Proyecto Simulador de Sonar en 2D

Granados Acuña Steven
Instituto Tecnológico de Costa Rica
2019037999
Cartago, Costa Rica
svengra01@gmail.com

Reveiz Rojas Marco Antonio
Instituto Tecnológico de Costa Rica
2019053583
Cartago, Costa Rica
revvace@gmail.com

***Abstract*—Este documento explica el funcionamiento de un simulador de sonar de eco-locación en dimensión 2D programado en el lenguaje Python. En este proyecto, se implementaron conceptos, de formulas físicas para representar el comportamiento de las ondas de sonido**

I. Introducción

En este documento se explica el funcionamiento de un programa en Python que se encarga de simular un sonar de eco-locación en dimensión 2D. El programa se centra en la implementación del algoritmo de Path Tracing, complementándolo con el algoritmo probabilístico Monte Carlo para la creación de imágenes virtuales de un entorno lleno de ”paredes”. En el caso del simulador, se debe implementar un sonar que dispara rayos sonoros que se reflejan y se encargan de brindarle al sonar la información suficiente para poder hacerse una imagen de lo que lo rodea, estos cálculos se hacen de acuerdo a varios factores como: la distancia, la energía del rayo, el ángulo de rebote, etc. El objetivo principal es lograr proyectar la imagen más fiel posible gastando los mínimos recursos en el proceso, es por ello que se aplica Monte Carlo.

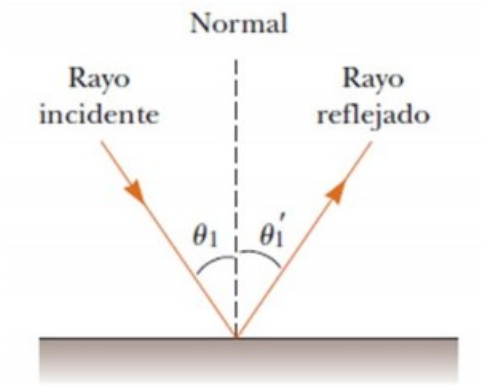
II. Trabajo Relacionado

A. Intensidad del Sonar

El sonido tiene las propiedades del movimiento en ondas, presenta reflexión, refracción, difracción y las mismas relaciones entre frecuencia, longitud de onda y velocidad e interferencia de propagación. La sonoridad del sonido está fuertemente relacionada con la intensidad del sonido. El proyecto debe simular rayos de viajen en varias direcciones de acuerdo a donde apunte el sonar, esto con el objetivo de representar las ondas en las que se propaga el sonido. Para hacer un óptimo simulador de eco-locación se debe calcular la intensidad de los rayos al chocar con las paredes, para que cuando vuelvan al emisor del sonar este obtenga la información relacionada con la distancia a la que se encuentra esa pared. La forma de calcular la intensidad en el proyecto quedaba al criterio del programador. En este simulador, utilizamos una que se basa en una escala del cero al uno según su distancia siendo cero el valor más bajo y uno el máximo, luego por medio de una regla de tres se transforma a escala de 0 a 255, esto ya que, se debe representar esa intensidad como un color en la escala de grises, entonces en este caso 0 es negro y 255 blanco, entre más cerca de 255 esté el número, más ”nítida” será la imagen.

B. Ángulo de Reflexión

El proyecto se basa en rayos emitidos a través de un sonar que buscan encontrar una pared de una escena previamente definida. Cada rayo al llegar a una pared debe simular el efecto de reflexión, para que esto ocurra el rayo debe rebotar y cambiar de dirección. El problema de emitir el rayo reflejado era determinar el ángulo de reflexión. Para solucionar este problema, se tomo en cuenta que ambos ángulos (rayo incidente y reflejado) son iguales de acuerdo a la normal del segmento en el que se refleja el rayo, por lo que al sumarle 180 grados al ángulo incidente, se obtendrá el ángulo reflejado pero en dirección contraria. La imagen a continuación ilustra la solución el concepto anteriormente mencionado.



C. Ray Casting

En este proyecto se debe implementar la técnica del Path Tracing algoritmo principal para que los rayos ilustren la escena. Para esto se utilizaron varias clases que usaban un par ordenado para determinar ciertas operaciones necesarias como la intersección entre el rayo y la pared. Las coordenadas se basan en las de un plano cartesiano con un eje X y un eje Y. Esto tiene como fin utilizar las formulas de intersección entre rectas siendo estas los rayos en las paredes de la escena.

D. Geometría

En un plano cartesiano dos rectas puede tiene como intersección un punto determinado. La solución a encontrar la intersección se da por medio de un sistema de ecuaciones lineales. Para el calculo del punto de intersección

de dos rectas no paralelas se utiliza la regla de Cramer:

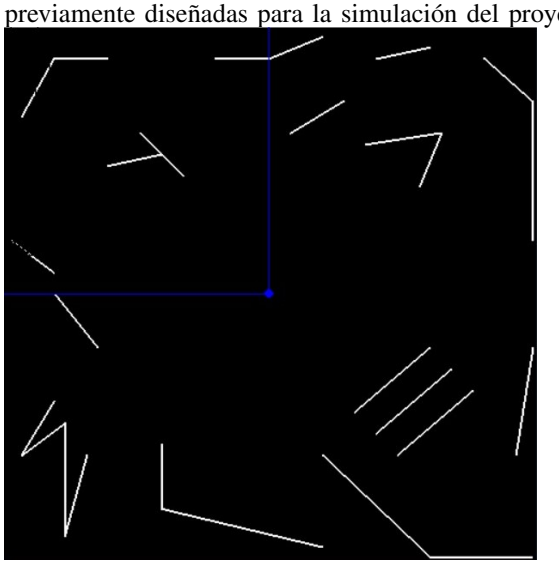
$$x_s = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1}, \quad y_s = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}.$$

III. Métodos Implementados

A. Estructuras

El programa posee tres clases principales el rayo, el sonar y las paredes.

- Particle: La clase llamada "Particle", es el sonar de la eco-locación. Este apunta hacia algún cuadrante según la elección del usuario. Su función consiste en definir la posición en la cual se disparan los rayos y hacia donde vuelve esa información,
- Ray: La clase "Ray", es el rayo de sonido que es disparado desde el sonar. Esta clase se podría considerar como la más importante del proyecto. En esta, se definen los puntos de los píxeles de partida y de llegada según el ángulo del rayo. Se verifica si el rayo colisiona con alguna pared de la escena, por lo que en esta clase se implementan algoritmos geométricos como calculo de intersecciones entre segmentos y calculo de distancia entre puntos con el fin de obtener datos como la distancia y a partir de ahí la intensidad y el lugar en el espacio donde se deben pintar los píxeles.
- Boundary: La clase "Boundary" define las paredes de la escena. Es una estructura bastante básica debido a que solo se necesitan sus coordenadas para determinar si colisiona o no con un rayo, además siempre se mantienen estáticas en la escena por lo que no tienen mayor complejidad. En la siguiente imagen se pueden ver las paredes previamente diseñadas para la simulación del proyecto.



B. Path Tracing

El Path Tracing es un algoritmo que envía rayos por un medio, y cuando los rayos impactan se reflejan o refractan en una superficie, y repite este proceso recursivamente hasta que la energía del rayo sea cero o el programador lo decida. En este proyecto, se debe implementar este algoritmo con múltiples rayos ligeramente modificados debido a que el algoritmo se basa en rayos de luz, en este caso lo que se quiere simular son ondas de sonido para una fiel representación del sonar.

C. Rayos Secundarios

Los rayos secundarios nacen a partir de la idea de que el rayo que dispara el sonar es un rayo de sonido y no de luz, entonces para representar de una manera más realista las ondas de sonido lo que se hace es generar rayos secundarios que acompañan al principal simulando este efecto. En el código los rayos secundarios y principales son en esencia lo mismo, se identifican por medio de una bandera que indica si un rayo es principal o secundario, y otro aspecto que cambia es que los rayos secundarios tendrán menos energía que el principal y conforme más "abierto" se el ángulo en el que salen disparados menor será la cantidad de energía inicial que estos tendrán.

D. Píxeles

Durante el planteamiento del plan para realizar el proyecto se presentó como un problema el guardar las coordenadas de los píxeles a colorear junto con su intensidad ya calculada. Esto debido a que, al utilizar algoritmos que requieran una gran cantidad de recursiones, también se obtiene un nivel de complejidad mayor para el programa. Se decidió implementar una variable global que se actualiza con cada recursión, agregando las nuevas coordenadas de cada píxel junto con su intensidad. Cada vez que se guarde un píxel se toma en cuenta la cantidad de rebotes que posee el rayo y se verifica si es secundario o primario para saber cual debe ser la locación donde se debe pintar (los rayos secundarios no pintan donde chocan, si no donde choca su rayo principal correspondiente, esto se explica más adelante). La implementación de la lista de listas como variable global facilitó el programa porque de esta manera se logro utilizar la variable en todas las funciones auxiliares, disminuyendo un poco la complejidad que se presenta a la hora de retornar variables en funciones recursivas.

En el proyecto se deben calcular de manera distinta los píxeles de los rayos primarios con respecto a los secundarios. Cada rayo primario tiene rayos secundarios. Los rayos primarios deben pintar el píxel en la coordenada en la cual chocan, por el contrario, sus rayos secundarios al chocar no pintan los píxeles en donde estos chocan, si no que pintan los píxeles cercanos al de su rayo principal, al ser un simulador de un sonar en 2D tiene que pintar los píxeles de tal manera que se simulen rayos de sonido, no de luz. El calcular las coordenadas correctas de cada rayos secundario se convirtió en un problema porque al principio no se sabía cómo se obtendría una coordenada que cumpliera las características requeridas. Como solución a este problema se investigó sobre una función que retorna la lista de puntos (con coordenadas enteras) que forman un segmento. Paso siguiente, dibujar una linea que empezara desde el punto de impacto del rayo principal hasta el final de la pantalla, de esta manera se tienen los puntos

que están detrás del pixel de impacto del rayo principal y por medio de una función, se obtiene el pixel posea una distancia similar a la que recorrió el rayo secundario. Al obtener este pixel ya el problema está solucionado, solo queda pintarlo en la pantalla según la intensidad del rayo secundario correspondiente.

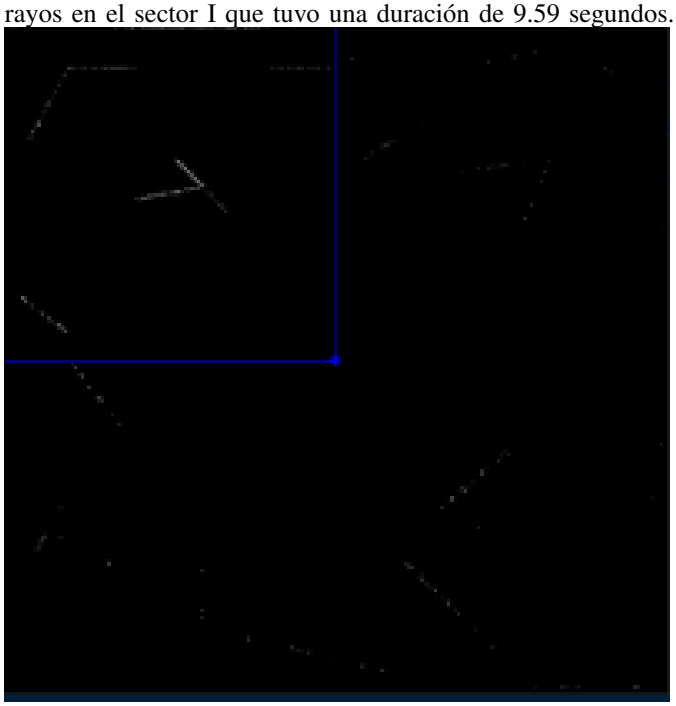
IV. Análisis Resultados

A. Diferente Cantidad de Rayos

En el proyecto se implementó un sonar que rotara libremente según el deseo del usuarios para que los rayos primarios y secundarios salgan disparados en una dirección en particular. En la simulación del proyecto programado se implementaron sectores para cuando el usuario deseara rotar la dirección de los rayos. La simulación del sonar se procesa de acuerdo al número de rayos primarios iniciales que sea previamente colocado en los parámetros. Al incorporar métodos de Monte Carlo, se debe tener en cuenta una gran cantidad de recursos. Durante las pruebas se tomó como punto fundamental la cantidad de rayos, debido a que conforme esta aumenta, los algoritmos recursivos de los rayos crecen mucho más y consumen gran cantidad de recursos. Se determinó que el número indicado para que la simulación se dé con un buen ritmo sin dificultades en la interfaz gráfica es de nueve. A partir de parámetros mayores a nueve la simulación puede presentar congelamientos, por la gran cantidad de recursos que se necesita para procesar dicha cantidad de recursiones, incluso se puede llegar al caso extremo de acabar la memoria de la pila y que el programa termine su ejecución de manera forzosa.

Como prueba se realizaron simulaciones con distintas cantidades de rayos primarios, todas las simulaciones se realizaron llamando a la función que dispara el sonar 20 veces en un ciclo. Se determinó que el límite de rayos primarios de la simulación es de 17, a partir de ahí la simulación no completa su proceso. Esto debido a que, por cada rebote de los rayos principales se generan N rayos secundarios, según lo defina el usuario, y además M rayos principales nuevos, siendo M un número elegido por el usuario, consumiendo una gran cantidad de recursos. La ejecución con 17 rayos principales tuvo una duración de 186.9133 segundos en el sector I. La segunda prueba se realizo con 15 rayos principales en el sector I. Esta tuvo una duración considerablemente más rápida que la anterior de 91.2081 segundos. Se puede ver con solo agregar dos rayos principales extra puede durar hasta dos veces más. Conforme se iba disminuyendo la cantidad de rayos principales el tiempo fue bajando considerablemente, por ejemplo el tiempo de una simulación en el sector II con 14 rayos fue de 67.3548 segundos mientras que con con 15 rayos en el mismo sector fue de 100.0157 segundos. También se descubrió que al probar 12 rayos empieza a disminuir el tiempo de ejecución y la simulación mejora debido a que se ve más natural para el usuario. El tiempo de ejecución con 12 rayos en el sector I fue de 25.8712 segundos.

La siguiente imagen ejemplifica una simulación con nueve rayos en el sector I que tuvo una duración de 9.59 segundos.

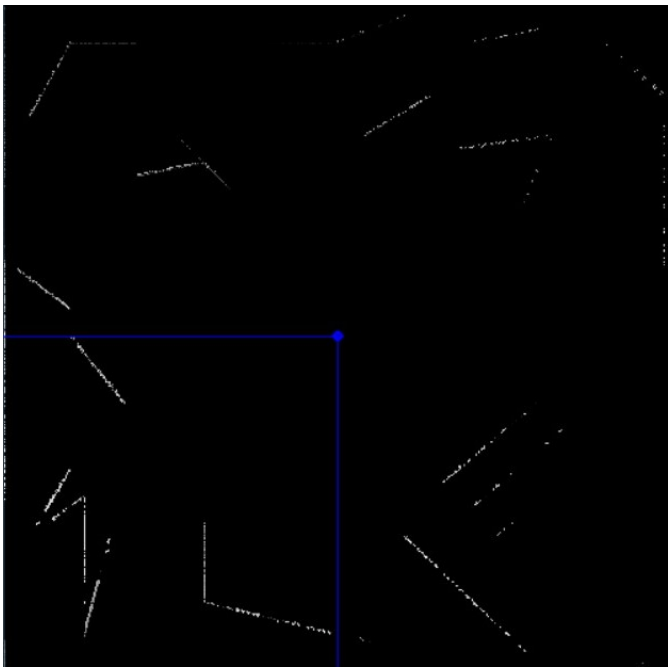


B. Tiempos de Ejecución por Sector

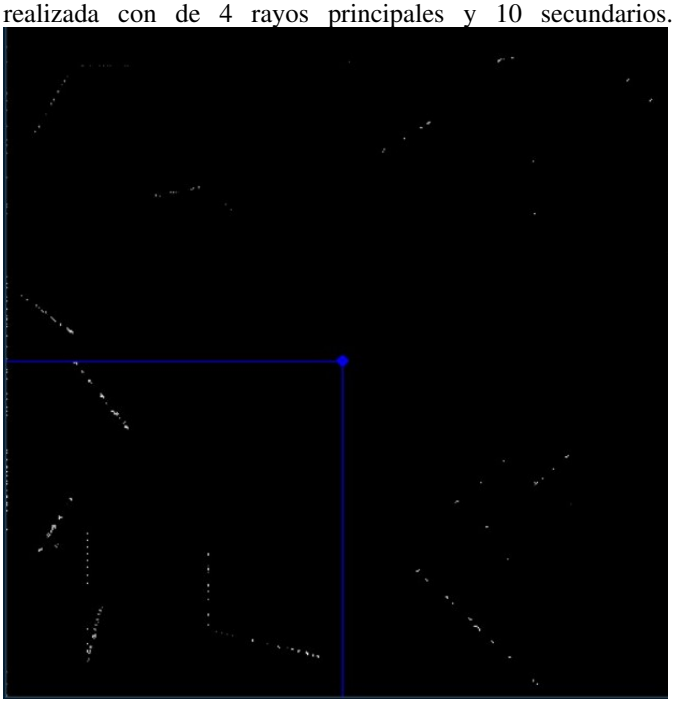
Una prueba que se realizó en la simulación conforme las paredes en la escena, fue comparar con ocho rayos, cuánto variaba el tiempo de ejecución con respecto al sector en que se realizaba. En el primer sector se dio la simulación más rápida, y por un gran margen con una duración de 7.2416 segundos. En el tercer sector se dio la simulación más lenta con 9.1520 segundos. La razón por la que hay una gran diferencia entre ambos es porque las paredes del tercer sector se encuentran con dimensiones que provocan que lo rayos reboten de maneras más extremas comparados con las del primer sector que posee paredes más regulares. En los otros sectores tuvo tiempos de ejecución muy parecidos con 8.7164 segundos en el segundo sector y 8.6396 segundos en el cuarto sector.

C. Rayos Primarios vs Rayos Secundarios

La última prueba es ver qué tanto aportan los rayos secundarios al resultado final de la imagen. Se hicieron dos corrida, la primera que verá a continuación se hizo con una cantidad de 8 rayos principales y 4 rayos secundarios:



Y en la siguiente imagen evidenciará la segunda corrida, realizada con de 4 rayos principales y 10 secundarios.



Como puede notar, la cantidad de rayos principales es la más importante, ya que son quienes determinan en qué posición está realmente la pared con la que chocan, por otro lado, los rayos secundarios aportan muy poca información, esto se ve en la segunda imagen que está más oscura en comparación con la primera. siendo la segunda imagen la que tenía mas rayos secundarios que primarios.

V. Conclusiones

El método de Monte Carlo, es usado para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud. Uno de los objetivos de este proyecto es aplicar el método de Monte Carlo en la simulación. Al aplicarlo, se logró percibir que este tipo de método es flexible, directo y aplicable para cualquier tipo de problema que tenga la característica principal de que la respuesta puede ser solo una aproximación a la realidad, después de eso elegir cuantos recursos se utilizan para obtener la mejor aproximación es la esencia del Monte Carlo y solo se consigue por medio de prueba y error, comparando cada uno de los resultados entre si. Las pruebas realizadas ejemplifican que conforme se tenga una mayor variedad cantidad de rayos, mejor será el resultado final, es decir, la respuesta aproximada, pero también mayores serán los tiempos de ejecución, la parte interesante es encontrar esa cantidad de recursos que dejen la mejor aproximación sin resentir en los tiempos de ejecución.

Otra conclusión es que el sonido es peor que la luz para la representación de imágenes, ya que el sonido genera imágenes más difusas e irregulares, mientras que la luz es capaz de representar la realidad de manera más fiel. En lo que concierne al proyecto, los rayos secundarios están ahí solo para aportar un efecto visual de imagen borrosa, pero no aportan ninguna otra cosa relevante, es decir, el resultado podría ser incluso mejor eliminando los rayos secundarios, y también se ahorrarían muchos recursos y tiempo de ejecución.

Por medio de este proyecto, se puede identificar la diferencia entre cómo viaja la luz y sonido. En la simulación los rayos principales por si solos podrían simular la forma en que funciona la luz, dejando una imagen más nítida, sin inconsistencias, por otro lado, al agregar rayos secundarios es claro que la imagen resultante posee ciertas inconsistencias que hace que la imagen pierda nitidez y no sea tan clara como una imagen generada por rayos de luz.

References

[1] Law, Y. (2020). Algoritmos Geométricos - Semana 7. https://drive.google.com/drive/folders/1iV5PDDpE7Wln8BiKedR2nQTzH7F3bSGt?usp=

[2] Law, Y. (2020). Algoritmos Probabilisticos - Semana 8. https://drive.google.com/drive/folders/1iV5PDDpE7Wln8BiKedR2nQTzH7F3bSGt?usp=

[3] Granados, S. Reveiz, M. (2020). Echo-Locator. https://github.com/revv-tech/Echo-Locator.git

[4] The Coding Train. (2019, 8 de Mayo). Coding Challenge 145: 2D Raycasting. https://www.youtube.com/watch?v=TOEi6T2mtHo

[5] Meznak. (2019). ray cast. https://github.com/meznak/ray_cast.git

[6] Law, Y. (2020). 2DRayTracer. https://github.com/yuenlw/2DRaytracer

[7] Fernández, J. (s. f.). Reflexión y refracción de la luz. https://www.fisicalab.com/apartado/reflexion-refraccion-luz

[8] StackOverflow. (2014). Get all points of a straight line in python. https://stackoverflow.com/questions/25837544/get-all-points-of-a-straight-line-in-python