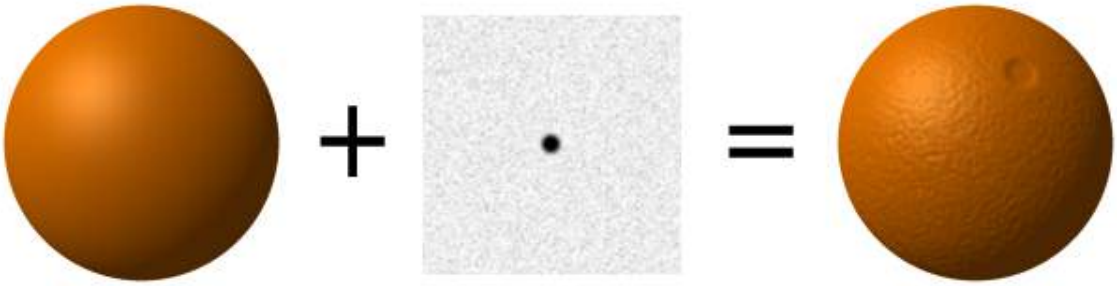
DOCUMENTACIÓN DE LA IMPLEMENTACIÓN:

Bump Mapping



informatika fakultatea
facultad de informática

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Iker Pintado García

Índice

1. Introducción

Esta técnica permite simular la rugosidad de los objetos, otorgando a la práctica un nivel añadido de realismo. Esto se obtiene aplicando pequeñas modificaciones en la normal de la superficie del objeto, dando lugar a variaciones en la iluminación del mismo, otorgando este efecto de rugosidad.

Para aplicar estas variaciones en la normal existen varias técnicas. La que se va a utilizar es la técnica *bump mapping*, que utiliza un mapa de textura para saber cómo debe ser la modificación de esta normal y el espacio tangente del vértice.

2. Implementación

Para la implementación de esta técnica solamente ha sido necesario crear tanto el vertex como el fragment shader. En el caso del fragment shader, se ha copiado el de **perfragment** con un par de modificaciones. Estas modificaciones se relacionan con la obtención de la normal del fragmento. Una versión inicial de esta normal se obtiene con la función **texture2D(bumpmap, f_texCoord)** pero sus valores se encuentran entre 0 y 1, y los queremos entre -1 y 1, por lo que el vector resultante lo multiplicaremos por 2 para obtener un rango entre 0 y 2, y le restaremos 1 para pasar el rango a entre -1 y 1:

```
.
.
.
vec4 f_color;
vec4 normal = 2.0 * texture2D (bumpmap, f_texCoord) - 1.0;
vec3 L, N, V;
vec3 acum_dif, acum_esp;
acum_dif = vec3(0.0);
acum_esp = vec3(0.0);
N = normalize(normal.xyz);
V = normalize(f_viewDirection);
.
.
.
```

Por la parte del vertex shader, trasladaremos **f_viewDirection**, **f_lightDirection (de todas las luces)**, **f_spotDirection (de todos los focos)** al espacio tangente multiplicando por la matriz de cambio de sistema de referencia de este (tangente en el espacio de la cámara, binormal en el espacio de la cámara, normal en el espacio de la cámara):

```
.
.
.
vec3 n,t,bn;
n = (modelToCameraMatrix * vec4(v_normal,0.0)).xyz;
n = normalize(n);
t = (modelToCameraMatrix * vec4(v_TBN_t,0.0)).xyz;
t = normalize(t);
bn = (modelToCameraMatrix * vec4(v_TBN_b,0.0)).xyz;
bn = normalize(bn);
vec3 v_pos = (modelToCameraMatrix * vec4(v_position,1.0)).xyz;
vec3 tmp;
tmp.x = dot(v_pos,t);//simulación de la matriz
tmp.y = dot(v_pos,bn);
tmp.z = dot(v_pos,n);
f_viewDirection = tmp;
vec3 tmp2, tmp3;
```

```
for(int i = 0; i < active_lights_n; i++){
    if(theLights[i].position.w == 0.0){
        tmp2 = -1.0 * theLights[i].position.xyz;
    }else{
        tmp2 = theLights[i].position.xyz - v_pos;
        if (theLights[i].cosCutOff != 0){ //si es foco
           tmp3 = theLights[i].spotDir.xyz;
           tmp.x = dot(tmp3,t);
           tmp.y = dot(tmp3,bn);
           tmp.z = dot(tmp3,n);
           f_spotDirection[i] = tmp;
        }
   }
   tmp.x = dot(tmp2,t);
   tmp.y = dot(tmp2,bn);
   tmp.z = dot(tmp2,n);
   f_lightDirection[i] = tmp;
}
.
.
.
```

3. Resultado en ejecución

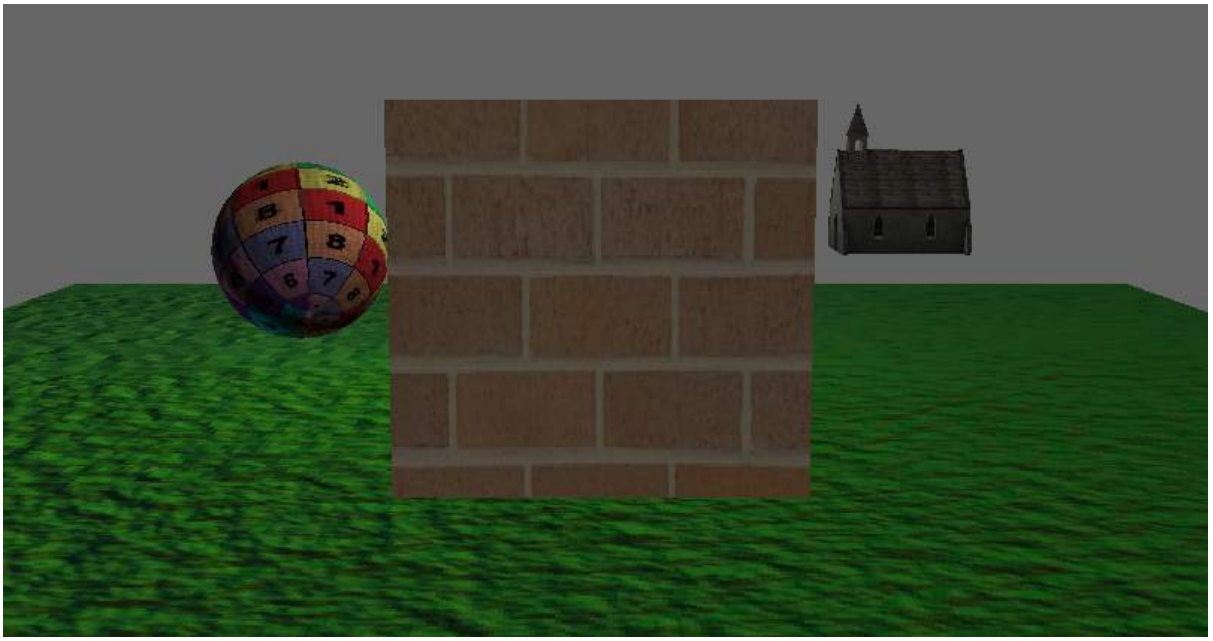


Fig. 1. Previsualización del mapa del shader

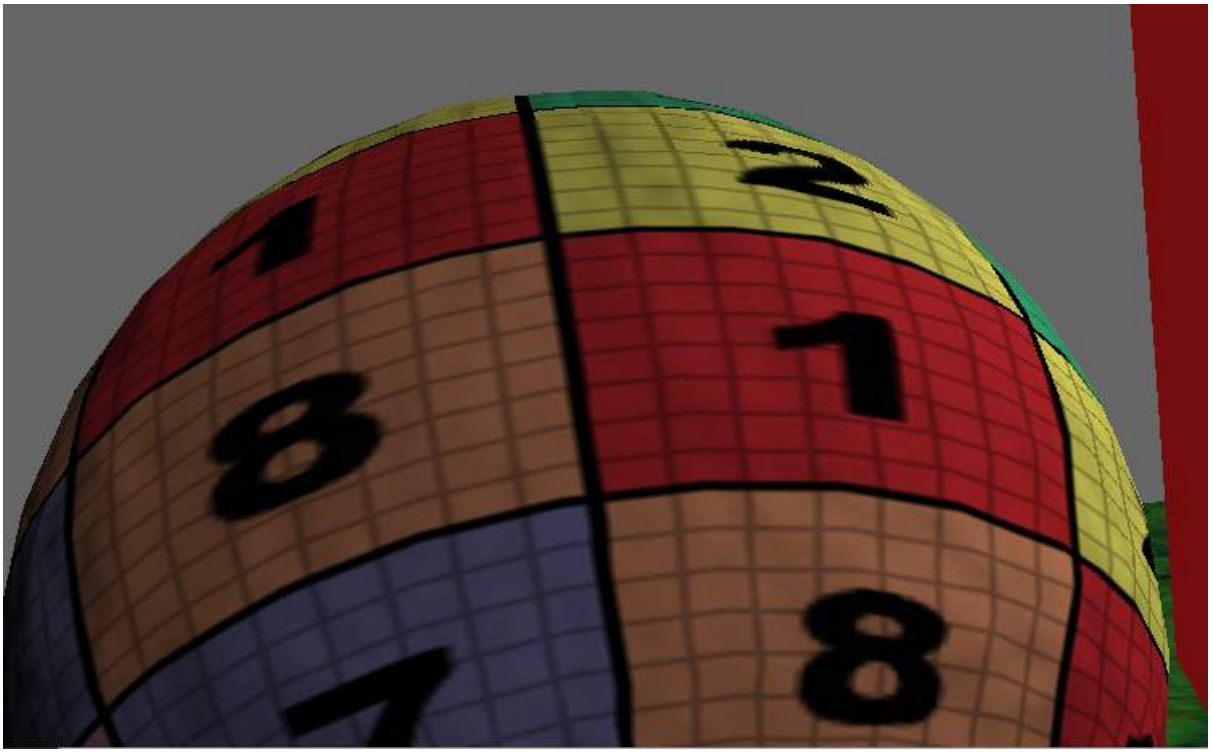


Fig. 2. Zoom a la pelota rugosa

4. Fuentes externas

- LearnOpenGL
- FabienSanglardWebsite