Capítulo 3

Integración de partículas a un sistema de cirugía de próstata

En este capítulo veremos la implementación de las partículas al simulador de cirugía del CCADET el cual fue citado en el Capítulo 1. Para la simulación de partículas fue necesario implementar los siguientes módulos: las burbujas, ya que la cirugía se realiza en un ambiente acuoso; la simulación de sangrado, para las partes en la que es necesario cortar alguna sección del tejido; y por último, la simulación del tejido desprendido.

En este trabajo no se aplicaron las formulas de Navier - Stokes, las cuales describen con mejor detalle el movimiento de un fluido, debido a los altos requerimientos computacionales del método utilizado para resolver numéricamente las ecuaciones diferenciales. Por lo tanto se opto por buscar una alternativa, la cual es explicada a continuación.

3.1. Movimiento de una partícula en un fluido

Para esta parte se dividió el movimiento de la partícula en tres, uno para cada eje, teniendo diferentes ecuaciones en cada uno de estos.

Para el movimiento en el eje Y, podemos apreciar un ascenso o un descenso. En este caso observamos en la Figura 3.1, las fuerzas que actúan sobre la esfera. Podemos apreciar que hay dos fuerzas ascendentes mientras que hay una descendente, las dos que apuntan hacia arriba son el empuje de Arquímedes y la fuerza de fricción para un objeto esférico o ley de Stokes vistos en el Capítulo 2, mientras que la fuerza restante es el peso de la partícula.

De aquí, la fuerza para la componente Y, es igual a la suma del empuje más la fuerza de fricción viscosa o fórmula de Stokes, menos el peso de la esfera, como se muestra en la Ecuación 3.1.

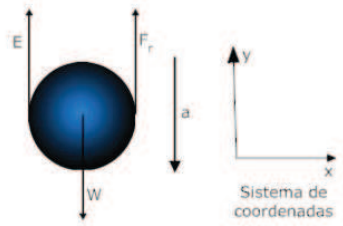

Figura 3.1: *Movimiento vertical de una esfera en un fluido*

$$\Sigma F_y = E + F_f - W \tag{3.1}$$

La formula de Stokes para la fricción debida a la viscosidad F_f, vista en el Capítulo 2, está dada por:

$$F_f = 6\pi r \mu V_y \tag{3.2}$$

Donde r es el radio de la partícula (en caso de una esfera), μ es la viscosidad del medio en el cual se desenvuelve la partícula y V_y es la velocidad de la partícula en la dirección Y.

El empuje E está dado por:

$$E = \frac{4}{3}\pi \rho_f r^3 g \tag{3.3}$$

Donde ρ_f es la densidad del medio (fluido) y g es la constante de gravedad ($g = 9.81\ m/s^2$).

El peso W está dado por:

$$W = mg \tag{3.4}$$

Para una esfera se puede expresar como:

$$W = \frac{4}{3}\pi \rho_e r^3 g \tag{3.5}$$

Donde ρ_e es la densidad de la esfera.

Sustituyendo tenemos la Ecuación 3.8:

$$\Sigma F_y = E + F_f - W \tag{3.6}$$

$$\Sigma F_y = \frac{4}{3}\pi\rho_f r^3 g + 6\pi r\mu V_y - \frac{4}{3}\pi\rho_e r^3 g \tag{3.7}$$

$$\Sigma F_y = \frac{4}{3}\pi r^3 g\left(\rho_f - \rho_e\right) + 6\pi r\mu V_y \tag{3.8}$$

Si tomamos a $k = 6\pi r\mu$ la Ecuación 3.8 queda como:

$$\Sigma F_y = \frac{4}{3}\pi r^3 g\left(\rho_f - \rho_e\right) + kV_y \tag{3.9}$$

Si hacemos un análisis dimensional de la ec. 3.9 obtenemos:

$$\Sigma F_y = L^3\frac{L}{T^2}\frac{M}{L^3} + L\frac{M}{L\cdot T}\frac{L}{T} \tag{3.10}$$

$$\Sigma F_y = \frac{L\cdot M}{T^2} + \frac{L\cdot M}{T^2} \tag{3.11}$$

$$\Sigma F_y = M\frac{L}{T^2} \tag{3.12}$$

Con lo que podemos observar que las unidades en ambos lados de la ec. 3.9 son consistentes:

$$\Sigma F_y = ma \tag{3.13}$$

Ahora, si en la Ecuación 3.9 representamos F como la suma del empuje y el peso, la ecuación queda como:

$$\Sigma F_y = F + kV_y \tag{3.14}$$

La aceleración de la partícula estará dada por:

$$a = \frac{\Sigma F_y}{m} \tag{3.15}$$

La velocidad de la partícula estará dada por:

$$\frac{dV}{dt} = \frac{\Sigma F_y}{m} \tag{3.16}$$

$$V(t) = \int a(t)dt \tag{3.17}$$

El desplazamiento sobre el eje Y de la partícula, estará dada por:

$$Y(t) = \int V(t)dt \tag{3.18}$$

Para el movimiento en el eje X, se considera una fuerza oscilante, con el objetivo de hacer que las partículas tengan un desplazamiento en dicha dirección, de tal manera que el movimiento cambie en cada instante de tiempo, resultando en una mejor representación visual. Para ello se multiplica una fuerza constante f_c, longitudinal por una función coseno, como se observa en la Ecuación 3.19.

$$\Sigma F_x = f_c \cos(\pi f_a) - F_f \tag{3.19}$$

El factor $\cos(\pi f_a)$ provoca un desplazamiento aleatorio en un instante de tiempo. La función f_a está distribuida uniformemente sobre un rango de valores entre 1 y 5.

Para el movimiento en el eje Z, consideramos la ecuación de Bernoulli en la cual expresa el movimiento de un flujo a lo largo de un tubo, como se ve en la Figura 3.2.

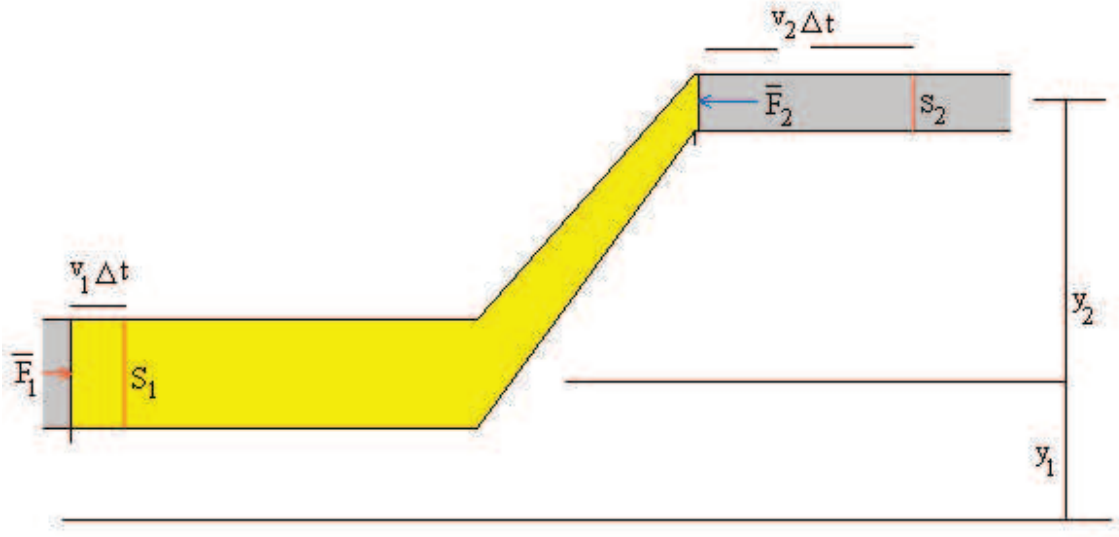

Figura 3.2: *Flujo a lo largo de un tubo [30]*

La Ecuación de Bernoulli se expresa como:

$$P_1 + \rho g y_1 + \frac{\rho V_1^2}{2} = P_2 + \rho g y_2 + \frac{\rho V_2^2}{2} \tag{3.20}$$

O también expresada de forma general como:

$$P + \rho g y + \frac{\rho V^2}{2} = C \tag{3.21}$$

Donde y es la altura del tubo, P es la presión y C es una constante.

Ahora consideremos que el tubo está a una altura constante y que es cero, y además despreciamos la presión como se muestra en la Figura 3.3.

Figura 3.3: *Caso Particular de la ecuación de Bernoulli*

Obtenemos una ecuación más sencilla:

$$F_q = \frac{\rho V^2}{2} = cte \tag{3.22}$$

La cual es una fuerza y es constante a lo largo de todo el trayecto. Por lo tanto la suma de fuerzas en Z se puede expresar como:

$$\Sigma F_z = F_q - F_f \tag{3.23}$$

Si sustituimos la ecuación se expresa como:

$$\Sigma F_z = \frac{\rho V_z^2}{2} - 6\pi r \mu V_z \tag{3.24}$$

3.2. Integrador de Euler

Ya que hemos obtenido las ecuaciones de movimiento de las partículas, ahora es necesario ver cómo obtener a partir de las fuerzas ejercidas en cada una de ellas, la aceleración, velocidad y posición.

Para esto se aplicó el integrador de Euler, que es un método numérico de primer orden para resolver ecuaciones diferenciales ordinarias. El cual por ser fácil de implementar y rápido, es una buena opción.

El método se basa de forma general, en extrapolar la pendiente estimada actual a una anterior, obteniendo el nuevo valor, es decir [31]:

$$valor = valorAnterior + pendiente * paso \tag{3.25}$$

O bien,

$$y_{i+1} = y_i + mh \tag{3.26}$$

Donde m es la pendiente y h es el paso.

Esto se aplica de manera recursiva, paso a paso, para obtener el siguiente valor y trazar la trayectoria de la solución, como se muestra en la Figura 3.4.

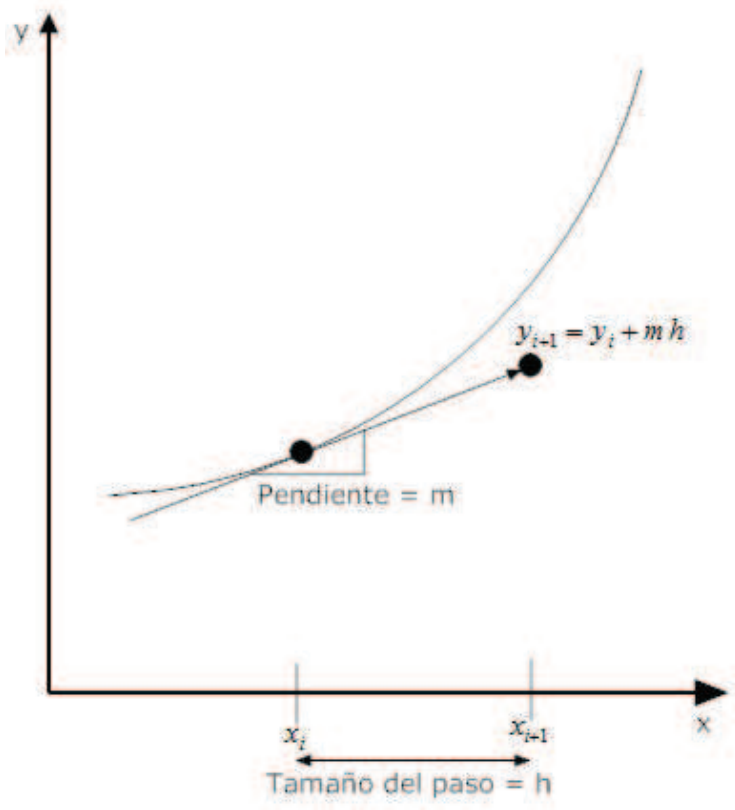

Figura 3.4: *Método de Euler*

Se puede apreciar que entre más pequeño es el paso más se acerca a la solución exacta de la ecuación diferencial.

El método de Euler utiliza la pendiente al inicio del intervalo como una aproximación de la pendiente promedio sobre todo el intervalo. La primera derivada proporciona una aproximación de la pendiente en x_i [31].

$$m = f(x_i, y_i) \tag{3.27}$$

Donde $f(x, y)$ es la ecuación diferencial evaluada en x_i y y_i. Si sustituimos en la Ecuación 3.26, obtenemos:

$$y_{i+1} = y_i + f(x_i, y_i)h \tag{3.28}$$

Esta ecuación se conoce como el método de Euler, en el cual se predice un nuevo valor de y a partir de la pendiente, que es la primera derivada en el valor x_i.

Con esto, podemos desarrollar un algoritmo que resuelva las ecuaciones planteadas para el movimiento de una partícula. El cual quedaría de la siguiente manera:

Inicializa velocidad, posición
Desde i = 0 hasta particulasMax
    Calcula fuerzas
    Obtiene el incremento o paso deltat
    *Velocidad = velocidad + (fuerza / masa) *deltat*
    *Posición = posición + velocidad * deltat*
Fin del programa

El algoritmo anterior se aplica para cada una de las direcciones, con su correspondiente ecuación de movimiento.

3.3. Visualización de las partículas

Ya que tenemos cómo es que se van a comportar las partículas y el método para poder obtener la posición de cada una de ellas, es necesario comenzar a ver como dibujarlas en pantalla.

La técnica usada en esta tesis es llamada *Billboarding* que es un procedimiento de orientación de objetos (como polígonos), de tal manera que siempre estén viendo hacia el observador [35].

Un *billboard* es un objeto simple poligonal texturizado que siempre mira hacia el observador en el ambiente virtual. Algunos tipos de *billboards* solo giran horizontalmente. Si el punto de vista del observador se coloca por encima, éste no girará. A éstos se les conoce como *billboard-z* [34]. Las razones por las cuales se usan *billboards* son muy sencillas: para optimizar la memoria y hacer que el proceso de dibujado sea más eficiente. A este objeto típicamente se le aplica una imagen bmp (*bitmap*), del fenómeno que se quiera simular (como puede ser una nube de humo). Entonces en lugar de tener una malla que represente el humo, se estarían dibujando polígonos usando una imagen que representa el fenómeno.

Esta técnica se utiliza para visualizar muchos fenómenos además de humo, tales como fuego, niebla, explosiones, campos de energía, rastros de vapor, nubes, y otros

fenómenos y partículas.

Además se puede tener una deformación del polígono, sin embargo, cuando el campo de vista y los objetos son pequeños, los efectos de deformación pueden ser ignorados y ser alineados a una orientación simple a la vista del plano usada. Otro caso puede ser cuando es necesario cambiar la normal deseada, que es igual al vector que va del centro del billboard a la posición del observador. Como se muestra en la Figura 3.5.

Un caso de esta técnica puede ser observado en la Figura 3.6 en la cual se generan nubes con planos texturizados, que giran con respecto a la posición del observador, y da la impresión de ser una representación en tres dimensiones.

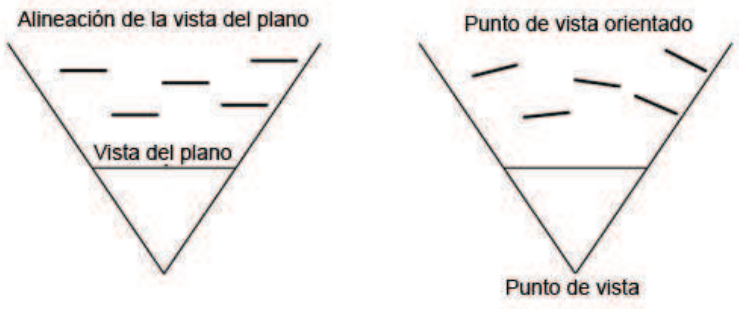

Figura 3.5: *Vista superior de dos técnicas de alineamiento de billboard [32].*

Figura 3.6: *Nubes generadas con planos [32].*

En nuestro caso utilizamos la misma técnica, de *billboard*, sin embargo, ya que la posición del observador no rota, o el ángulo de rotación es muy pequeño, se usó la técnica simple vista en la Figura 3.5 de alineación de la vista del plano.

Ya con la técnica de visualización elegida, y con las funciones de *OpenGL* para crear transparencias se puede crear el mismo efecto.

3.4. Adaptación de las partículas al simulador RTUP

Una vez que hemos definido los aspectos a considerar, podemos comenzar con la implementación de las diferentes partículas al simulador.

Ya que los objetos que se encuentran en el escenario están dentro de un flujo, podemos generalizar el método para todas las partículas, simplemente variando las características de cada una de ellas.

El simulador de cirugía de próstata RTUP del CCADET, cuenta ya con la implementación de varios módulos que ayudan a realizar diferentes trabajos. Un diagrama de la estructura de ejecicion del simulador puede apreciarse en la Figura 3.7. En el cual se observa que cuenta con un proceso principal en el que se ejecutan todos los cálculos para el despliegue gráfico, y otro hilo donde se reciben datos de una interfaz mecatrónica o de un dispositivo háptico[1].

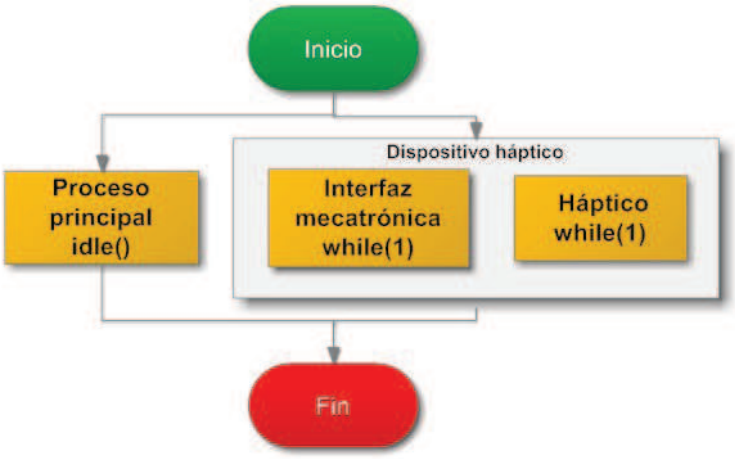

Figura 3.7: *Estructura general del bloque de ejecución del programa del simulador [33].*

Ahora si descomponemos el proceso principal, éste queda como se muestra en la Figura 3.8

Como vemos ya se integra el módulo de partículas, el cual depende del ciclo principal de *OpenGL* para calculos y para el despliegue.

[1]Interface de comunicación con una computadora, que tiene dispositivos envolventes que censan el movimiento del cuerpo [36].

3.4.1. Estructura de las partículas

Como hemos observado, las partículas son objetos que tienen propiedades que las caracterizan, dependiendo del tipo de material que estemos considerando para su estudio. Por lo tanto, es necesario primero definir una estructura que contenga todas las características necesarias para poder trabajar con ellas. Para nuestro caso es considerada como se muestra en la Figura 3.9, donde definimos diferentes características de la partícula, que son:

- Masa: cantidad de materia de cada una de las partículas para realizar los cálculos dependiendo del objeto.
- Fuerza: fuerza total que actua en las diferentes direcciones, en este caso la fuerza es un vector de tres componentes.
- Posición inicial: posición en la cual las partículas inician su ciclo de vida.
- Velocidad: la velocidad que lleva la partícula en un instante, a partir de la cual podemos obtener otros parámetros.
- Posición: la posición donde se encuentra en un instante de tiempo.
- Tamaño: radio de la esfera.
- Tiempo de vida: la cantidad de tiempo que va a mostrarse en pantalla.
- Tiempo trascurrido: el tiempo que ha pasado desde su nacimiento.

De tal forma que tendremos una estructura que como la que se muestra a continuación:

```
1  struct Particula{
2          float      masa;
3          float      tiempoVida;
4          float      tiempoTrascurrido;
5          float      radio;
6          VTPoint3D  posicion;
7          VTPoint3D  velocidad;
8          VTPoint3D  fuerza;
9          VTPoint3D  posicionInical;
10 };
```

Donde VTPoint3D pertenece a un tipo de datos definido en el simulador para el manejo de vectores.

Como podemos observar, se almacena información importante de cada una de las partículas, que es actualizada en cada instante, como es: la fuerza, velocidad y posición, adicionalmente también se puede modificar la posición donde se comienzan a desplegar, mientras que el tiempo de vida de cada una de las partículas es aleatorio.

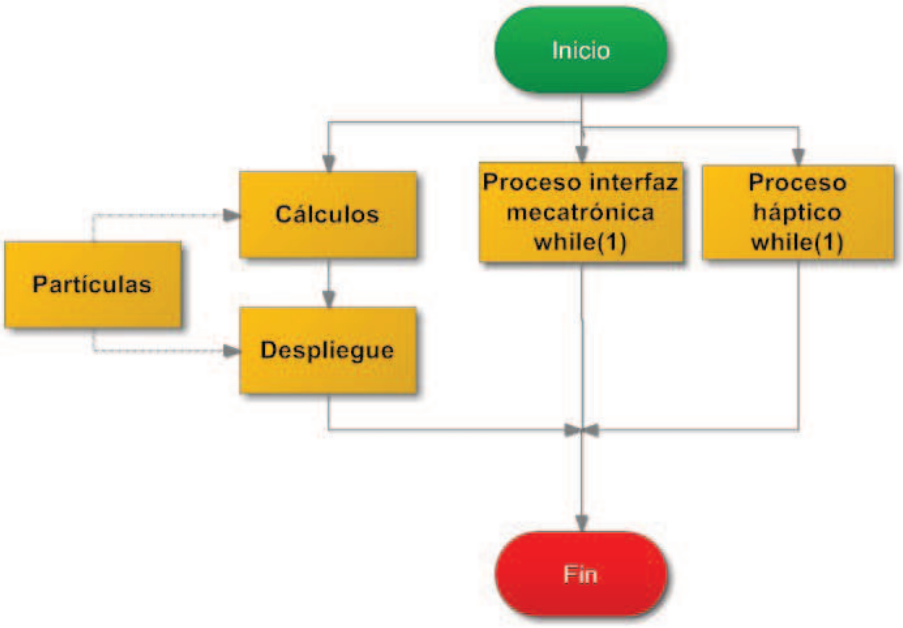

Figura 3.8: *Estructura general del bloque de ejecución del programa del simulador con el módulo de partículas.*

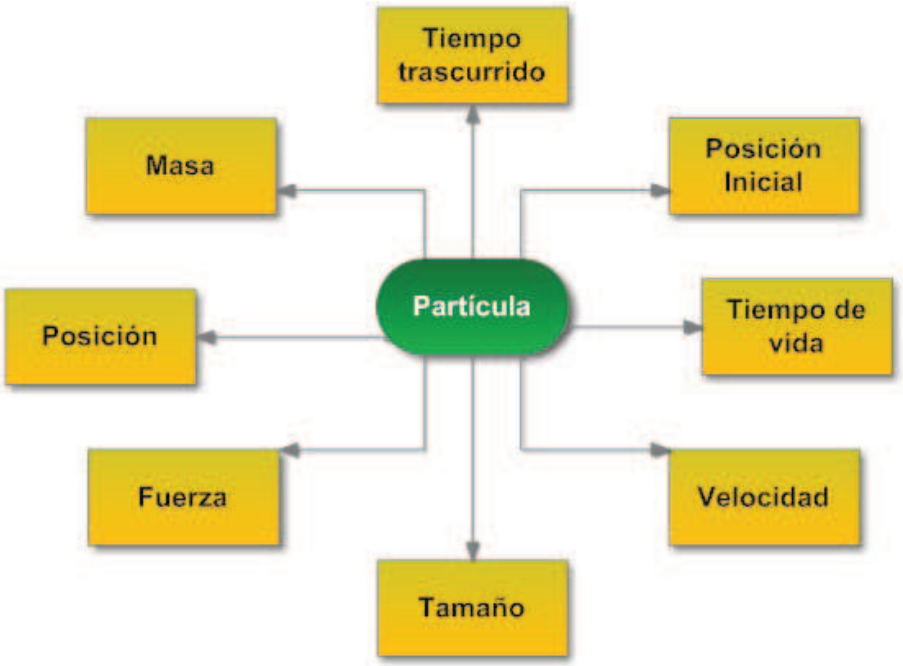

Figura 3.9: *Estructura de una partícula.*

3.4.2. Estructura del programa

El programa consta de diferentes partes, las cuales son: la creación e inicialización de las partículas, la carga y asignación de texturas, el procesamiento de los datos, y el despliegue. Esta estructura general del programa para partículas se muestra en la Figura 3.10.

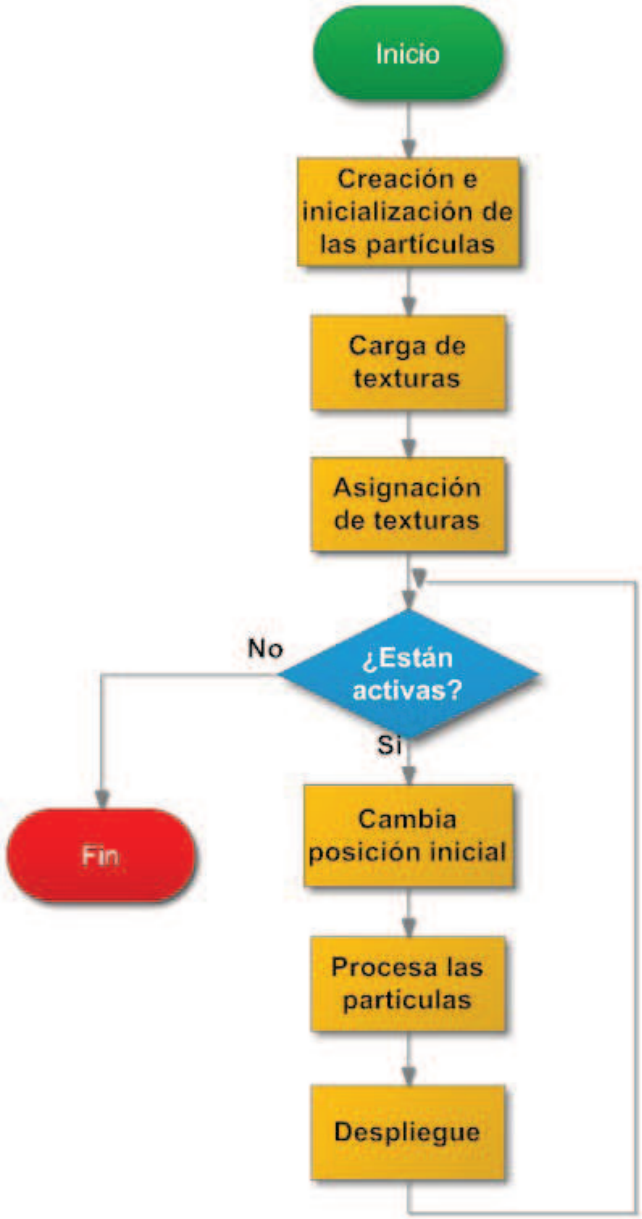

Figura 3.10: *Estructura general del programa para partículas.*

Primero se deben asignar los valores iniciales a cada una de las partículas, los

cuales ya fueron listados en la sección "estructura de las partículas"; estos valores se asignan directamente con el uso de la siguiente función:

```
1 initParticle(VTPoint3D position, int type)
```

En la cual se le pasan como argumentos un vector con la posición en la que se desea que comience el dibujado, y el tipo de partícula que se inicializa. Para el caso de los tipos de partículas que se pueden asignar son los siguientes:

- BUBBLE
- BLOOD
- TISSUE

Lo que hace esta función es simplemente asignar los valores de cada uno de los elementos de la estructura, y agrega una fuerza inicial con la que son expulsadas, esta fuerza es aleatoria para darle volumen al despliegue.

A continuación se hace la carga de texturas, la cual se lleva a cabo con la instrucción:

```
1 textureLoadTGAchar( char *nameDirectory, char *nameTex)
```

Que carga imágenes de tipo *.tga* que contienen un canal alfa para las transparencias, y regresa un tipo de dato *GLuint*. Recibe como parámetros el nombre del directorio donde se encuentra la textura y el nombre del archivo respectivamente.

Ya que se tienen las texturas en variables, se asignan para después ser dibujadas en la escena con la instrucción:

```
1         particleSetTexture( GLuint texture, int type )
```

Esta función recibe como argumentos la textura y el tipo al que se le va a asignar, y asigna las texturas a una variable global que es un arreglo del tipo *GLuint*.

Después, ya que se han realizado los ajustes previos, verifica el estado de las partículas, si no están activas, la ejecución del programa termina, pero si se encuentran activas, se hace un cambio de la posición inicial, que es de donde surgirán de nuevo en cada ciclo; esto se hace, debido a que en el escenario se dibujan en diferentes posiciones, dependiendo del usuario. La modificación de la posición inicial se realiza con la instrucción:

```
1         particleChangeInitPosition( VTPoint3D position, int type )
```

Que recibe la nueva posición y el tipo de partícula que va a ser modificado. Y lo que hace es asignar el nuevo valor al que tenía guardado anteriormente, para cada uno de los elementos de un tipo.

Y por último, antes del despliegue, realiza los cálculos; sin embargo, para esta parte que es importante, se define una secuencia de pasos que debe hacer para poder obtener los resultados, que se verán reflejados en pantalla. Este proceso se muestra en la Figura 3.11.

Esta parte es la más importante del programa ya que aquí se realizan los cálculos necesarios para que a partir de las ecuaciones se obtenga la posición que después es mostrada en pantalla.

Para esta parte se manda llamar a la siguiente función:

```
1       processParticle( int type )
```

La cual realiza el trabajo, mencionado anteriormente, y recibe solo el tipo de partículas que van a ser procesadas.

Este proceso se realiza para todo el arreglo de partículas de un tipo específico, es decir tenemos un conjunto de elementos que van a ser procesados uno a uno dentro de un ciclo.

Entonces lo que se hace primero es declarar un conjunto de elementos de la estructura *Particula*. Como se muestra a continuación:

```
1       Particula particula[ MAXPARTICLES ];
```

Donde estamos definiendo el número de elementos que se van a procesar y a mostrar, además los cálculos que se realizan son para cada una de las partículas, de tal forma que entre más objetos tengamos más recursos y tiempo necesitaran.

El código de esta sección se muestra a continuación:

```
1  for( int i=0; i < MAXPARTICLES; i++ ){
2              /*****************************************/
3              /*                Integramos             */
4              /*****************************************/
5              particula [i] = eulerIntegration(particula [i], BUBBLE );
6
7              /*****************************************/
8              /*          Fuerzas de fricción          */
9              /*****************************************/
10             Ffx = res1 * particula[i].velocity.x;
11             Ffy = res1 * particula[i].velocity.y;
12             Ffz = res1 * particula[i].velocity.z;
13             /*****************************************/
14             /*                Empuje                 */
15             /*****************************************/
16             E  = res2;
17            /*****************************************/
18            /*             Fuerza en X               */
19            /*****************************************/
20            Fex = Fc * cos( M_PI * randRange( 1, 5) );
```

```
21              /*****************************************/
22              /*              Suma de fuerzas          */
23              /*****************************************/
24              Ftot.x = Fex - Ffx;
25              Ftot.y = E - FG - Ffy;
26              Ftot.z = -(Fq - Ffz);
27              /*****************************************/
28              /*           Agregamos fuerza            */
29              /*****************************************/
30              particula[i] = addForce( particula[i], Ftot );
31              /*****************************************/
32              /*        Reiniciamos las particulas     */
33              /*****************************************/
34              if( processLifeTime( particula[i].currentTime ) >
35                                   particula[i].lifeTime ){
36                  particula[i] = reset( particula[i], BUBBLE );
37              }
38  }
```

En este fragmento de código, vemos que ejecuta los cálculos para cada uno de los elementos del arreglo de partículas, en la línea número 5, comienza llamando a la función eulerIntegration, la cual se le envía como parámetro el elemento a integrar y el tipo de objeto del que se trata, y ésta regresa una partícula con la nueva velocidad y posición. El algoritmo de esta función fue explicado en la sección 3.2.

A continuación en las líneas 10, 11 y 12 se calculan las fuerzas de fricción para cada uno de los ejes coordenados, donde res1 es el resultado de la Ecuación 3.2 que es multiplicado por la velocidad de la partícula en un instante de tiempo.

En la línea 16, E es el resultado de la Ecuación 3.3, mientras que en la línea 20 tenemos el resultado de la Ecuación 3.19.

Después en las líneas 24 a 26 se hace la suma de las fuerzas en cada uno de los ejes, en la línea 30 se suman las fuerzas anteriores a las actuales, y se asignan a la partícula.

Y por último en la línea 34 se comprueba el tiempo transcurrido con el que tiene asignado la partícula, ya que si este es mayor, se resetean los valores de la partícula a los originales para volver a comenzar el ciclo.

Este fragmento de código es la idea general del proceso, sin embargo las características de cada conjunto de objetos son diferentes.

Una vez que hemos terminado todos los procesos, es tiempo de realizar el despliegue con una función de dibujado, sin embargo, cada uno de los tipos de elementos tiene una forma diferente de dibujado. Estas se explican a continuación.

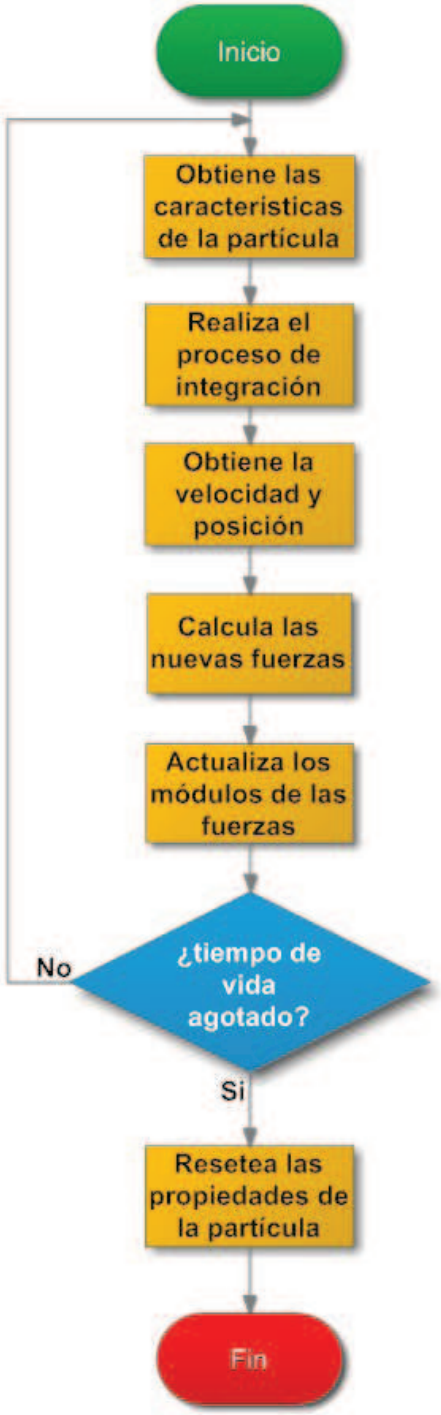

Figura 3.11: *Diagrama del submódulo de procesamiento de partículas.*

3.5. Simulación de burbujas

Para el despliegue de las burbujas, como ya se mencionó antes, se utilizó la técnica de *billboards*. Se aplicó una textura a un plano generado con la directiva de *OpenGl* GL_QUADS, con canal alfa para las regiones transparentes. La textura utilizada para estos objetos es la que se muestra en la Figura 3.12, donde las regiones negras es la que se va a quitar de la escena, mostrándose lo que este en la parte de atrás.

Figura 3.12: *Textura de las burbujas.*

El plano es escalado al radio definido por medio de la instrucción *glScalef*, y después es trasladado en cada instante de tiempo por medio de la función *glTranslatef*, de tal forma que tiene un movimiento continuo, como se muestra en la Figura 3.13.

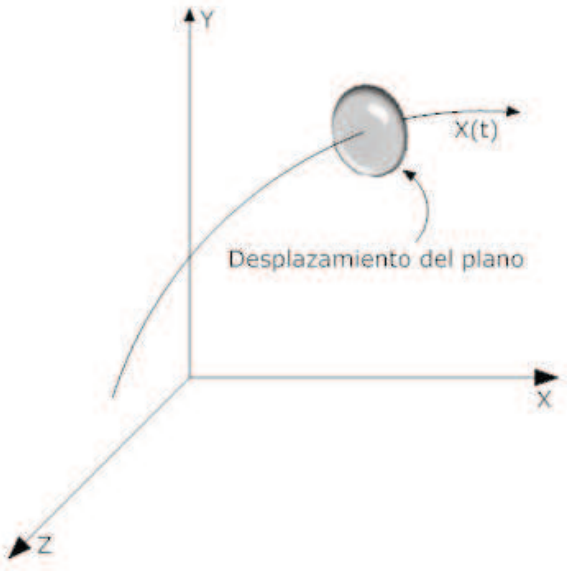

Figura 3.13: *Movimiento del plano texturizado.*

3.6. Simulación de sangre

A diferencia de las burbujas, para esta parte, no se usó un plano texturizado, sino que se utilizó una esfera texturizada con la ayuda de la instrucción *gluSphere*, de tal modo que tenga volumen. La esfera es desplegada con pocos segmentos y anillos para que no se reduzca la velocidad de procesamiento, y la textura que se utilizó se muestra en la Figura 3.14.

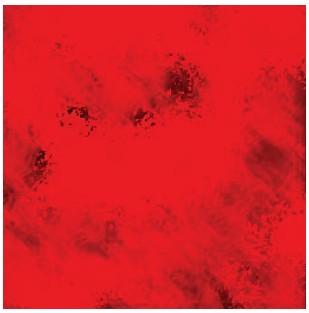

Figura 3.14: *Movimiento del plano texturizado.*

Al igual que en las burbujas, se hace un escalamiento de la esfera y una traslación para crear el movimiento.

3.7. Simulación de tejido

En esta parte fue necesario cargar un modelo de un tejido para hacer la simulación del desprendimiento de este, y que además este modelo es texturizado con una imagen que fue generada con un programa desarrollado por uno de los miembros del Laboratorio de Imágenes y Visualización del CCADET. El modelo texturizado del tejido se muestra en la Figura 3.15.

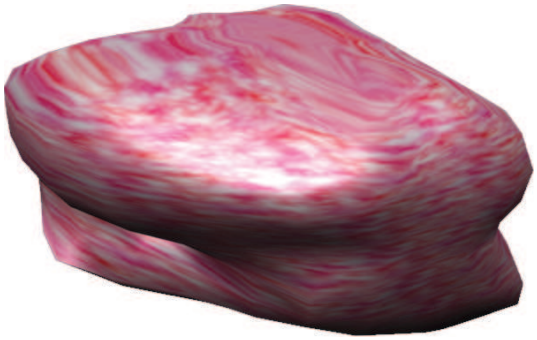

Figura 3.15: *Modelo del tejido desprendido.*

Para hacer la simulación del tejido desprendido se siguen los mismos pasos de los elementos anteriores, con la diferencia que ahora el escalamiento del tejido es proporcional al corte que realice el usuario, es decir, cuando se detecta una colisión del resectoscopio y el tejido, y además se encuentra en modo corte, se guarda la posición y se va escalando hasta que deja de haber contacto entre los dos objetos.