



**DEPARTAMENTO  
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Final

Toon Shader

Fundamentos de la Computación Gráfica  
1<sup>er</sup> Cuatrimestre de 2021

Integrante	LU	Correo electrónico
Alem, Santiago	650/14	santialem.trev@gmail.com
Bustamante, Joaquín	670/17	joaquin.i.bustamante@gmail.com



**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

## 1. Introducción

Un toon shader es un tipo de shader que intenta simular la apariencia de un cómic o cartoon. Para resolver el toon shader tomamos varios enfoques. El primero de todos era el dummy, donde la luz era 0.0 o 1.0 dependiendo del valor del producto interno entre la normal de la superficie y la dirección de la luz. Con formula generalizada:

$$\frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{c_i < x\}}(\langle \text{luzDir}, \text{norm} \rangle),$$

siendo  $\{c_1, \dots, c_n : c_i \leq c_{i+1} \wedge c_i \in (0, 1) \forall i \in \{1, \dots, n\}\}$

Siguiendo el articulo Unity Toon Shader Tutorial [6] llegamos al modelo final de renderización, con el agregado de sombras proyectadas y un skybox.

## 2. Toon Shader

Para realizar los cortes más suaves usamos la función smoothstep. Ésta devuelve el resultado de la interpolación hermitiana, donde en el primer parámetro vale 0 y en el segundo 1 con polinomios cúbicos, evaluada en el tercer parámetro. Modificando las distancias entre los primeros dos parámetros alteramos el ancho del degradé. Si quisieramos que los bordes sean duros acercaríamos estos valores el uno del otro cerca del 0.5. Con una ventana del 0.04 logramos nuestro degradé.

La formula mencionada de al introducción la utilizamos usando en vez de la indicadora, una interpolación hermitiana que nos permite hacer los cortes más suaves entre las fases.

En el lugar de los  $c_i$  usamos  $\frac{i}{n}$ . Para atenuar la luz le aplicamos raíz cuadrada al producto interno de la dirección de la luz y la normal de la superficie.

El rim es luz que se agrega en los bordes iluminados para simular el reflejo de la luz o las luces detrás del objeto. Para conseguirla tenemos tres pasos:

1. Calculamos el complemento del producto interno entre la dirección de camara y la normal ( $1.0 - \text{dot}(\text{normal}, \text{point\_of\_view})$ ).
2. Con smoothstep hacemos el cambio de una luces sea abrupto.
3. Multiplicamos por  $\text{pow}(\text{dot}(\text{normal}, \text{point\_of\_view}), 0.1)$  para controlar las areas donde aplicamos el rim.

## 2.1. Contorno

Para lograr el contorno inicialmente utilizamos el producto interno entre la normal de la superficie y la dirección de la luz, si era menor que una variable del tamaño del trazo se pinta el color del trazo[2]. Luego decidimos extender el dibujado de contorno para que también trace el límite de las sombras por lo que para las distintas fases si se encuentra en el borde lo marcamos. Este método tiene como desventaja que las normales del modelo tienen que estar bien definidas para lograr un buen efecto.



Figura 1: Podemos ver los trazos en los bordes de las distintas partes del sombreado

Esto no era suficiente para las sombras proyectadas por lo que encontramos la extensión **OES\_standard\_derivatives** de webgl que permite realizar derivadas de una variable en un vecindario de fragmentos. Esto nos permite encontrar los bordes en los que las sombras proyectadas [5]. Este método si bien es sencillo de implementar gracias a las funciones de la extensión, al solo realizar las operaciones en un vecindario chico de fragmentos, no logra resultados suaves al pintar el contorno.

## 3. Sombras Proyectadas / Shadow Maps

Decidimos extender el shader para que soporte sombras proyectadas. Utilizamos proyección de texturas para lograr el efecto de sombras [1]. La idea detrás de esta técnica es realizar dos pasadas del render. En la primera pasada con una cámara que consideramos cómo el origen de la luz donde computamos solamente la profundidad que tiene cada vértice. En la segunda pasada es cuando renderizamos los objetos, iluminando los vértices más cercanos.

Para esto usamos la extensión de WebGL llamada **WEBGL\_depth\_texture**. En nuestro código observamos la profundidad desde la fuente de luz.

Simplificando un poco los pasos, realizamos lo siguiente al renderizar la escena:

1. Realizamos una primera pasada en el espacio  $MV_lO$  para generar el mapa de sombras utilizando el framebuffer. Dónde  $O$  es una matriz de proyección ortogonal, de esta manera podemos tener la sombra proyectada bajo una luz direccional como la que realizamos en nuestro shader. Y  $V_l$  es la matriz de transformación correspondiente al espacio de la cámara de la luz. Y  $M$  es la matriz de transformación al espacio mundo. La información de la distancia a cámara luz es guardada en una textura asociada al framebuffer.
2. En la segunda pasada además de llevar los vértices a un espacio  $MVP$ , también utilizamos una variable varying  $v_{projectedTexCoord}$  para tener la posición del vértice en el espacio  $MV_lO$ . De esta manera podemos obtener la distancia del vértice actual en ese espacio, la llamamos *currentDepth*.

3. Con estas  $v_{projectedTexcoord}$  podemos también obtener el valor de la distancia guardada en la textura del shadow map. La llamamos  $projectedDepth$
4. Luego evaluamos  $projectedDepth \leq currentDepth - bias + tan(\text{acos}(luzNormal))$  si esto se cumple sombreadmos el vértice, donde  $bias$  es una constante positiva, al sumar este valor a la profundidad actual se aminora el efecto de "shadow acne" [4].

### 3.1. Limitaciones

Algunas de las limitaciones que encontramos con este método para obtener sombras fueron

- El área que podes proyectar sombras esta limitada por la proyección y posición de la cámara de luz[Fig. 4].
- No se producen buenos resultados en objetos "Delgados", en un plano la sombra se dibuja de ambos lados del planos.
- Puede generar artefactos en ciertas situaciones. Cómo el *shadow acne*[Fig. 2] o problemas de aliasing[Fig.4] en la textura. El primero se puede aminorar utilizando una constante de  $bias$ [4].
- A veces al considerar el modelo entero se reproducen algunos de estos artefactos, por lo que se puede considerar realizar la primera pasada sin los vértices que enfrentan a la cámara ("Face culling") [Fig. 5]

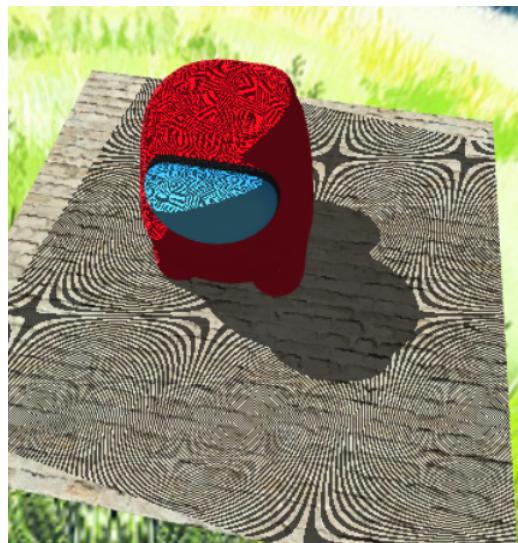


Figura 2: Ejemplo del artefacto de *shadow acne*.



Figura 3: Ejemplo de un resultado en las que no se pueden generar por la posición de la cámara que genera el mapa de sombras. Notar el paraguas.



Figura 4: Problemas de aliasing con el mapa de sombras. El resultado depende del tamaño del shadow map.



Figura 5: Diferencia entre realizar o no el "*Face culling*". Las mallas planas dejan de proyectar sombras pero algunos artefactos dejan de aparecer.

## 4. Sky Box

Para lograr el efecto del skybox, asignamos la distancia  $z = 1$  de manera que sea lo más distante de la cámara. Utilizamos una textura *cube map* a la que se puede asociar una imagen por cada lado del cubo de manera de crear el efecto de cubrir el espacio[8].

## 5. Conclusión

Este trabajo comenzó con el objetivo de realizar un toon shader, y fuimos decidiendo incorporar otras técnicas como la del mapa de sombras o lograr el contorno en la búsqueda de un mejor resultado final. Estos métodos no solo nos dieron una introducción a temas de rendering, como el de las texturas proyectadas para realizar las sombras, si no también a algunas extensiones de WebGL que facilitan algunos de los objetivos que buscábamos, que inicialmente no conocíamos.

## Referencias

- [1] Steve Marschner y Peter Shirley. *Fundamentals of Computer Graphics*. 4a edición. A K Peters/CRC Press, 1964, págs. 270-272. ISBN: 9781482229394.
- [2] Steve Marschner y Peter Shirley. *Fundamentals of Computer Graphics*. 4a edición. A K Peters/CRC Press, 1964, págs. 239-241. ISBN: 9781482229394.
- [3] *Repositorio del código en github*. URL: <https://github.com/santi-alem/final-fcg>.
- [4] *Shadow mapping*. URL: <https://www.opengl-tutorial.org/es/intermediate-tutorials/tutorial-16-shadow-mapping/>.
- [5] *Simplest and Fastest GLSL Edge Detection using Fwidth*. URL: <https://blog.ruofeidu.com/simplest-fatest-glsl-edge-detection-using-fwidth/>.
- [6] *Toon Shader*. URL: <https://roystan.net/articles/toon-shader.html>.
- [7] *Webgl Shadows*. URL: <https://webglfundamentals.org/webgl/lessons/webgl-shadows.html>.
- [8] *Webgl Skybox*. URL: <https://webglfundamentals.org/webgl/lessons/webgl-skybox.html>.
- [9] *Webgl Utils - Biblioteca de funciones auxiliares basada en twgl, lo utilizamos para facilitar ciertas partes del código*. URL: <https://webglfundamentals.org/docs/module-webgl-utils.html>.