

# **Trabajo de fin de grado**

## Funciones de distancia con signo

**Lukas Häring García**



**UNIVERSIDAD  
DE GRANADA**

# Índice General

## 1 Introducción

## 2 Lenguaje GLSL

- Vectores
- Matrices
- Operadores matemáticos

## 3 Spheremarcher

## 4 Modelo de iluminación

## 5 Funciones de distancia con signo (FDS)

- Primitivas sobre  $\mathbb{R}^2$
- Primitivas sobre  $\mathbb{R}^3$

## 6 Resolución de artefactos

## 7 Materiales

## 8 Conclusiones

# Introducción

El rápido incremento en potencia de la unidad de procesamiento gráfico (*GPU*) ha permitido utilizar técnicas de renderizado propuestas en los años noventa. Presentaremos el lenguaje *GLSL* del que hace uso la tecnología web y que es utilizado durante todo el desarrollo del proyecto.

Presentaremos las *funciones de distancia con signo*, una serie de funciones del que hace uso la técnica de trazado *spheremarching*, presentada por John C. Hart en 1996 y del que centraremos nuestro trabajo.

Haremos uso del modelo empírico de *iluminación de Phong*, presentado por Thuong Phong en 1975, que es indispensable para dar realismo y sensación tridimensional a una escena, junto a los materiales.

# Lenguaje GLSL

# Tipos

Mantiene una sintaxis similar a C, encontramos los siguientes tipos más importantes.

- **int**. Entero con signo.
- **float**. Número real, con precisión de 32 bits.
- **bool**. Ocupa un byte, *true* o *false*.
- **vecN**. Vector matemático,  $N$ -úpla de floats.  
Definidos: `vec2`, `vec3`, `vec4`.
- **matN**. Matriz cuadrada de dimension  $N$ .  
Encontramos: `mat2`, `mat3`, `mat4`.
- **matNxM**. Matriz de dimensiones  $N \times M$ .  
Encontramos: `mat2x2`, `mat2x3`, `mat2x4`, `mat3x2`, `mat3x3`, `mat3x4`, `mat4x2`, `mat4x3`, `mat4x4`.

# Vectores

El tipo vector, `vecN`, definido por una t-úpla:  $(x, y[, z[, w]])$  ó  $(r, g[, b[, a]])$ . Utilizaremos el operador «.» para acceder y copiar estas componentes.

## Constructores

- `vecN(float, ..., float)`
- `vecN(vecM, float)`
- `vecN(float, vecM)`
- `vecN(vecP, vecQ)`

## Funciones

- `length(vecN vector)`
- `distance(vecN p1, vecN p2)`
- `normalize(vecN vector)`
- `dot(vecN v1, vecN v2)`
- `cross(vecN v1, vecN v2)`

# Matrices

Las matrices  $matN \times M$  y  $matN$ , formadas por  $N \times M$  y  $N^2$  componentes flotantes, respectivamente. El operador de acceso a las componentes es similar al lenguaje C, del tal forma que:  $[j][i]$  accede a la celda de la fila  $j$ -ésima y columna  $i$ -ésima.

## Constructores

- $matN \times M(float, \dots, float)$
- $matN \times M(float, \dots, float)$
- $matN(vecN, \dots, vecN)$
- $matN \times M(vecM, \dots, vecM)$
- $matN(matM)$

## Funciones

- $transpose(mat\ matrix)$
- $matrix1 * matrix2$
- $determinant(matN\ matrix)$

# Operadores matemáticos

Agrupamos *float* y *vecN* con el nombre de *genType* para reunir los tipos de argumentos. Cuando utilizamos un operador sobre el tipo *vecN*, este se aplicará sobre cada una de sus componentes.

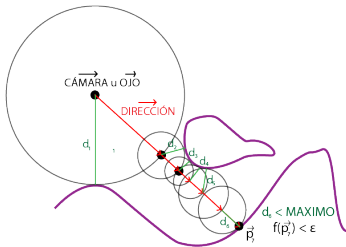
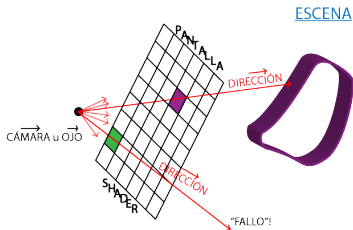
- `radians(genType var)`
- `sin(genType var)`
- `tan(genType var)`
- `asin(genType var)`
- `atan(genType var)`
- `pow(genType a, genType b)`
- `exp(genType var)`
- `sqrt(genType var)`
- `sqrt(genType var)`
- `abs(genType a)`
- `sign(genType a)`
- `min(genType a, genType b)`
- `max(genType a, genType b)`
- `mix(  
    genType a,  
    genType b,  
    (genType ó float ó bool) h  
)`



**Spheremarcher**

# Spheremarcher

Un *fragment shader* es aplicado a cada píxel de nuestra pantalla, que es procesado por una *hebra* de la *GPU*. «Lanzaremos un rayo», de manera numérica, para cada píxel y se aproxima la intersección, de manera iterativa, desde el ojo en la dirección del píxel dirección.



# Spheremarcher 2

Como se ha comentado anteriormente, hacemos uso de las *funciones de distancia con signo* las cuales codifican la escena,  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ .

Definimos la posición del «rayo» en la iteración  $n$ -ésima, como:

$$\text{rayo}_n = \vec{ojo} + \text{dirección} \cdot d_n$$

donde  $d_n$  es la distancia total recorrida por todas las iteraciones:

$$d_n = d_{n-1} + f(\vec{p}_{n-1}) \text{ con } d_0 = 0$$

## Definition

Sea  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ , una función de distancia con signo, definimos como *isoperímetro*,  $L = \{\vec{p} | f(\vec{p}) = 0\}$ .

## Definition

Sea  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ , una función de distancia con signo, definimos como *isosuperficie*,  $S = \{\vec{p} | f(\vec{p}) = 0\}$ .

# Condiciones de parada

Al tratarse de un método numérico, vamos a definir las condiciones de parada:

- 1 Primera condición.** Utilizaremos una variable de control,  $\epsilon$  que relajará la restricción de la definición de *isosuperficie*, haciendo  $f(\vec{p}_n) < \epsilon$ , ya que, si  $\epsilon = 0$ , trataríamos de un modelo analítico.
- 2 Segunda condición.** Superar una cierta distancia recorrida,  $d_n \geq MAXIMO$ , creando una esfera de trazado sobre el punto de la cámara.
- 3 Tercera condición.** Superar el número de iteraciones máximas,  $n \geq PASOS$ . *PASOS* es una constante fijada.

Este algoritmo devolverá  $d_n$ , cuando el algoritmo finaliza debido a la **segunda o tercera condición**, devolverá,  $d_n = MAXIMO$ , recibiendo el nombre de «*fallo*». Un fallo, representando un pixel vacío, sin superficie trazada, pudiéndose considerar el fondo de la escena.

# **Modelo de iluminación**

# Normal de una isosuperficie

Para un modelo de iluminación es indispensable el cálculo de la normal de una isosuperficie, por ello, vamos a presentar el siguiente teorema:

## Theorem

*El vector gradiente  $\nabla f(x_0, y_0, z_0)$  es perpendicular a la curva de la tangente de una isosuperficie en el punto  $\vec{p} = (x_0, y_0, z_0)$ .*

En realidad, nos quiere decir que la normal de una *isosuperficie* es proporcional a su gradiente o exacta en caso de su posterior normalización:

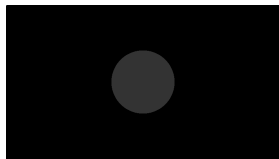
$$\vec{n} = \text{norm}(\nabla f(x, y, z)) \approx \text{norm} \left( \left\langle \begin{array}{c} \frac{f(x + 0.001, y, z) - f(x, y, z)}{0.001} \\ \frac{f(x, y + 0.001, z) - f(x, y, z)}{0.001} \\ \frac{f(x, y, z + 0.001) - f(x, y, z)}{0.001} \end{array} \right\rangle \right)$$

# Intensidad lumínica

Para cada luz  $\vec{l}_i \in L$ , definimos el vector director de la luz hasta el punto  $\vec{p}$  como  $\vec{d}_i = \text{norm}(\vec{l}_i - \vec{p})$ , la intensidad es un factor multiplicativo.

## Intensidad ambiental

Intensidad mínima sobre la isosuperficie.



$$I_a \in [0, 1]$$

## Intensidad difusa

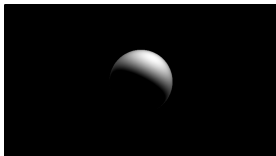
Intensidad por la luz refractada por la superficie.



$$I_d = \sum_{\vec{l}_i \in L} \vec{n} \cdot \vec{d}_i$$

## Intensidad especular

Intensidad por la incidencia en el ojo de la luz reflectada.



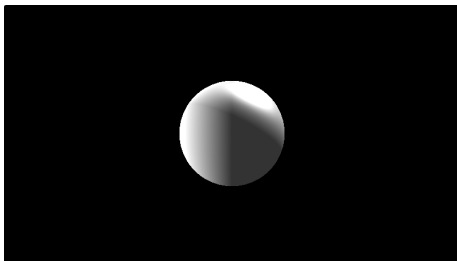
$$I_e = \sum_{\vec{l}_i \in L} \vec{o} \cdot \vec{d}_i \cdot (\vec{d}_i \cdot \vec{n})$$

# Modelo de Iluminación de Phong

Presentado por Thuong Phong en 1975 como un modelo empírico, resultado de las sumas de las intensidades anteriores, además, utiliza un *homeomorfismo* como factor de brillo para la intensidad especular:

$$h_k : [0, 1] \longrightarrow [0, 1], h_k(x) = x^{2^k}$$

$$I_{Phong} = I_a + \underbrace{\sum_{\vec{l}_i \in L} \vec{n} \cdot_{[0,1]} (\vec{l}_i - \vec{p})}_{\text{Intensidad Difusa}} + \underbrace{h_k \left( \vec{ojo} \cdot_{[0,1]} \left( (\vec{l}_i - \vec{p}) \vee \vec{n} \right) \right)}_{\text{Intensidad Especular}}$$



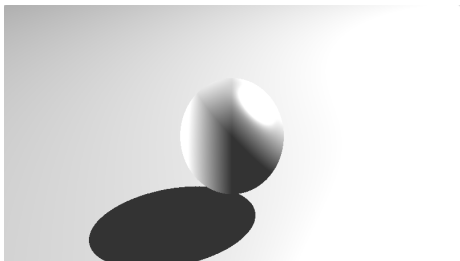


# Umbral

Dado un punto  $\vec{p}$  sobre la superficie, lanzaremos otro rayo hacia la luz para ver si este es ocluido, en caso de trazar otro punto  $\vec{q}$  en esa dirección, la intensidad se mantendrá constante.

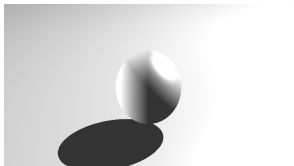
Al lanzar el rayo desde la una isosuperficie, las primeras iteraciones resultan de bolas pequeñas, por ello, separaremos el punto  $\vec{p}$  de la superficie haciendo uso de la normal de la superficie y un factor de empuje  $k \in \mathbb{R}_0^+$ .

$$\vec{p}' = \vec{p} + \vec{n} \cdot k$$



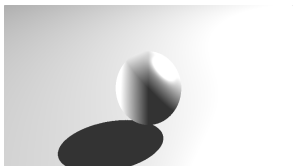
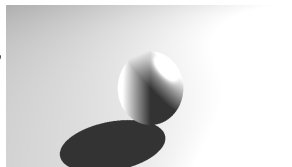
# Funciones de distancia con signo (FDS)

# Primitivas sobre $\mathbb{R}^2$



```
float SDFCircunsferencia(vec2 p, float r){  
    return length(p) - r;  
}
```

```
float SDFRectangulo(vec2 p, vec2 s){  
    vec2 a = abs(p) - s;  
    return length(max(a, 0.0)) + min(max(a.x,  
    }  
}
```



```
float SDFCircunsferencia(vec2 p, float r){  
    return length(p) - r;  
}
```

# Primitivas sobre $\mathbb{R}^3$

# Resolución de artefactos

# Materiales

# Conclusiones