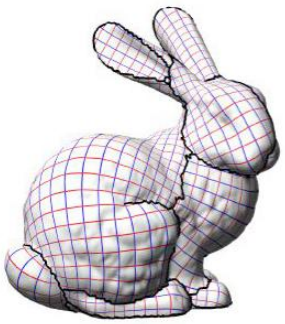
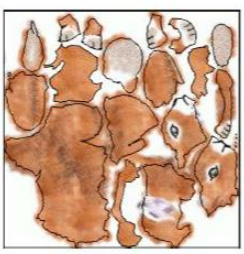
```
float s0=-.5,s1=2.5,
      t0=-5.f/14,t1=25.f/14;
glBegin(GL_QUADS);
  glTexCoord2f(s0,t1); glVertex2i(-15,-15);
  glTexCoord2f(s1,t1); glVertex2i( 15,-15);
  glTexCoord2f(s1,t0); glVertex2i( 15, 15);
  glTexCoord2f(s0,t0); glVertex2i(-15, 15);
glEnd();
```



Las inmersiones 1D y 3D que se mencionaron antes como ejemplos, podrían considerarse de este tipo.

También se considera “manual” al mapeo analítico en un algoritmo, que asigna coordenadas a los vértices de cada cuadradito elemental de una superficie analítica simple como un cilindro o una esfera; las coordenadas de textura se ponen en función de las coordenadas cilíndricas o esféricas de los vértices.

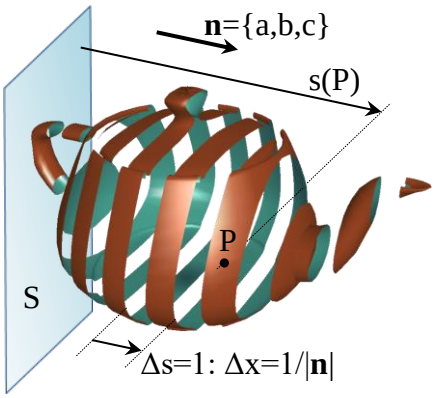
Un método emparentado con el anterior, se conoce como *UV mapping*. Consiste en mapear la superficie del objeto sobre una o varias superficies planas de coordenadas (u,v) (esto es “parametrizar” la superficie) y luego identificar algunos puntos del mapa con algunos puntos de la imagen. Normalmente se debe disponer de una geometría de base sencilla (NURBS o *subdivision surfaces*) o de algoritmos que permitan estirar y deformar (*unwrap*) la “piel” del objeto (una triangulación) sobre un plano, con pocos cortes y sin solapamientos, para realizar luego unas pocas asignaciones manuales punto a punto y el resto por software. Para evitar los solapamientos se recortan las partes complicadas y se ubican sueltas en un mapa general que se conoce como “atlas de textura”.


Mapeo Plano:

Es el método más sencillo de mapeo automático, que viene programado en OpenGL. Consiste en asignar un plano a cada coordenada de textura, que se define en función de la distancia del punto al plano.

La idea parte de las simples coordenadas cartesianas: Una forma trivial de asignar las coordenadas (s,t) de una imagen 2D, es usar $s=x$, $t=y$, independientemente de z. Esto es: a un punto de coordenadas (x,y,z) se asignan automáticamente las coordenadas de textura (s,t) = (x,y); que se clampean o repiten cuando los valores salen del rango [0,1]. La coordenada x se puede ver como la distancia a un plano S vertical por el origen (plano yz); la coordenada y es la distancia a un plano T horizontal (xz); asignamos coordenadas de textura a un punto P, haciendo: s = dist(S,P); t = dist(T,P).



OpenGL permite definir cuatro planos en cualquier posición y orientación, para hacer lo mismo con cualquiera de las cuatro coordenadas de textura. En CG y en particular en OpenGL, el plano $ax+by+cz+d=0$ se define mediante coordenadas $\{a,b,c,d\}$. Los primeros tres valores especifican un vector normal al plano y el cuarto es función lineal de la distancia del plano al origen (Ver el apéndice "Planos y Normales" en los apuntes de transformaciones). Consideremos un punto P en $\{x,y,z\}$ de coordenadas homogéneas $\{wx,wy,wz,w\}$ con $w\neq0$. Cada coordenada de textura se obtiene del producto escalar entre las coordenadas 4D del punto y las del plano correspondiente. Por ejemplo, para la coordenada s de textura se define un plano S$\{a_s,b_s,c_s,d_s\}$ y se calcula: $s(P) = wxa_s + wyb_s + wzc_s + wd_s$, que luego se clampea o repite. Esto da un número proporcional a la distancia entre el punto y el plano y así la utiliza OpenGL. El módulo del vector normal ($a^2+b^2+c^2$) es la inversa de la escala o la frecuencia con que se repetirá la textura. La componente d indica la posición del plano, y con ello la de la textura. Supongamos el plano S = yz: (a,0,0,0). Si $a=1 \Rightarrow s=x$; la imagen abarca una unidad espacial. Si $a=0.1 \Rightarrow s=x/10$; con lo cual s será 1 cuando x sea 10 unidades, la imagen ocupa 10 unidades; si se repite, se repite cada 10 unidades. La componente d se suma (w=1), con lo cual define el desplazamiento de la imagen.

En OpenGL se puede decidir si el plano se define en el mismo espacio en el que están dadas las coordenadas del objeto (espacio del modelo) o en un espacio fijo al ojo (espacio visual); también se puede definir en cuál de los dos espacios se realiza el producto escalar, es decir si las coordenadas del punto se toman del espacio del modelo o del ojo, esto da cuatro combinaciones posibles para elegir. Para que la textura aparezca fija a un objeto que se mueve, el plano debe estar definido en el espacio del modelo y moverse junto con el objeto. Si se utiliza el plano fijo al espacio del ojo, la textura parecerá proyectada sobre el objeto y "resbalará" cuando el objeto se mueva (reflejos).

Mapeo en dos partes y mapeo ambiental:

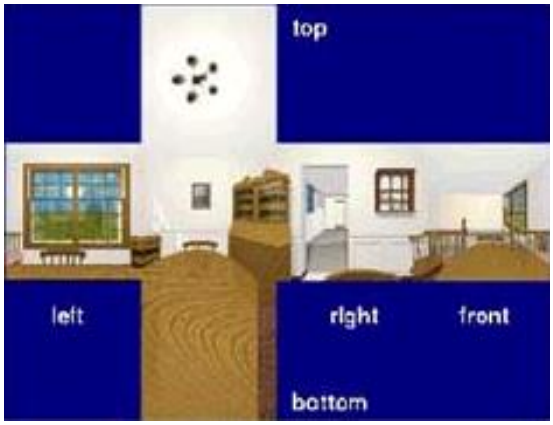
Consisten en mapear manual o algorítmicamente (UV) la textura sobre una superficie intermediaria muy simple: un plano, esfera, cubo o cilindro; para luego identificar de forma sencilla cada punto del objeto con algún punto la superficie intermedia. Esta técnica se conoce como mapeo en dos partes o en dos pasos (*two-pass mapping*); tiene un primer mapeo sobre la superficie intermediaria, el *S-mapping*, y el segundo es el *O-mapping*, sobre el objeto.

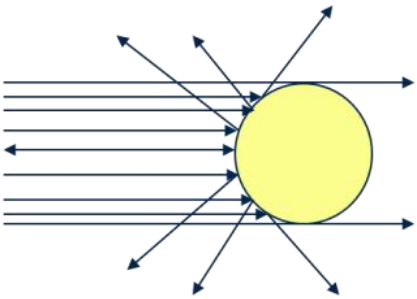
El mapeo sobre la superficie intermediaria (*S–maping*) es siempre simple: si se trata de una superficie biparamétrica P(u,v) se mapea la coordenada s en el rango de u y la t en el rango de v (normalmente en forma directa si el rango es [0,1]). En una esfera, por ejemplo, se puede considerar que s es la longitud $\theta = \text{atan2}(y,x) \in [0,2\pi]$ y t la latitud $\phi = \text{atan2}(z,\text{sqrt}(x^2+y^2)) \in [-\pi,\pi] \Rightarrow s = \theta/(2\pi)$; $t = ½+\phi/\pi$.

Para realizar el mapeo o proyección desde el intermediario al modelo (*O–maping*) hay varios métodos posibles. Normalmente el centroide (u otro punto central del objeto) se considera encerrado por la superficie intermediaria y se puede usar, por ejemplo, la línea que une el centro del objeto (*object centroid*) y cada punto de su superficie para asignar las coordenadas de textura del punto, donde esa línea corta al objeto intermediario. Otras técnicas de proyección son *object normal*¸ donde la línea se define mediante el punto y su normal e ISN o *intermediate surface normal* donde cada punto de la superficie intermedia proyecta sobre el objeto de acuerdo a la normal del intermediario.

Por último, en el método de *O–maping* por *reflected ray*, el rayo que va del ojo al punto del modelo rebota hasta chocar con la superficie intermediaria. Este método se utiliza para realizar mapeos de entorno o ambiente (*environment mapping*); considerando la textura pegada a la superficie intermediaria, que suele ser una semiesfera (piso y cielo) o un cubo (seis fotos del entorno).

La técnica del mapeo cúbico del ambiente tiene la gran ventaja de que el objeto intermediario se realiza con seis imágenes muy fáciles de obtener, pero no está exento de algunos problemas técnicos, pues las coordenadas de textura se deben asignar de acuerdo al rayo reflejado "en cada fragmento", no se pueden interpolar las coordenadas de textura calculando el reflejo sólo en los vértices de una primitiva pues puede suceder que cada vértice caiga en una cara distinta del cubo, generando problemas con la asignación de dos coordenadas en el cubo desarrollado sobre un plano (figura derecha); aun si se resuelve eso, se pierde la suavidad al pasar de una cara a otra.





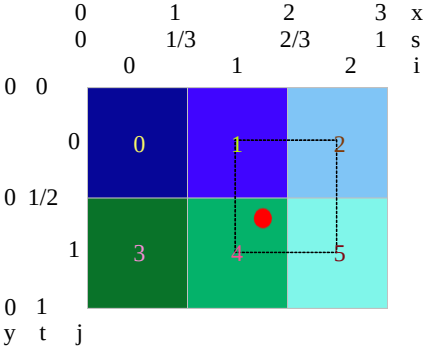
Textura

Mapeo de entorno esférico

El mapeo de entorno esférico es un truco para simular un mapeo de entorno por rayo reflejado. Utiliza una textura consistente en una imagen del entorno como si fuese una foto de una esfera espejada. Esa imagen se puede construir por medio de un algoritmo que la genere a partir de las seis caras de un cubo, también se puede generar falsa, pero creíblemente, deformando una sola foto. Contando con esa imagen y suponiendo al observador en el infinito, basta conocer la normal del objeto en el espacio del ojo, para definir en forma analítica las coordenadas de textura. La obvia desventaja es que solo funciona con cámara fija, si se quiere mostrar el objeto girando queda muy bien, pero si se quiere mover la cámara en una escena fija el objeto siempre refleja lo mismo en forma independiente del punto de vista, como si el universo detrás de la cámara se moviese con la cámara.

Filtrado (interpolación)

Cambiando el zoom o la distancia entre el objeto y la cámara, los téxeles "pegados" cambian de tamaño visual. Si acercamos el ojo (ampliación o magnificación) llega un momento en que vemos la textura pixelada. Si lo alejamos (reducción o minificación) cada fragmento puede ocupar varios téxeles, pero siempre hay que decidir un único color de textura por fragmento. Las transiciones serán más suaves y agradables a la vista si, en lugar de elegir el color del téxel donde cae el punto (s,t), se realiza un promedio ponderado de vecinos. Se puede considerar cualquier cantidad de vecinos y método de ponderación, pero lo más rápido y lo usual es utilizar los cuatro téxeles más cercanos al punto exacto y hacer con ellos una interpolación bilineal. Veamos los detalles.



En la memoria hay un arreglo bidimensional de colores al que se accede como C(i,j), pero está linealmente guardado como C(W*i+j). En coordenadas (x,y) de la imagen, un téxel es un cuadradito de 1x1; la imagen de WxH, tiene ancho W y altura H, reales en esas unidades. Las coordenadas de textura (s,t) varían, en el mismo rectángulo, pero entre (0,0) y (1,1) reales. Para el téxel (i,j), las coordenadas (s,t) varían entre (i/W,j/H) e ((i+1)/W,(j+1)/H).

La figura muestra una imagen de 3x2, indicando (x,y), (s,t) e (i,j) en los bordes, y en los centros el orden de almacenamiento W*i+j. También muestra un punto en ~(.6,.6), con s un poco menor que 2/3 y t un poco mayor que 1/2.

Para saber qué téxeles usar o promediar para definir el color C en función del par (s,t) del fragmento, primero hay que saber en qué téxel (i,j) caen esas coordenadas de textura. Usando las proporciones y el dibujo de referencia: (i,j) = (int(sW),int(tH)). Ese es el téxel que tiene el centro más cercano al punto (*nearest*). Es el que se usa cuando no se necesita o no se quiere suavizar:

Nearest: (i,j)=(int(sW),int(tH)) C=C(i,j)

Para interpolar entre los cuatro vecinos más cercanos, primero hay que identificarlos. Imagínese al téxel *nearest* dividido en cuatro, el cuadrante en el que cae el punto define cuáles son los cuatro que se deben interpolar. El punto queda dentro de un cuadrado formado por los cuatro centros vecinos y sus colores se interpolan en forma bilineal. El vértice de menores coordenadas es: (i,j) = (int(sW−.5),int(tH−.5)) los demás son: (i+1,j), (i,j+1) e (i+1,j+1). Si los consideramos en ese orden, las distancias al "origen" (i,j) definen el par (u,v) para interpolar, pero un simple análisis muestra que el par (u,v) está dado por las fracciones truncadas:

Linear: (S,T)= (sW−.5,tH−.5); (i,j) = (int(S),int(T)) (u,v) = (frac(S),frac(T)) = (S−i,T−j);

C=(1−v)[(1−u) C(i,j)+ uC(i+1,j)]+ v[(1−u) C(i,j+1)+u C(i+1,j+1)]

La interpolación bilineal se muestra como una interpolación lineal vertical, con parámetro v, de dos interpolaciones lineales horizontales con parámetro u, pero redistribuyendo da lo mismo.

Cuando se hace un acercamiento, ampliación o magnificación, cada téxel cubre más de un píxel y esos son los dos filtros prefabricados que se pueden utilizar: *nearest* y *linear*. También se podrían utilizar otros filtros que den mejores resultados, interpolando entre más vecinos para definir mejor los bordes o suavizar más, pero habrá que hacerlo mediante un programa en el *fragment-shader* y asumiendo el alto costo de acceder a valores alejados en la memoria ordenada por filas.

Por el contrario, cuando se aleja mucho el objeto, se hace una reducción o minificación, los téxeles pueden hacerse mucho más pequeños que los píxeles. Si un fragmento ocupa varios téxeles y solo usa uno o cuatro téxeles centrales, la imagen saldrá muy mal; pues, el fragmento de al lado, utilizará otros muy distintos. Sería más razonable que promedie los colores de todos los téxeles que atrapa; pero eso es mucho trabajo, aún para un *fragment shader*. Una solución sencilla consiste en reducir la imagen original varias veces a la mitad promediando vecinos y finalmente promediar promedios al llegar al tamaño adecuado; esa solución sí está implementada en el sistema clásico de OpenGL con funcionalidad fija y el promedio de promedios automáticamente tiene en cuenta varios téxeles alrededor de cada punto.

Se predefinen varias copias de la misma textura en resoluciones decrecientes; cada imagen con la mitad de ancho y de alto, hasta llegar a 1x1, aun cuando la imagen sea rectangular. El espacio ocupado por esta serie de imágenes llamadas *mipmaps* no es mayor que una vez y media el espacio ocupado por la mayor de ellas. OpenGL trae, en la biblioteca GLU, una función que permite cargar la imagen mayor y construye automáticamente los *mipmaps*. Al hacer reducciones sucesivas, reemplazando cada grupo de cuatro téxeles por su promedio, los defectos no son tan visibles como si se hiciera de golpe una gran reducción, pues se van promediando los promedios previos.

Contando con los *mipmaps,* se puede elegir el tamaño más adecuado (*mipmap nearest*) o bien realizar una interpolación lineal entre los dos niveles de tamaño más próximos al necesario (en OpenGL es *mipmap-linear*). A su vez, en la imagen elegida o las dos cercanas, se puede utilizar, como antes, un téxel o una interpolación, dando cuatro combinaciones posibles.

La selección (automática) del nivel de *mipmap* a utilizar depende del grado de minificación, esto es la relación entre el tamaño del pixel y el del téxel. Se calculan los gradientes ∇s y ∇t en el espacio de la imagen (unidades píxel), que luego se multiplican por W y H, se elige el de mayor módulo y se toma el logaritmo en base dos, para obtener el nivel de *mipmap* o los dos más cercanos.

Mezcla

Una vez fijado el color de la textura para el fragmento, resta definir cómo se mezcla con el color del fragmento, interpolado o calculado del modelo de iluminación. OpenGL permite definir cuatro métodos:

- *Replace* (Sustitución)
- *Decal* (Mezcla con transparencia – Calcomanía)
- *Modulate* (Modulación 0-1)
- *Blend* (Mezcla utilizando un tercer color)

El método *replace*, remplaza el color preexistente en el fragmento por el que proviene de la textura, es el método más económico y adecuado para pegar una textura mate (sin brillo) o un reflejo perfecto.

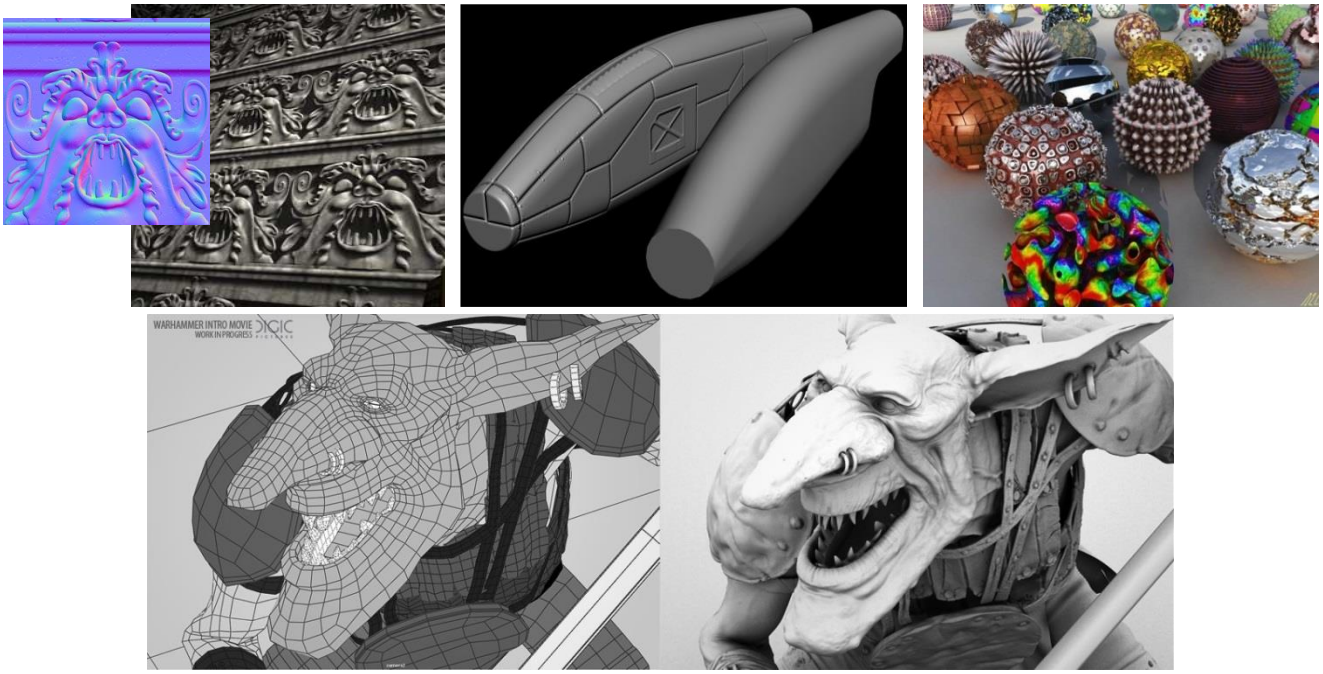
Decal significa calcomanía. Las partes de la textura con alfa menor que uno son semitransparentes y las que tienen alfa nulo son totalmente invisibles. El color de la textura se mezcla con el del objeto en proporciones que dependen del alfa de la textura. En el proceso, el alfa del fragmento no se altera.

La modulación es un proceso como el de modular una luz para atenuarla, consiste en multiplicar el color y el alfa del fragmento por el color y el alfa de la textura, componente a componente. Siendo números entre cero y uno, al multiplicarlos se acercan a cero, disminuyendo el valor original. Permite hacer, por ejemplo, manchas oscuras en la superficie o definir iluminación diferente en cada zona (*light mapping*) simulando la iluminación de una geometría compleja, pero en una geometría simple.

El modo *blend* utiliza un tercer color fijo (cuyo alfa no importa) y los valores de la textura se usan como parámetro para interpolar en cada canal R, G o B, entre el color fijo y el que traía el fragmento. Este método permite hacer, por ejemplo, un marmolado de dos colores o vetear u oxidar una superficie. El alfa del fragmento se multiplica por el de la textura sin intervención del color fijo.

Otras aplicaciones

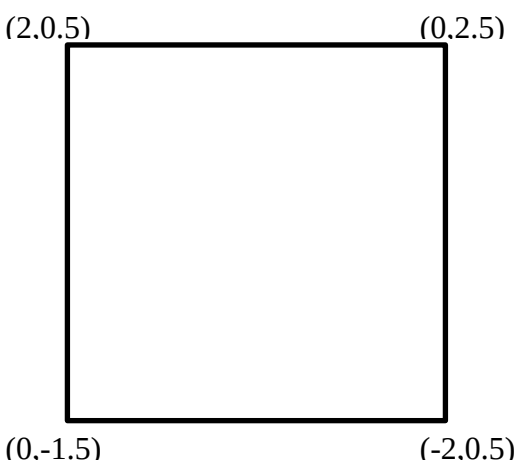
Debido a que con el tiempo se han ido desarrollando estructuras de hardware para manejar las texturas y el proceso de aplicación en forma muy rápida y eficiente en la GPU, las texturas han adquirido muchas aplicaciones distintas de la original. Actualmente las texturas se aplican para trasmitir cualquier variable 4D a los fragmentos, antes de componer la imagen final; los datos van codificados en el vector de valores RGBA de cada téxel y el programador decide cómo y para qué los utiliza. La primera aplicación extra fue el *bump mapping*, que consiste en usar esos valores para modificar la normal de la superficie, reproduciendo la iluminación que tendría una superficie rugosa o con relieves variables. En los *shaders* geométricos se pueden utilizar para modificar efectivamente la posición espacial de la geometría (*displacement mapping*) o modificar las componentes del material en el modelo de iluminación.

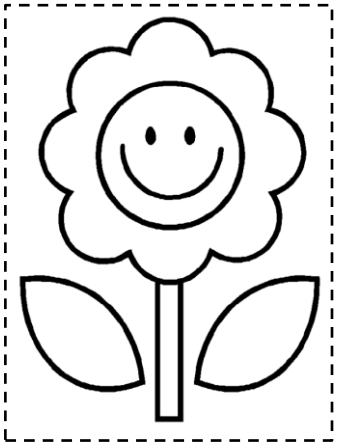


Ejercicios:

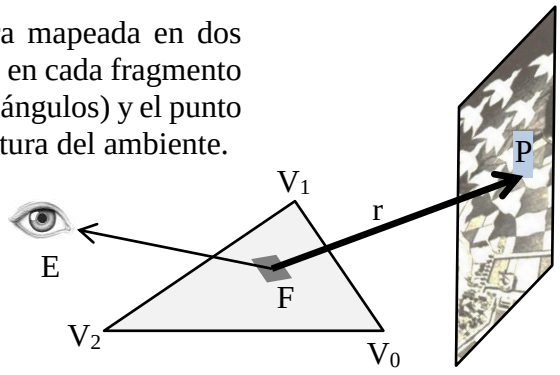
- a) ¿Cómo se aplica textura a un objeto?
  b) ¿Cómo asigna las coordenadas de textura un mapeo plano?

- Los vértices de un cuadrado tienen las coordenadas de textura que indica la figura. La imagen de la flor es la textura que se mapea mediante *clamping* de las coordenadas de textura (la imagen aparece sin repetición).
  Dibuje el resultado (indique el marco y esquematice la posición).
  ¿Por qué el contenido de la imagen (la flor) no debe llegar al borde de la imagen?





- Para realizar reflejos del entorno, mediante una textura mapeada en dos partes, se requiere conocer el rayo reflejado por el objeto en cada fragmento de cada primitiva que lo conforma (asumamos que son triángulos) y el punto de impacto P sobre la superficie intermediaria con la textura del ambiente. Determine la ecuación del rayo reflejado r (no se pide averiguar P) conociendo la posición E del observador, las coordenadas Vi de los vértices del triángulo y las tres coordenadas baricéntricas α_i del fragmento en F. Ayuda: El paso final se consigue utilizando dos componentes para el vector (E − F) y para su reflejo.



- Conocemos las cotas, respecto al nivel del mar, en un gran conjunto de puntos de la ciudad. ¿Cómo podemos armar un mapa visual de cotas sobre el plano de la ciudad?

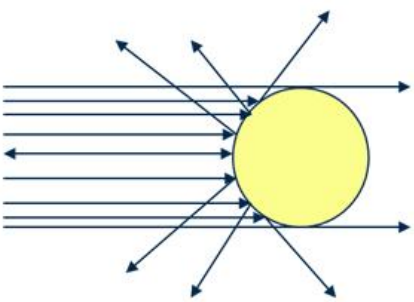
- ¿Cómo se mapea una calcomanía de WxH sobre un cilindro de radio r y altura h, de modo que ocupe 90º a lo ancho, sin deformar la imagen y quede centrada en altura?

Apéndice I: Ecuaciones del mapeo de entorno esférico

Para lograrlo, se utiliza una textura que es (o simula) el reflejo del ambiente en una esfera espejada, centrada en el entorno y supuestamente infinitesimal o, equivalentemente, vista desde el infinito.

La imagen debería cambiar junto con el punto de vista; por lo tanto, este mapeo sólo tiene sentido si el ojo se considera fijo y, en tal caso, es superior al mapeo cúbico, pues las coordenadas de textura se pueden interpolar a partir de los vértices de las primitivas.

La imagen se puede construir con seis fotos o *renderings* con 90° de apertura, tomadas en dirección a las seis caras de un cubo; pero contando con esas fotos, sería preferible un mapeo cúbico que permite cambiar el punto de vista, sin reconstruir la imagen de textura.



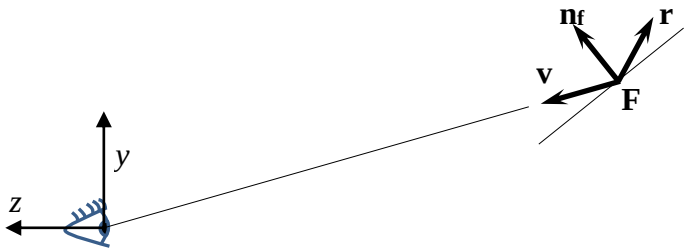
Textura

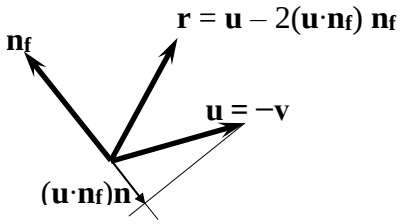
Mapeo de entorno esférico

El mapeo consiste en buscar el punto de la esfera que tendría la misma dirección de reflejo que el fragmento del objeto y con eso, ubicar el téxel (s,t) en la imagen. La esfera está centrada y ocupa toda la textura $(s,t)\in[0,1]^2$; la normal $\mathbf{n_s}$ de la esfera, con componentes dadas en el espacio de la imagen, define biunívocamente la posición en la imagen:

$$s = ½ + ½\, n_{sx} \qquad\qquad t = ½ + ½\, n_{sy}$$

En el caso general, hay que calcular el reflejo del versor **v**, que apunta desde el fragmento en $F\{x,y,z\}$ hacia el ojo en $O\{0,0,0\}$. Reflejando **v** mediante la normal $\mathbf{n_f}$, se obtiene el versor **r**. Por simplicidad, utilizaremos el vector posición normalizado, que resulta opuesto a **v**: $\mathbf{u} = -\mathbf{v} = \{x,y,z\}_1$:





El resultado no necesita normalizarse pues: $\mathbf{r}^2 = [\mathbf{u}–2(\mathbf{u}\cdot\mathbf{n_f})\mathbf{n_f}]^2 = \mathbf{u}^2 – 2\mathbf{u}\cdot[2(\mathbf{u}\cdot\mathbf{n_f})\mathbf{n_f}] + 4(\mathbf{u}\cdot\mathbf{n_f})^2\mathbf{n_f}^2 = 1$.

Para que la esfera infinitesimal provoque el mismo reflejo **r**, su normal $\mathbf{n_s}$ debería ubicarse a medio camino entre **r** y el versor $\mathbf{e_z}$; siendo ambos unitarios, $\mathbf{n_s}$ se obtiene mediante la suma normalizada:

$$\mathbf{n_s} = (\mathbf{r} + \mathbf{e_z}) / |\mathbf{r} + \mathbf{e_z}| = (\mathbf{r} + \mathbf{e_z}) / \sqrt{\mathbf{r}^2 + 2\,\mathbf{r}\cdot\mathbf{e_z} + \mathbf{e_z}^2} = (\mathbf{r} + \mathbf{e_z}) / \sqrt{2\,(1 + r_z)}$$

$$s = ½ + ½\, r_x / \sqrt{2\,(1 + r_z)} \qquad\qquad t = ½ + ½\, r_y / \sqrt{2\,(1 + r_z)}$$

Para poner las ecuaciones como se especifican en OpenGL (en una forma retorcida que sólo tiene sentido si no se normaliza **u**) se reescribe el denominador en forma equivalente:

$$s = ½ + ½\, r_x / \sqrt{r_x^2 + r_y^2 + (1 + r_z)^2} \qquad\qquad t = ½ + ½\, r_y / \sqrt{r_x^2 + r_y^2 + (1 + r_z)^2}$$

Si el ojo se puede considerar infinitamente alejado del objeto, la dirección de los rayos visuales es siempre la misma, independientemente de la posición del fragmento. El objeto reflejará lo mismo que la esfera en el lugar en que la normal $\mathbf{n_s}$ sea la misma que la $\mathbf{n_f}$ del fragmento; es decir que el téxel solo depende de la normal del fragmento y no hace falta calcular el rayo reflejado:

$$s = ½ + ½\, n_{fx} \qquad\qquad t = ½ + ½\, n_{fy}$$