Tarea Programada 2 Path Tracing

Tarea Programada de Análisis de Algoritmos

1st Ignacio Álvarez Barrantes
2019039643

nacho.alvarez2001@gmail.com

2nd Sebastián Gamboa Bolaños
2019044679

gamboasebas@gmail.com

***Abstract*—This project consists in the recreation of the method Path Tracing (with usage of Monte Carlo type algorithm) in order to create an application capable of simulating the behavior of real light and illumination effects. Given a certain 2D scenario the application must be capable of setting 1 or more points of light and illuminate the scene pixel by pixel. The application can be programmed in any language desired, for this project we will be using Pythn language with help of libraries like pygame, numpy, pyMTX and random.**

***Index Terms*—Path Tracing, Ray Tracing, Monte Carlo, Ray Casting, Intensidad Luminosa, Ángulo de Reflexión, Ángulo de Refracción**

I. Introducción

Este PDF tiene el fin de documentar el proceso de elaboración de la segunda tarea programada del curso de Anáisis de Algoritmos la cual consiste en la elaboración de una aplicación capaz de simular el comportamiento de un cuerpo luminoso y sus efectos al resto del ambiente en que se encuentre.

Para llevar acabo este proyecto es necesario investigar las fuentes de luz, sus comportamientos y características para luego así poder aplicar las mismas a la resolución del problema planteado. Se hará uso de diversos algoritmos y métodos ya conocidos como el Ray Tracing, Ray Casting, Monte Carlo Path Tracing y otros para así lograr simular el comportamiento de un cuerpo luminoso y sus efectos en el ambiente generado. Para el ambiente se usará un escenario de pregenerado por medio de casillas con ayuda de la librería pyTMX y assets conseguidos en internet, la cual posee diferentes figuras geométricas y assets para así poder obvservar el comportamiento de los rayos de luz al impactar con los distintos objetos.

Acontinuación una breve explicación de los métodos utilizados, estos serán vistos aún más a fondo y con código secciones adelante.

Path Tracing

El Path Tracing es un algoritmo de tipo Monte Carlo para la generación de gráficos por computadora. Este consiste en la creación de rayos con direcciones aleatorias (originadas de un epicentro), las cuales luego son rastreadas hasta un solo punto de un objeto (en este caso pixeles), luego se aplican diversas fórmulas matemáticas para así poder calcular la intensidad de la luz que se ve proyectada en dicho pixel, generando así sombras y focos de luz. Este método permite la creación de sombras suaves, generar la apariencia de profundidad e iluminación indirecta

Ray Casting

El Ray Casting es otro tipo de algoritmo que busca imitar el comportamiento de los rayos solares, generando lineas desde un punto con una dirección indicada si estas lineas impactan con un limite la linea es dibujada de no ser así existen 2 casos posibles: delimitar un cierto largo para la linea o simplemente no pintarla e ignorar el rayo (usualmente se usan los primeros casos). Entre más rayos sean usados mayor es la similitud con una fuente de luz real. Este método puede ser combinado con una variación de Path Tracing y por medio del uso de algoritmos Monte Carlo crear un programa interactivo (un personaje en movimiento iluminando una escena)

Ray Tracing

El Ray Tracing basado en el Ray Casting, aumenta la complejidad de las lineas generadas al igual que la de sus límites dándole características como las de un rayo de verdad (dirección, intensidad, color, capacidad de refracción, capacidad de reflexión y otros). Todos estos aspectos permiten la creación de una simulación fiel e interactiva

II. Planteamiento del Problema

A. Generación de la escena

Para la escena se utilizará hará uso de un mapa pre-generado por medio de la herramienta Tiled (https://www.mapeditor.org/) la cual permite crear mapas y agregar assets descargados de internet para luego ser cargados al proyecto por medio de la librería pyTMX. La escena cuenta con diversos obstáculos como paredes, mesas, sillas y arbustos para así ver el comportamiento de la fuente de luz al impactar con los distintos objetos. Los assets fueron obtenidos desde la página de assets 2D Kenney (https://kenney.nl/)

B. Creación de límites

Los límites usados serán simples lineas invisibles con características específicas, su ubicación es basada en el mapa prediseñado del cual se habló anteriormente. Las lineas poseen un tipo (opacas, translúcidas o espejos) según el tipo harán que el rayo se comporte de formas distintas. Cabe destacar que los tipos no pueden ser cambiados durante la corrida del algoritmo, si se desea cambiar el tipo de pared se debe de cerrar el programa establecer el nuevo tipo de pared y realizar la corrida nuevamente.

C. Fuente de luz

La fuente de luz utilizada en el programa corresponde al sujeto de camisa azul (personaje de la escena), la luz será emanada en la dirección que este apunte (como si hiciera uso de una linterna), para establecer los bordes se usará raycasting.

D. Rayos de luz

Los rayos de luz serán vistos como un simple objeto con tres simples atributos

- Color: en un principio el color base es el blanco, pero este puede ser modificado para que los rayos emanados iluminen de otros colores (su configuración no puede realizarse durante la ejecución del algoritmo).
- Dirección: todo rayo posee un ángulo o dirección el cual es asignado de forma pseudo-aleatoria, basado en el ángulo de visión del personaje.
- Fuente de origen: los rayos poseen una fuente de origen o ”pixel de origen”, los rayos de luz directa deben poseer todos puntos de origen similares pero los rayos de ”respuesta” (refracción o reflexión) poseen como punto de origen el punto de impacto de su rayo antecesor.

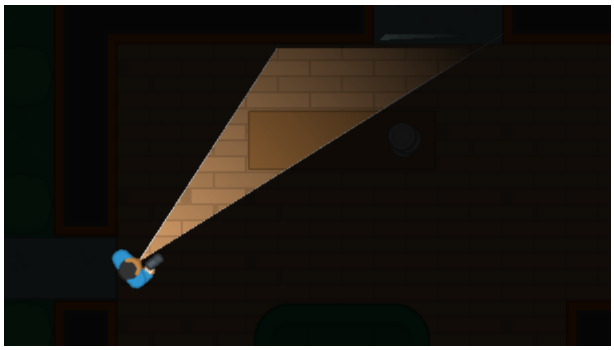
III. Trabajo Relacionado

A lo largo de la elaboración de este trabajo se hizo uso de diversos proyectos encontrados en internet para comprender el funcionamiento de Path Tracing y el Ray Casring y como poder aplicar estos métodos para iluminar un mundo en 2D, además la busqueda de información nos permitió encontrar la mejor forma para poder crear un mapa 2D, acontinuación una lista de lás páginas (En referencias) y proyectos utilizados para la elaboración de la tarea programada seguido de una breve explicación de los fragmentos usados de cada proyecto:

- Creación del escenario: Para la creación del escenario, como fue mencionado anteriormente, se utilizó la herramienta Tiled para la creación de mapas basados en casillas y para su carga se hizo uso de la librería pyTMX. La información y código para el uso de estos fue obtenida de un proyecto de programacón del canal de Youtube ”KidsCanCode” en cual explica el funcionamiento tanto de Tiled como de la librería pyTMX
- Ray Casting y Ray Tracing: Para la comprensión del código y el algoritmo en sí se recurrió a varios proyectos y videos de internet, de ellos se rescataron secciones de código optimizado para una mayor fluidez y sencillez del mismo. A su vez se usaron las páginas para comprender la diferencia entre Path Tracing y Ray Tracing y si existe la posibilidad de mezclar ambos métodos.
- Path Tracing: Para la comprensión del código se recurrió al proyecto adjunto del profesor, para la información sobre las secciones matemáticas del código se recurrió a Wikipedia y otras páginas

IV. Metodología

Para la resolución del problema se optó por usar una mezcla entre 2 métodos de iluminación encontrados ”Ray Casting” y ”Path Tracing”. El Path Tracing será utilizado para realizar la iluminación principal de la escena, pero esta iluminación será delimitada por un cono creado usando el método de Ray Casting para así dar la sensación de una linterna (Ver imágen adjunta).



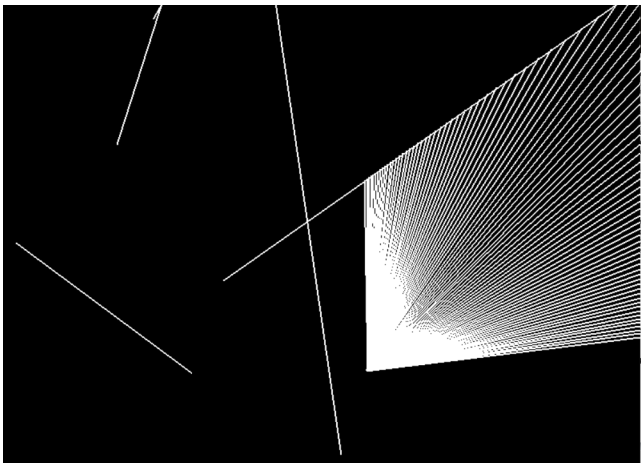
El ray Casting es el más básico de muchos algoritmos de renderización de gráficos por computadora que usan el algoritmo geométrico de ray tracing. Los algoritmos basados en ray tracing operan en orden de imagen para renderizar escenas tridimensionales a imágenes bidimensionales. Los rayos geométricos son trazados desde el ojo del observador (trazado hacia atrás) para calcular la radiancia que viaja hacia el observador en la dirección del rayo. La rapidez y simplicidad del trazado de los rayos provienen de computar el color de la luz sin trazar recursivamente rayos adicionales para obtener la radiancia incidente en el punto donde el rayo intercepta.

Esto elimina la posibilidad de renderizar con exactitud las reflexiones, refracciones, y las sombras. Por lo cual el Ray Casting es utilizado de forma primitiva para así crear un margen y poder reducir el tiempo de ejecución del algoritmo Path Tracing y dar una sensación de interactividad en tiempo real, como se explicará más adelante.

Uso del Ray Casting

Como se mecionó en párrafos anterirores el Ray Casting por si solo no es capaz de iluminar una escena, para ello requiere del Ray Tracing, en el caso de este proyecto la implementación del Ray Tracing no fue requerida.

El Ray Casting fue utilizado para delimitar la zona de afectación del Path Tracing permitiendo así disminuir el tiempo de ejecución del algoritmo al disminuir el total de pixeles a analizar, ya que debido al ángulo y paredes existen pixeles que no llegan a ser afectados del todo por los rayos generado por el Path Tracing



El total de rayos creados por el Ray Casting es disminuido a 2 con un ángulo entre ellos de 40 grados creando así un cono (en el cúal se encuentran los pixeles afectados por el Path Tracing).

La implementación del Ray Casting consiste en 3 clases:

A. *Ray:*

Contiene un vector3 el cual contiene [x,y,dirección], donde los puntos (x,y) indican el pixel de inicio del rayo la dirección es un vector en radianes usando la formula trigonometrica dado un radio = γ:

$$direccion = [sin(\gamma), cos(\gamma)] \tag{1}$$

Además de una tuple que marca el punto de intersección del rayo para así poder determinar el área de la figura.

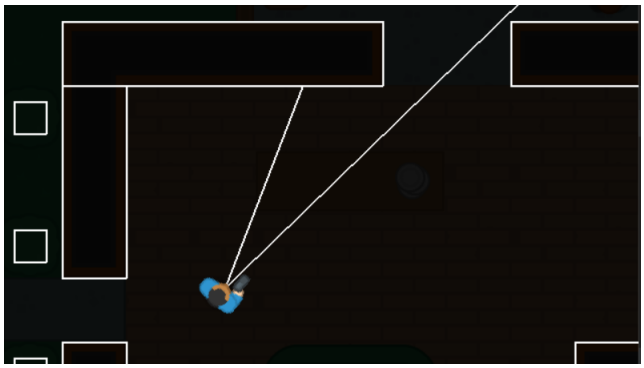
B. *Limite:*

Son las ”paredes” que marcan hasta donde llegan los rayos (en caso de ser opacas). Son rectas, lo que quiere decir que estan formadas por dos vectores A y B con sus respectivas coordenadas (x,y) además de un atributo el cual almacena el tipo de pared (opaca, translúcida o reflectiva)

C. *Particle:*

Es el punto de donde salen los rayos, posee coordenadas (x,y) además de un ángulo, indicando la dirección en la que está viendo la partícula en un momento dado, además de un array para los limites y para los rayos de la partícula.

La función principal es cast() la cual se encuentra dentro de la clase partícula esta es la función encargada de ”castear” los rayos sobre las paredes afectadas (retornando dichas paredes en un array)



USO DEL PATH TRACING

El Path Tracing es un método de Monte Carlo de gráficos por computadoras para el renderizado de imágenes de escenas tridimensionales de tal manera que la iluminación global sea fiel a la realidad. Fundamentalmente, el algoritmo está integrando sobre toda la iluminancia que llega a un solo punto de la superficie de un objeto. Esta iluminación se reduce entonces por una función de reflectancia de la superficie (BRDF) para determinar cuánto de esta se destinará a la cámara. Este procedimiento de integración se repite para cada píxel de la imagen de salida. Cuando se combina con modelos físicamente exactos de las superficies, modelos precisos de las fuentes reales de luz (bombillas), y cámaras ópticamente correctas, trazado de caminos puede producir imágenes fijas que son indistinguibles de las fotografías.

Trazado de caminos simula naturalmente muchos efectos que tienen que ser añadidos específicamente a otros métodos (trazado de rayos convencional o renderizado por exploración de líneas), tales como sombras suaves, profundidad de campo, desenfoque de movimiento y la iluminación indirecta. Una implementación de un renderizador que incluya estos efectos es correspondientemente más simple.

Debido a su exactitud e imparcial naturaleza, trazado de caminos se utiliza para generar imágenes de referencia cuando se prueba la calidad de otros algoritmos de renderizado. Con el fin de obtener imágenes de alta calidad con trazado de caminos, un gran número de rayos deben ser procesados para evitar ruido visible.

Al ser este el método usado para la iluminación sus clases y atributos aumentan en complejidad comparado con las clases anteriores su implemenatción consta de las siguientes clases:

D. Limite:

Usa la misma clase Limite para definir las barreras de los rayos

E. Pixel:

Clase que intenta simular los pixeles de la imagen, simplemente contiene un vector [x,y] indicando la ubicación de cada pixel

F. VectorOperations:

Clase que maneja los cálculos de vectores para determinar las direcciones de los rayos y sus intensidades.

G. Path Tracing:

No es una clase en sí, pero cabe mencionarla por si sola ya que esta es la función encargada de realizar la iluminación:

```
def pathTracer(self):

    minW = self.player.pos[0] -
        LIGHT_MAX_DISTANCE
    if (minW < 0):
        minW = 0

    maxW = self.player.pos[0] +
        LIGHT_MAX_DISTANCE
    if (maxW > WIDTH):
        maxW = WIDTH

    minH = self.player.pos[1] -
        LIGHT_MAX_DISTANCE
    if (minH < 0):
        minH = 0
    maxH = self.player.pos[1] +
        LIGHT_MAX_DISTANCE
    if (maxH > HEIGHT):
        maxH = HEIGHT


    for i in range(int(minW), int(maxW)):
        for j in range(int(minH), int(maxH)):

            alpha = 0
            point = Point(i, j)

            if(self.player.particle.inRange
            (point.x,point.y)):

                source =
                    Point(self.player.pos[0],
                    self.player.pos[1])

                if (point.x != source.x or
                    point.y != source.y):
                    point = Point(i, j)

                dir = source - point
```

```
                length =
                    vectorOperation.length(dir)
                length2 =
                    vectorOperation.length
                (vectorOperation.normalize(dir))

                free = True
                for seg in segments:

                    dist = vectorOperation.
                    raySegmentIntersect(point,
                        dir,
                    seg[0], seg[1])

                    if dist > 0 and length2 >
                        dist:
                       free = False
                       break

                if free:
                    alpha =
                        round(((LIGHT_MAX_DISTANCE
                        - length) /
                        LIGHT_MAX_DISTANCE) *
                        255)

            if (alpha < 0):
                alpha = 0

            pygame.gfxdraw.pixel(self.fog, i,
                j,
            (255, 255, 255, alpha))
```

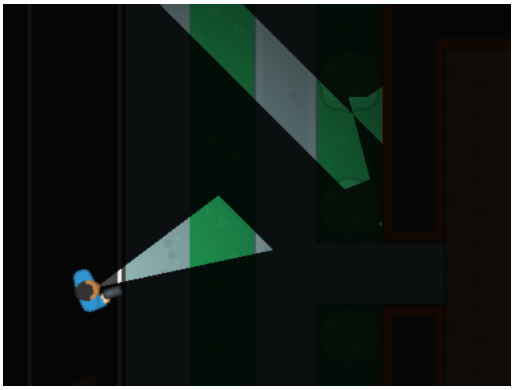
El algoritmo funciona como un Path Tracing normal, la única diferencia es la forma en la que los pixeles son elegidos, ya que se usa un pseudo-random (como se usaría normalmente en algoritmos Monte Carlo) pero el rango usado para los randoms está definido por el área abarcada por el cono (La cual es calculada por medio del Ray Casting), para así disminuir el total de pixeles a ser procesados durante la corrida del algoritmo.

V. Análisis de Resultados y Conclusiones

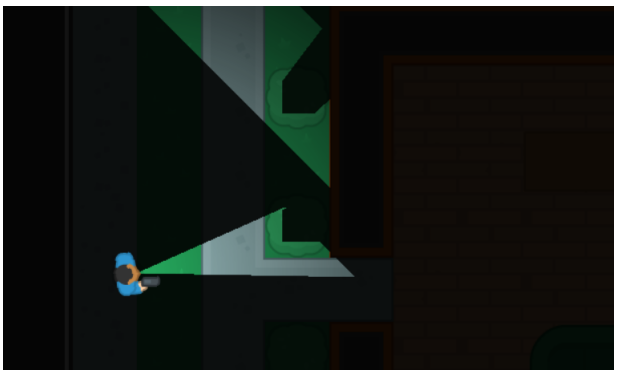
La forma en la que esta implementado el programa permite interactuar con la fuente de luz, dándole al usario la posibilidad de iluminar o no ciertos sectores del ”mapa”. Dependiendo del sector iluminado las corridas pueden ser tardías (1-2 minutos) o muy cortas(20-30 segundos), además depende del rango asignado a la linterna ya que puede ser configurada para afectar menos pixeles o más pixeles. Además el número de iteraciones por pixel incide directamente en la duración del tiempo, el algoritmo realiza 15 por pixel para disminuir el ”noise” al máximo en la imágen resultante.

Por ejemplo:

La imagen acontinuación muestra el ejemplo de una corrida, con una intensidad de linterna de 400. El algoritmo tardó un total de 48.23 segundos en ejecutarse.



La siguiente imágen muestra una segunda corrida realizada en la misma posición, pero con una intensidad de 600, el tiempo de corrida fue de 1:42.32 segundos



Al concluir las pruebas se obtuvieron varias conclusiones:

- Entre mayor intensidad, mayor es la distancia de los rayos, al igual que la intensidad de sus refracciones y reflexiones.
- Los tiempos de corrida aumentan de forma exponencial dependiendo del numero de iteraciones por pixel, ya que si se elige aumentarel numero de iteraciones los análisis de pixel por pixel serían (Largo*Ancho)* número de iteraciones deseadas.
- Al ser dinámico se logra determinar como iluminar ciertos sectores permiten realizar una corrida más rápida debido a que pueden existir un número menor o mayor de interacciones entre los rayos y el mapa.
- Además cabe recalcar que aunque utilizando los pseudo-randoms gracias al Ray Casting la imagen obtenida no es una simulación 100 por ciento exacta a una iluminación real, ya que la imágen se ve afectada por factores como el ”noise” creado al intentar iluminarla por medio de

este método. Al igual que ciertas iluminaciones indirectas pueden llegar a ser omitidas. Pero si reduce significablemente el tiempo de corrida. Por lo cual si se desea obtener una imágen más precisa (si se poseen recursos y tiempo) lo mejor es evitar el uso del Ray Cast y recorrer la matriz completa a costa de perder el dinamismo de iluminación.

VI. References

2-D Path Tracing With Inverse Kinematics- MATLAB Simulink. Mathworks.com. (2015). Rescatado el 18 June 2020, de https://www.mathworks.com/help/robotics/ug/2d-inverse-kinematics-example.html.

Coding Challenge 145: 2D Raycasting. YouTube. (2019). Rescatado el 18 June 2020, de https://www.youtube.com/watch?v=TOEi6T2mtHo.

Disney's Practical Guide to Path Tracing. YouTube. (2016). Rescatado el 18 June 2020, de https://www.youtube.com/watch?v=frLwRLS-ZR0.

Kuri, D. (2018). GPU Path Tracing in Unity – Part 2 – Three Eyed Games. Three-eyed-games.com. Rescatado el 18 June 2020, de http://three-eyed-games.com/2018/05/12/gpu-path-tracing-in-unity-part-2/.

Vandevenne, L. (2012). Raycasting. Lodev.org. Rescatado el 18 June 2020, de https://lodev.org/cgtutor/raycasting.html.

Building a Ray Tracer in Python - Part 1 (Points in 3D Space) YouTube. (2019). Rescatado el 18 June 2020, de https://www.youtube.com/watch?v=KaCe63v4D-Q

Building a Ray Tracer in Python - Part 3 (3D Balls in 2D Space) (2019). Rescatado el 18 June 2020, de https://www.youtube.com/watch?v=1DKlb3BQ-k8

Online Computer Graphics II: Rendering: Monte Carlo Path Tracing: Simple Path Tracer (2019). Rescatado el 18 June 2020, de https://www.youtube.com/watch?v=KCYroQVaARs

Online Computer Graphics II: Rendering: Monte Carlo Integration: Basic Probability and Sampling (2019). Rescatado el 18 June 2020, de https://www.youtube.com/watch?v=CArmbrH-4VI