Julio Martín Saez y Raquel Peces Muñoz.

El viaje alucinante

Práctica 1 – RENDERING AVANZADO

Contenido

[Introducción 2](#_Toc413587868)

[Generación de la curva 3](#_Toc413587869)

[Generación de la spline 3](#_Toc413587870)

[Cálculos sobre la spline 3](#_Toc413587871)

[Generación de la vena 4](#_Toc413587872)

[Generación de los glóbulos 5](#_Toc413587873)

[Cámara 6](#_Toc413587874)

[Material utilizado 7](#_Toc413587875)

[Problemas encontrados y características no implementadas 8](#_Toc413587876)

El viaje alucinante

# Introducción

# Generación de la curva

## Generación de la spline

Para la generación de la curva nos hemos basado en el documento facilitado en el que se explica la interpolación por Catmull-Rom, en el que el conjunto original de puntos también es parte de los puntos de control de la curva spline que se genera.

Por un lado la clase “BezierCurve” se encarga de a partir de unos puntos de control especificados, los cuales pueden variar al aplicarle un random a esos números de suma o resta entre unos límites, para evitar que la vena se cruce entre sí, generar la curva a través de la interpolación Catmull-Rom.

Por otro lado, para poder ver la línea que sigue la curva, con los puntos generados, los hemos pintado. La clase “DrawCurve” es la encargada de coger dichos puntos y mandárselos al shader de vértices y fragmentos correspondientes, en este caso “curveVshader” y “curveFshader” los cuales toman dicha curva en forma de puntos y la pintan en modo línea continua, discontinua o puntos, dependiendo del modo en el que nos encontremos.

## Cálculos sobre la spline

Para que la cámara pueda seguir correctamente la curva, necesitamos situarla correctamente en ella. Para ello necesitamos el vector tangente a la curva en ese punto, y el vector binormal en ese punto. Por otro lado, para el cálculo de la estructura de la vena a partir del marco de Frenet también necesitamos la normal en dicho punto.

Es por esto que para cada punto de la curva calculamos estos valores. Una de las decisiones que tuvimos que tomar era generar una curva en el plano XY, para que la normal fuese el vector Z, porque en otro caso la generación de la vena nos hacía pliegues a la hora de construir las caras que la forman.

Por tanto, el cálculo de estos vectores para cada punto queda del siguiente modo:

Estos valores se guardarán en arrays para utilizarlos tanto en la Cámara como en la generación de la curva.

# Generación de la vena

## Geometría de la vena

Para el cálculo de la geometría de la vena,en primer lugar hacemos uso de la clase “Poligon”.Esta clase es la encargada de generar un polígono regular de N lados centrado en el origen, el cual lo utilizamos para aproximar la circunferencia.

Una vez calculado el polígono, se obtiene la transformación del sistema de coordenadas global en un sistema de coordenadas local cuyo origen se encuentra en los puntos de la curva. Para ello haremos uso del Marco de Frenet, en el cual se parte de un perfil dado, en nuestro caso el polígono mencionado, y se producen sucesivos perfiles, cuyos puntos se unen, igual que se hace en una malla por extrusión.

El cálculo de la transformación por el Marco de Frenet es la matriz:

Donde N, B y T son los vectores normal, binormal y tangente respectivamente correspondientes a las coordenadas C en el punto t.

Por otro lado, para el cálculo de los vectores normales por cada cara, hacemos uso del Método de Newell, en el cual por cada cara se calcula su vector normal a partir de las coordenadas de los puntos que lo forman.

## Ruido de Perlin

La deformación de la vena se realiza a través de un mapa de Perlin. Este cálculo lo realizamos en la CPU después de tener calculada la geometría. Y se aplica una deformación de los vértices hacia el centro de la vena.

Para el cálculo del mapa de ruido de Perlin se usa la clase “PerlinGenerator” el cual tiene dos métodos relevantes “generate” y “generateNextLevel”, los cuales se encargan de generar la imagen final del ruido de Perlin y de generar la imagen de un nivel específico respectivamente.

Para ello el método “generateNextLevel” recibe el número de cuadrados que forman dicho nivel y obtiene los valores aleatoreos que corresponderán a las aristas exteriores, en este caso, están asignados para que pueda pegarse de manera circular, y después se itera por los puntos interiores de cada cuadrado. Por último se interpolan los valores de manera bilineal a lo largo de cada cuadrado.

Para depurar la generación de los mapas y comprobar su correcto funcionamiento, se ha utilizado una clase *BitmapSaver* que se encarga de escribir una imagen. Al ser una funcionalidad totalmente irrelevante y con carácter únicamente depurativo, se ha obtenido el código de internet y se ha adaptado para los fines específicos.

A continuación se muestran las imágenes para una ejecución concreta:



Figura 2 Nivel 1

Figura Nivel 0



Figura Nivel 2



Figura Nivel 3

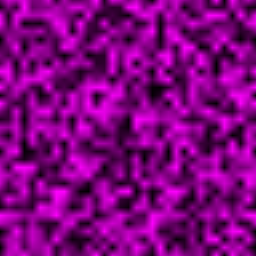


Figura Nivel 4



Figura Nivel 5

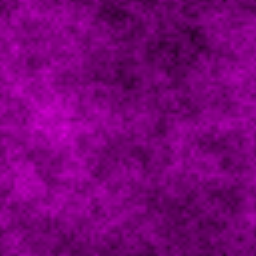


Figura Combinación de los niveles

# Generación de los glóbulos

## Cálculo de la geometría

Para la generación de los glóbulos hemos hecho uso de la clase “Blood” la cual contiene un array de “BloodElement”. Esta clase recibe el número de glóbulos rojos y blancos que queremos y procede a crearlos.

Para ello, ambas funciones realizan un algoritmo similar, eligen aleatoriamente un punto de la curva, y a él le suman un valor acotado, para que la posición no se salga de la vena. Una vez calculada la posición del glóbulo se calcula su rotación, para ello también obtenemos un número aleatorio para cada componente de la rotación con respecto a cada uno de los ejes. Con estos valores, realizamos el cálculo de los glóbulos.

La geometría de los glóbulos se ha obtenido a través de la geometría de un toroide. Esta función la hemos obtenido de internet, para poder obtener las coordenadas de los vértices, así como sus normales y sus coordenadas de textura, puesto que usando funciones como “glutSolidTorus” no podríamos obtener a ellas. El cálculo de esta geometría se realiza en el origen y luego se transforma a través de las matrices de rotación y translación a las coordenadas que hemos generado aleatoriamente.

Los “BloodElements” pueden ser de dos tipos, rojos y blancos. Los glóbulos rojos se pasan los parámetros específicos para generar una toroide de tipo “horn torus” para que tenga una forma más similar al glóbulo rojo. Por otro lado los glóbulos blancos están formados de dos toroides uno rodeando a otro.

Una vez calculada toda la geometría se pasan a sus shaders correspondientes, en este caso el “bloodVShader” y “bloodFShader” los cuales se encargan de realizar el pintado de los glóbulos.

Estos shaders son muy similares a los de la vena, pero en este caso se pasa un color como uniform encargado de pintar dicho color en todos los vértices de la geometría.

## Mutación

Para realizar la parte de la mutación de los glóbulos, pasamos una variable “mutation” la cual realizará un coloreado similar al efecto plasma que aprendimos con los shaders de frágmentos en la asignatura de Procesadores Gráficos.

# Cámara

# Material utilizado

# Problemas encontrados y características no implementadas