PROGRAMACIÓN 3D

Máster en Programación de Videojuegos

TEMA 8: NormalMapping

Juan Mira Núñez

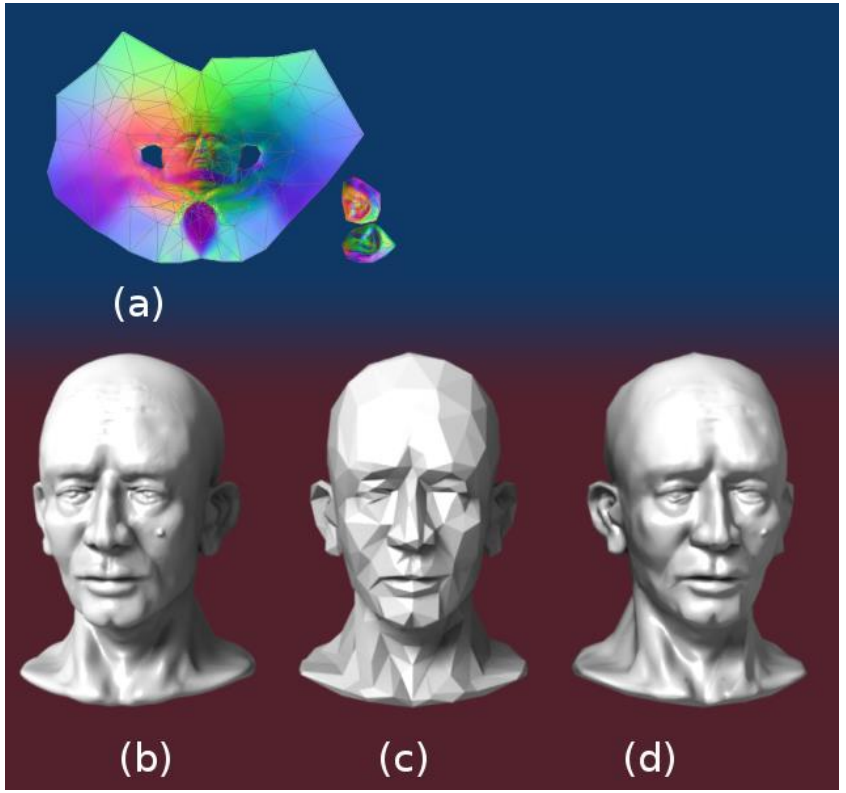
U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

Índice

- Introducción
- NormalMapping
- Matriz tangete
  - Vertex shader
  - Fragment shader

Introducción.

- NormalMapping es la aplicación de una técnica 3D que permite dar una iluminación y relieve mucho más detallado a la superficie de un objeto.
- Es una evolución del mapeado topológico (bump mapping) y se aplica en videojuegos, principalmente en detalles pequeños como arrugas o poros, así como en películas de animación o escenas 3D para agilizar los cálculos y reducir por tanto el número de polígonos con los que en un principio contaban los objetos.



Introducción.

- Mientras que el mapeado topológico (bump mapping) sería un mapa de relieve en el que solo se representa la altura, el mapeado normal representa los tres ejes: X, Y, Z.
- Luego la información por píxel no se aplica en tonos de grises el cual representaría la altura, sino en colores RGB dando más fidelidad al original que se quiere imitar.
- Este efecto recrea el relieve de una malla detallada, aunque el observador al acercarse a dicho objeto perderá la sensación de relieve ya que se trata solo de un efecto visual creado por los shaders.

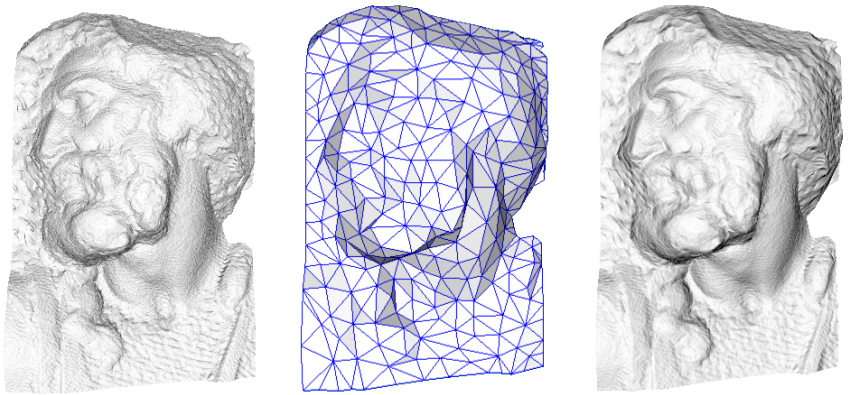
NormalMapping

- Sin normalmapping

- Con normalmapping

NormalMapping

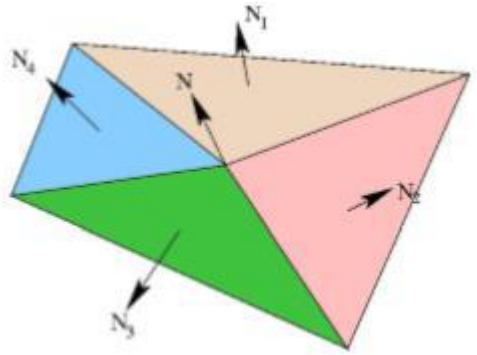
- La idea es conseguir que un modelo 3D siga teniendo detalle en sus superficies a pesar de tener pocos polígonos.
- De esta manera conseguimos conservar el detalle que tendría el modelo si estuviera renderizado con un número alto de polígonos.

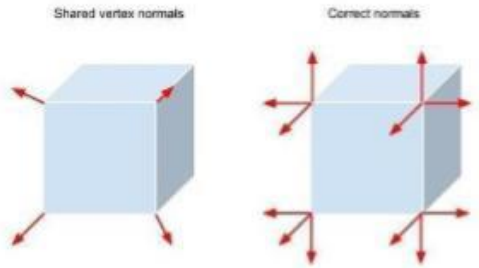


- Usamos la iluminación para simular este “detalle” en los modelos.

NormalMapping

- Ya sabemos que la iluminación se puede realizar a nivel de vértice o de fragmento.
- En ambos casos, usamos la normal al vértice para los cálculos de iluminación en superficies.

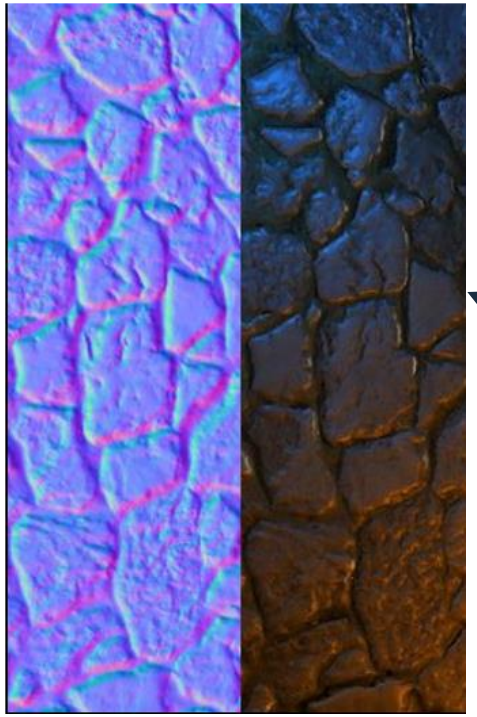




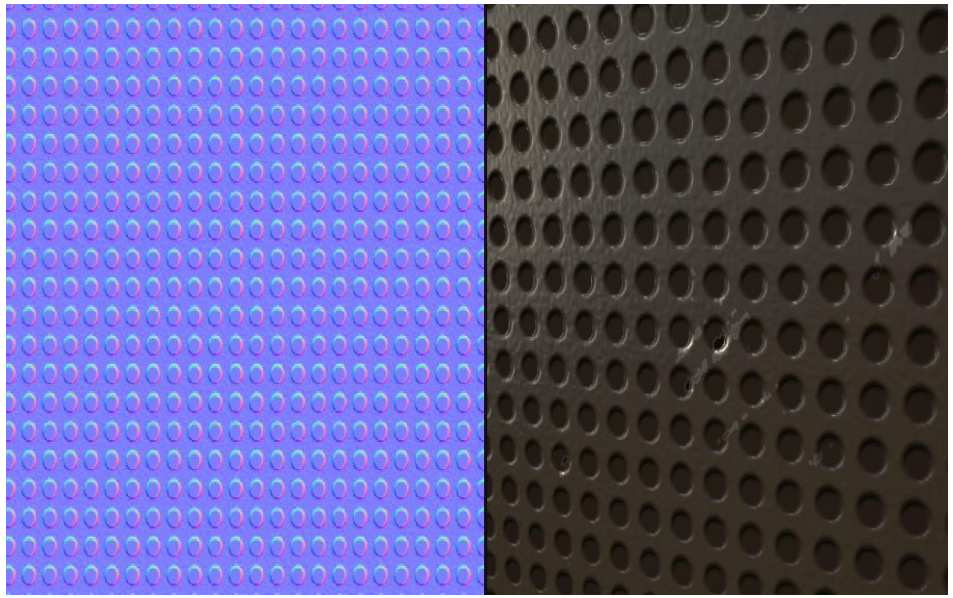
- Con el normalmapping vamos a almacenar las normales en una textura, con lo que tendremos esta información a nivel de fragmento.

NormalMapping

NormalMap



Resultado



Textura

- **Texturas NormalMap**
  - Las normales están definidas en el rango [-1, 1]
  - Al estar almacenadas en una textura, sus valores vendrán dados en el rango [0,1]
  - Tenemos que convertir estos valores.
- **Vector Normal en Textura:**
  - En la textura tenemos los valores XYZ representados en RGB.
  - Un pixel que representa el vector perpendicular a la superficie, sin desviaciones ni en X ni en Y, es (0,0,1)
  - Si hacemos (1, 0, 0) tenemos un vector con la normal del triangulo desviada 90 grados a la derecha.
  - Estos valores no tienen en cuenta su posición en el objeto/escena ni su orientación en el espacio del observador. Tenemos que construir una matriz para convertirlos al espacio de observador.

Tangente

- **Espacio Tangente:**

Hemos visto el espacio de proyección, de observador y de modelo.

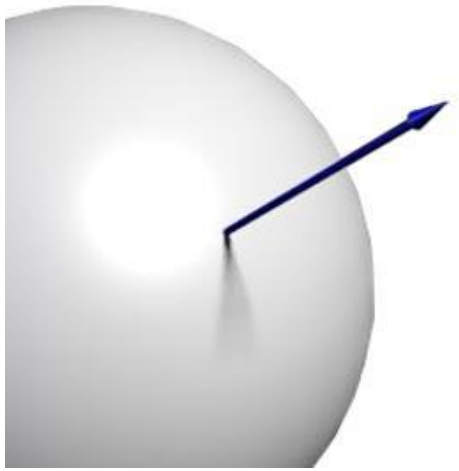
- Vamos a usar un nuevo espacio, el espacio tangente, para las **desviaciones sobre la normal** que tenemos guardadas en la textura.
- Lo primero es pasar los datos de la normal de la textura, en espacio tangente, combinados con la normal a cada superficie , al espacio del observador.
- Las normales representan orientaciones en el espacio, así que cambiaremos de espacio mediante una **matriz de rotación**.
- Las tres columnas de la matriz representan una transformación sobre los ejes X, Y, y Z respectivamente, que permiten pasar la normal del normal map a su correspondiente orientación en el espacio del observador de manera relativa a la superficie del modelo.

Tangente

- **Espacio Tangente:**

$$\begin{vmatrix} tangente.x & bitangente.x & normal.x \\ tangente.y & bitangente.y & normal.y \\ tangente.z & bitangente.z & normal.z \end{vmatrix}$$

La normal de un triángulo conforma la columna del eje Z.

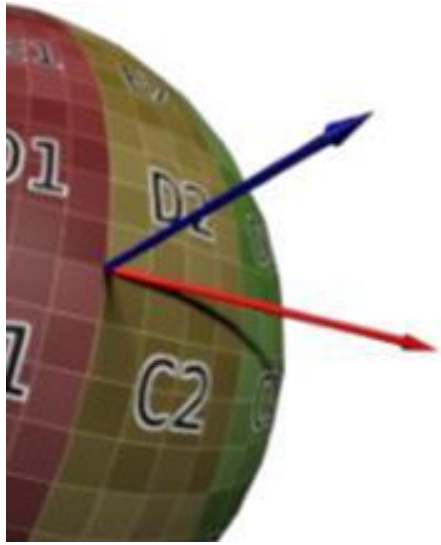


Tangente

- **Espacio Tangente:**

$$\begin{vmatrix} tangente.x & bitangente.x & normal.x \\ tangente.y & bitangente.y & normal.y \\ tangente.z & bitangente.z & normal.z \end{vmatrix}$$

El vector tangente del vértice (añadido igual que las normales en el modelo) conforma el eje X.

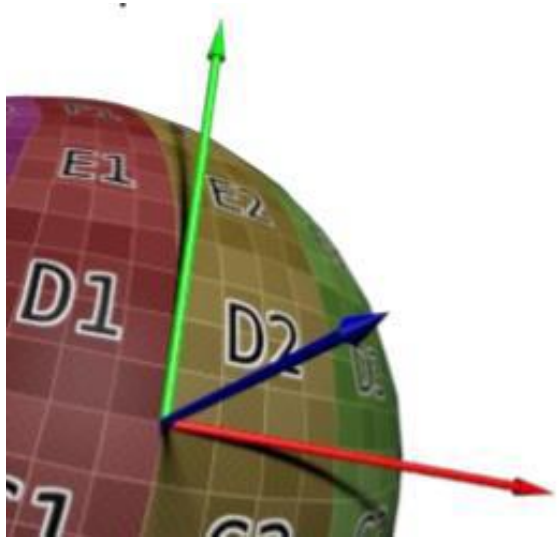


Tangente

- **Espacio Tangente:**

$$\begin{vmatrix} tangente.x & bitangente.x & normal.x \\ tangente.y & bitangente.y & normal.y \\ tangente.z & bitangente.z & normal.z \end{vmatrix}$$

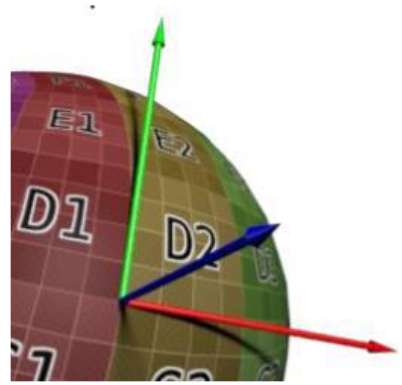
El vector bitangente es el producto vectorial de los otros dos y conforma el eje Y.



Tangente. Vertex shader

El vector normal y el vector tangente están en el espacio del modelo, los convertimos al espacio del observador antes de colocarlos en la matriz.

```
fnormal = normalize((mNormal * vec4(vnormal, 0.0)).xyz);
vec3 tangent = (mNormal * vec4(vtangent, 1.0)).xyz;
vec3 bitangent = cross(fnormal , tangent);
```



Está matriz se genera en el vertex shader y la pasamos al fragment shader

```
mTbn = mat3(tangent , bitangent , fnormal);
```

Tangente. Fragment shader

En el fragment shader , obtenemos el vector normal de la textura con la función texture2D y multiplicándolo por la matriz, obtenemos la normal en el espacio del observador.

```
N = normalize(fnormal);

N = texture2D(normalTexture, ftex).rgb;

N = normalize(N * 2.0 - 1.0); // Convertir al rango [-1.0, 1.0].

N = normalize(mTbn * N);
```

Este valor, sustituye a la normal al vértice en todos los cálculos de iluminación.

Ejemplo Vertex shader

```
//...
attribute vec3 vtangent;
varying mat3 mTbn;
//...
void main (void)
{
//...
        vec3 tangent = mNormal * vec4(vtangent, 1.0).xyz;
        vec3 bitangent = cross(fnormal , tangent);
        mTbn = mat3(tangent , bitangent , fnormal);
//...
}
```

Ejemplo fragment shader

```
Void main(void)
{
//...
N = normalize(fnormal);
If(useNormalTexture) {
        N = texture2D(normalTexture , ftex).rgb;
        N = normalize (N * 2.0 - 1.0); // pasamos al rango [-1.0, 1.0].
        N = normalize(tbn * N);
}
//...
}
```

¿Dudas?