Problema 1 - DCCRayo de fotones

El primer problema que DCCerebro plantea es simular esta casa de espejos y que dada una configuración inicial de fotones con cierta velocidad y dirección se pueda observar y registrar todas las colisiones con espejos. Pinky ya escribió una solución. Sin embargo, esta es sumamente ineficiente, por lo que DCCerebro te contactó. En la modelación se encuentra lo siguiente:

- Particle: representa un fotón. Posee

```
Particle {
   id
   position
   velocity
}
```

- Segment: representa un espejo. La cual es una recta entre un punto y otro.

```
Segment {
   id
   startPosition
   endPosition
}
```

Tras tener los estados iniciales de *photons* y *segments* se comienza la simulación. Esta es modelada mediante Frames[1]. Al inicio se revisan todas las colisiones entre un fotón y las paredes y, de acuerdo a esto, se calcula y actualiza la nueva dirección de los fotones involucrados. Luego, se actualiza la posición de cada fotón según su velocidad, lo que modificará las colisiones en el siguiente frame.

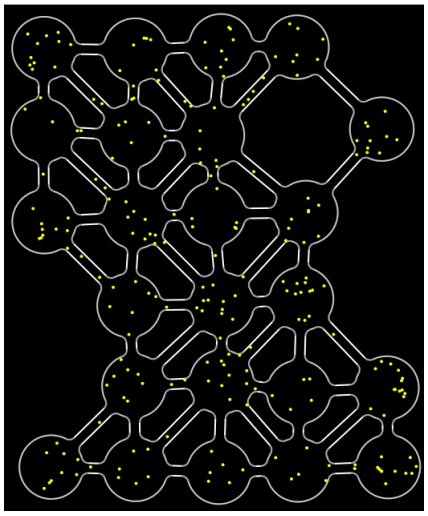

Figura 1: Visualización de la simulación

[1]Son una representación del tiempo que indica cuanto durará la ejecución (en función de cuadros, no segundos), para más información acá, la revisión de colisiones se realiza por cada frame

Solución

En el código base entregado se encuentra una solución directa al problema. Esta consiste en el enfoque directo de revisar que en cada frame, revisar cada fotón y por cada fotón revisar si existe o no colisión con alguna pared. En caso de empates (más de un choque en el mismo frame) la simulación debe seleccionar al segmento con menor id. El pseudocódigo del algoritmo descrito es el siguiente

Algorithm 1 Pseudocódigo del algoritmo directo

```
for frame in F frames do
    for particle in P particles do
        set collision element X to none
        for segments in S segments do
            if particle collides with segment then
                update X to S
            end if
        end for
        Update direction of particle according to X
        Update position of particle
    end for
end for
```

Fácilmente se puede observar que la complejidad de este problema es $\mathcal{O}(FPS)$, con F cantidad de frames, P el número de partículas y S el número de segmentos.

Para esta tarea deberás organizar los segmentos en una estructura de datos que cumpla que su complejidad sea $\mathcal{O}(FP * log(S))$, para poder ejecutar el algoritmo frente a inputs con gran cantidad de segmentos.

Para resolver este problema se recomienda utilizar el algoritmo **B**ounding **V**olume **H**ierarchy (o **BVH** para los amigos). Este algoritmo es esencial en el mundo de los videojuegos ya que permite seccionar el espacio de tal forma que la deteccion de colisiones sea mucho mas sencilla. **BVH** es un tipo de arbol para objetos geometricos, donde cada uno de los objetos se encuentra contenido por una caja. Esta estructura tiene la caracteristica de que la caja que envuelve un nodo x siempre contiene a las cajas de los hijos de x.

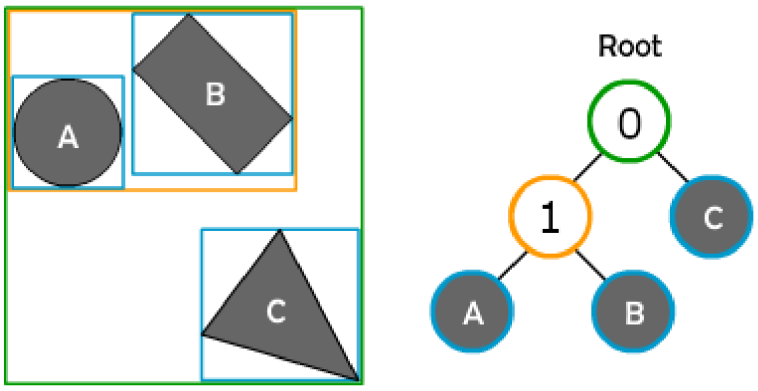

Figura 2: Arbol BVH

En la *Figura 2.* Se observa un ejemplo de **BVH**[2][3] donde la caja del nodo 1 envuelve completamente a los elementos A y B. y a su vez, todos los objetos estan envueltos por una caja $raiz$.

Para la solucion de este problema se recomienda que implementes un **KD-Tree**[4] de segmentos en dos dimensiones. Sin embargo, si deseas utilizar otra estructura que solucione el problema, eres totalmente libre de hacerlo siempre y cuando no supere la complejidad objetivo[5]

[2] Muy importante para computer graphics Ejemplo
[3] Un muy buen articulo al respecto Link
[4] Wikipedia
[5] $\mathcal{O}(FPlog(S)$