Renderizado de modelos 3D utilizando iluminación de tipo toon shading

Walter Ariel Baya

Agosto 2021

1 Introducción

Todo el proceso de implementación del shader que se realizará en este trabajo sigue la propuesta del libro computer graphics, principles and practice.

Para realizar toon shading, se computa el producto escalar entre la normal de cada vertice y la dirección de la luz, la diferencia a los procedimientos usuales de renderizado realista, es que en este caso, se realiza un proceso de umbralización del color, lo cual lleva a tener una imagen en la que hay una variedad de colores mucho menor a la que se tendría en el otro caso, obteniendo sombreados similares a los de los cartoons.

Existen muchas variaciones posibles de toon shading: se puede hacer la umbralización de la manera en la que se realizará en este trabajo, pero también es posible utilizar un modelo de iluminación por debajo como un phong shading o gouraud shading y luego aplicar la umbralización, es posible utilizar varias umbralizaciónes o tener variaciónes en la intensidad de la luz más complejas a la luz de singular.

2 Implementación del toon shading

2.1 Implementación utlizando el vertex shader

Lo primero que se realizará es computar el producto escalar entre la normal en cada vértice y el vector de la luz (la intensidad), esto se hace para cada vértice dentro del vertex shader, luego simplemente WebGl interpolará cada valor para cada triangulo y luego se limitarán en el fragment shader las intensidades resultantes.

Si se efectúa el shading utlizando el vertex shader y sin realizar un normalización previa de la normal en cada vértice, cuando se interpole la intensidad a través de cada triangulo y luego se realice la umbralización se obtendra como resultado que habrá bordes rectos, como cortes (bordes poligonales) entre cada par de regiones coloreadas y ese efecto no es lo deseado.

```
148
149     // Vertex Shader
150     var meshVS = `
151         precision mediump float;
152
153         attribute vec3 vertex;
154         attribute vec3 normal;
155         attribute vec2 tex;
156
157         uniform mat4 mvp;
158         uniform mat4 permutacion;
159         uniform vec3 wsEyePosition;
160         uniform int permutar;
161
162         varying vec3 wsInterpolatedNormal;
163         varying vec2 texCoord;
164
165
166         void main()
167         {
168             wsInterpolatedNormal = normal;
169             texCoord = tex;
170
171             if(permutar == 1){
172                 gl_Position =  mvp * permutacion * vec4(vertex,1);
173             }
174             else{
175                 gl_Position = mvp * vec4(vertex,1);
176             }
177         }
178     `;
179
```

```
184     // Fragment Shader
185     var meshFS = `
186
187         precision mediump float;
188         uniform sampler2D texGPU;
189         uniform vec3 wsEyePosition;
190         uniform int mostrar;
191         varying vec2 texCoord;
192         varying vec3 wsInterpolatedNormal;
193
194         void main()
195         {
196             vec4 kd = texture2D(texGPU, texCoord);
197             float intensity = dot(wsInterpolatedNormal,wsEyePosition);
198             if(mostrar == 1){
199                 if (intensity > 0.95){
200                     gl_FragColor = kd;
201                 }
202                 else if (intensity > 0.4){
203                     gl_FragColor = kd * 0.6;
204                 }
205                 else{
206                     gl_FragColor = kd * 0.2;
207                 }
208             }
209             else{
210                 kd = vec4(1.0,0.0,0.0,0.0);
211             if (intensity > 0.95)
212                 gl_FragColor = kd;
213             else if (intensity > 0.5)
214                 gl_FragColor = kd * 0.6;
215             else if (intensity > 0.25)
216                 gl_FragColor = kd * 0.4;
217             else
218                 gl_FragColor = kd * 0.2;
219             }
220         }
221     `;
```

Figure 1: Primer intento de shading, vertex y fragment shader

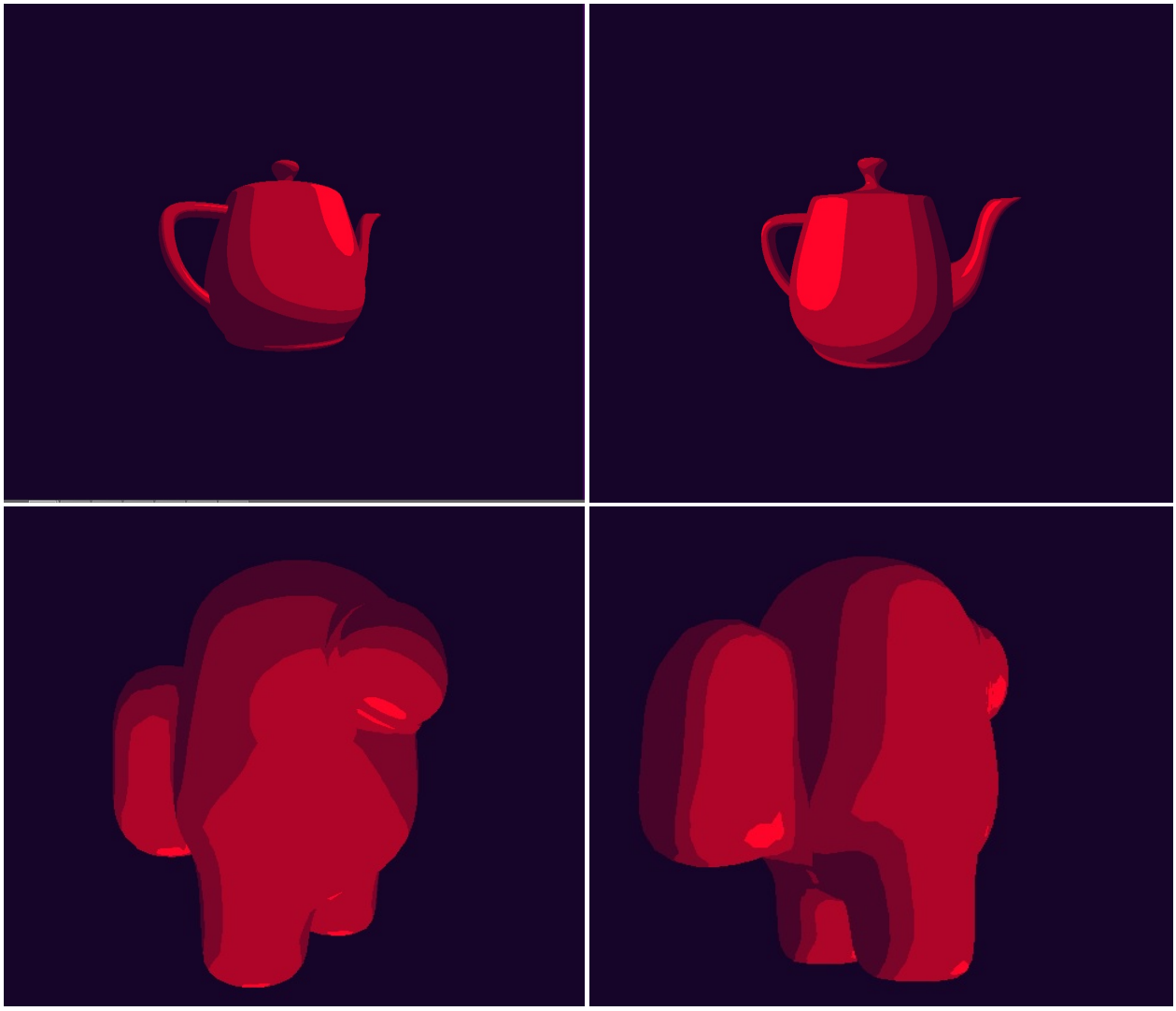

Figure 2: Shading utilizando el vertex shader y la normal no normalizada

2.2 Implementación utilizando el fragment shader

Es posible mejorar esto utilizando las normales normalizadas previamente y luego interpolar el vector de la luz en el fragment shader para computar el valor de la intensidad, eso va a gener una variación más suave a través de los polígonos(triángulos), lo cual al momento de generar la umbralización llevará a tener bordes mas suaves entre las regiones coloreadas, dando un efecto más agradable a la vista.

Se pueden observar debajo los cambios que se realizaron en el fragment shader,los mismos consisten en modificar la umbralización realizando algunos cambios, pero más importante y destacable resulta la normalización previa de cada normal a cada vértice, lo cual tendrá una gran contribución para lograr el suavizado de las lineas entre los distintos bloques de color.

```
// Fragment Shader
var meshFS = `

    precision mediump float;
    uniform sampler2D texGPU;
    uniform vec3 wsEyePosition;
    uniform int mostrar;
    varying vec2 texCoord;
    varying vec3 wsInterpolatedNormal;

    void main()
    {
        vec4 kd = texture2D(texGPU, texCoord);
        float intensity = dot(normalize(wsInterpolatedNormal),wsEyePosition);
        if(mostrar == 1){
            if (intensity > 0.95){
                gl_FragColor = kd;
            }
            else if (intensity > 0.4){
                gl_FragColor = kd * 0.6;
            }
            else{
                gl_FragColor = kd * 0.2;
            }
        }
        else{
            kd = vec4(1.0,0.0,0.0,0.0);
            if (intensity > 0.95){
                gl_FragColor = kd;
            }
            else if (intensity > 0.4){
                gl_FragColor = kd * 0.6;
            }
            else{
                gl_FragColor = kd * 0.2;
            }
        }
    }
`;
```

Figure 3: Segundo intento de shading, vertex y fragment shader

Todos estos cambios se llevaron a cabo dentro del fragment shader, es fácil notar que tanto el personaje de Among Us y la tetera tienen sombreados con bordes mas suaves entre las regiones roja no sombreada y roja sombreada, dando un efecto de caricatura mucho más agradable y menos poligonal.

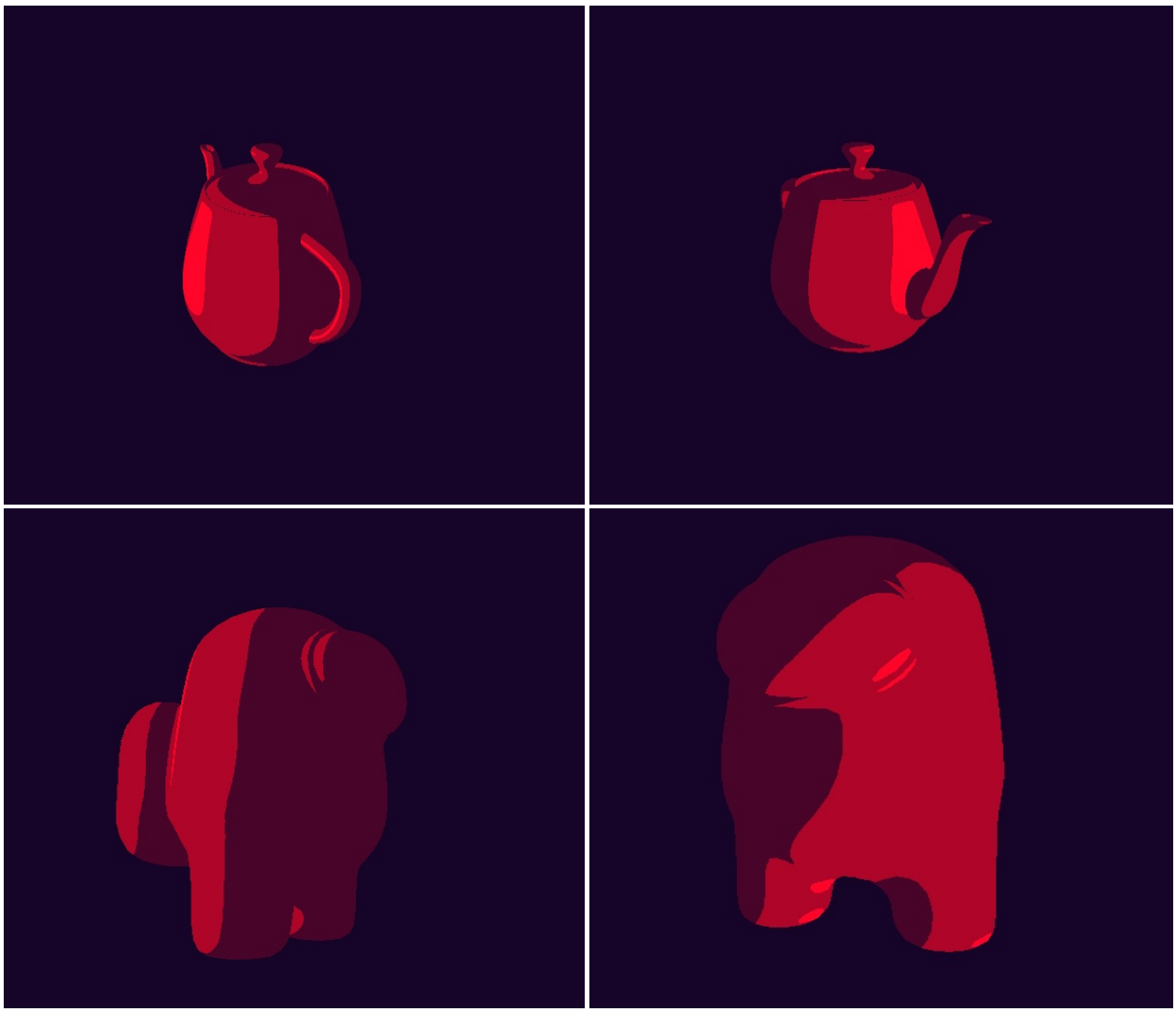

Figure 4: Shading utilizando el fragment shader y la normal normalizada