Trabajo Práctico Final

Computación Grafica

"Aplicación: Toon Shader”

Alumnos: Lopez Lorenzo, Pozzer Andrés

Docente: Novara Pablo, Mainero Francisco

Fecha de entrega: 23/11/2022

- INTRODUCCION

El presente informe consiste en desarrollar la técnica Toon Shader, la misma es no fotorrealista, que busca simular los dibujos hechos a mano con bordes bien definidos de color negro y colores planos con un evidente cambio en la iluminación. Uno de los primeros juegos en aplicar esta técnica Jet set radio en el 2000 y en el 2003 el juego XII:

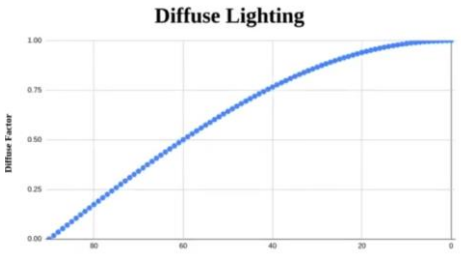
Se utilizó tanto para animación como la serie Futurama o la película el gigante de hierro:

Es una técnica que sigue vigente en la actualidad tanto para animación como en juegos, The Legend of Zelda: Breath of the Wild 2 que saldrá en 2023.

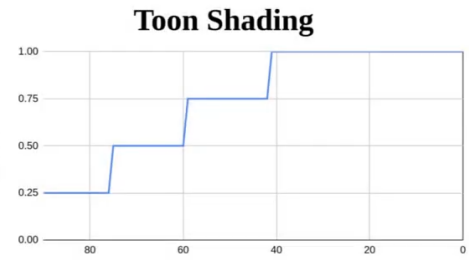
- DESARROLLO DE LA TECNICA

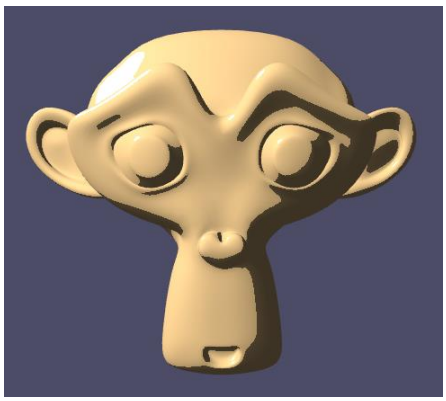
  El funcionamiento de la técnica en tiempo real consta de dos partes, el sombreado plano y la detección de bordes.

  - SOMBREADO PLANO
    El sombreado no debe ser continuo, sino que debe tener regiones bien definida das y separadas. Por lo tanto debemos analizar la componente difusa del modelo de pong, donde se realiza una curva suave:



Mientras que en el toon shader es una escalonada:





Esos escalones dependerán de los niveles que queremos conseguir, donde generalmente no se utilizan más de 3.
En la implementación utilizamos, shader donde:

1. Vertex: Necesitamos las posiciones de los vértices, la posición de la luz, y las matrices model, view y projection. Para poder calcular la dirección normal del fragmento y dirección de la luz al fragmento.
2. Fragment: aplicamos el producto punto la normal y la dirección de la luz, donde obtendremos 0 para el mínimo y 1 para el máximo, lo llamaremos intensidad. Así los valores de intensidad que se encuentren entre dos límites aplicamos un escalado, al color. Donde el color es la difusa multiplicada por la intensidad y sumada la ambiente.

- BORDES DE SILUETA

Definiendo las aristas, una escena está compuesta por triángulos que está conformado por arista, la silueta es la arista que queremos remarcar. Existen dos:

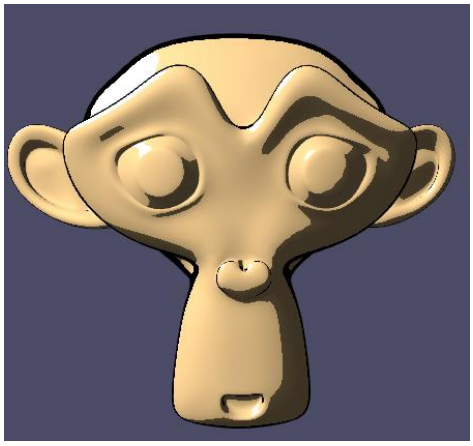
1. Contorno: conectan la cara trasera con la cara delantera del polígono.
2. Pliegue: depende del Angulo diedro de los polígonos adyacentes, generalmente son los bordes interiores.

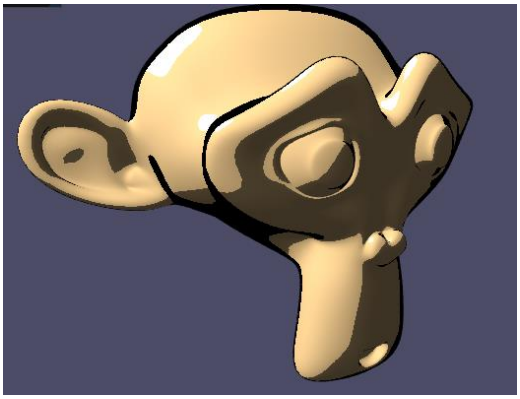
La técnica que utilizaremos es el normal map que es capaz de detectar ambas siluetas, la misma consta de obtener las direcciones normales por fragmento para luego aplicar un producto punto con la dirección de la vista. Así las que que son perpendiculares o el producto punto cero, son las aristas que queremos remarcar.

En la implementación usamos shader donde:

1. Vertex: Necesitamos las posiciones de los vértices, la posición de la luz, y las matrices model, view y projection. Para poder calcular la dirección normal del fragmento y dirección vista al fragmento.
2. Fragment: aplicamos el producto punto la normal y la dirección de la vista, donde obtendremos 0 para el mínimo y 1 para el máximo. Entonces todos los valores que sean cercanos a 0, serán las aristas y el color será negro.

- RESULTADOS:





- IMPLEMENTACION

La implementación es una modificación de los shader, del ejercicio introductorio de la catedra.

phong.vert

```
#version 330 core

in vec3 vertexPosition;
in vec3 vertexNormal;

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;

out vec3 fragPosition;
out vec3 fragNormal;
out vec4 lightVSPosition;

void main() {
    mat4 vm = viewMatrix * modelMatrix;
    vec4 vmp = vm * vec4(vertexPosition,1.f);
    fragPosition = vec3(vmp);
    gl_Position = projectionMatrix * vmp;
    fragNormal = mat3(transpose(inverse(vm))) * vertexNormal;
    lightVSPosition = viewMatrix * lightPosition;
}
```

phong.frag

```
# version 330 core

in vec3 fragNormal;
in vec3 fragPosition;
in vec4 lightVSPosition;

// propiedades del material
uniform vec3 ambientColor;
uniform vec3 specularColor;
uniform vec3 diffuseColor;
uniform vec3 emissionColor;
uniform float opacity;
uniform float shininess;

// propiedades de la luz
uniform float ambientStrength;
uniform vec3 lightColor;

out vec4 fragColor;
vec3 color;

void calcularColor( float intensidad){
    color=ambientColor;
    color+= diffuseColor * intensidad;
}
void main() {
    vec3 lightDir = normalize(vec3(lightVSPosition));
    float intensidad =  max(0.0, dot(normalize(fragNormal), lightDir));
    calcularColor(intensidad);
    vec3 aux=vec3(color);
    vec3 viewDir = normalize(-fragPosition);
    float silueta = max(0.0,dot(normalize(fragNormal),viewDir));
    if (silueta <= 0.2) {aux = vec3(0,0,0);}
    if (intensidad>=0.97 ){aux*=shininess;}
    if (intensidad>0.2 && intensidad<0.97){color =aux*0.7;}
    else {color=aux *0.3;}
    fragColor=vec4(color,1.f);
}
```

- REFERENCIAS

[1] Vivo Gonzalez Patricio. The book of shaders. https://thebookofshaders.com/?lan=es.

[2]Unity. Unity User Manual(2019.4 LTS), Normal Map. 2019.

[3] Wikipedia. Sombreado plano. https://es.wikipedia.org/wiki/Sombreado_plano_(animaci%C3%B3n).