Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
Segundo semestre 2017

Tarea 2

Fecha de entrega: Jueves 11 de Octubre

Objetivos

- Investigar sobre una nueva estructura de datos.
- Modelar y resolver problemas de visualización de manera eficiente.

Introducción

DCConstructora es una empresa de arquitectura que busca crear un software para analizar cómo se vería la iluminación de sus diseños. Es por esto que te han contactado para que puedas computar la iluminación de los distintos puntos según las luces y las paredes dentro de un plano.

Problema: Visibilidad

El problema de iluminar un plano según una serie de luces y obstáculos es una generalización del problema de **visibilidad** entre dos puntos. Para que un agente pueda ver a otro en el plano, es necesario que no hayan obstáculos entre ellos. Esto significa responder la siguiente pregunta:

Dados dos puntos A y B en el plano, ¿Existe algún obstáculo que impida ir en línea recta de uno a otro?

Lo que de manera geométrica puede expresarse como

Dados dos puntos A y B en el plano, ¿El segmento que une A y B es intersectado por algún obstáculo?

En dos dimensiones, la figura básica es el segmento de recta, el cual puede ser usado para construir obstáculos de forma arbitraria. Asumiendo que los obstáculos están expresados como segmentos de recta, es posible entonces plantear lo siguiente:

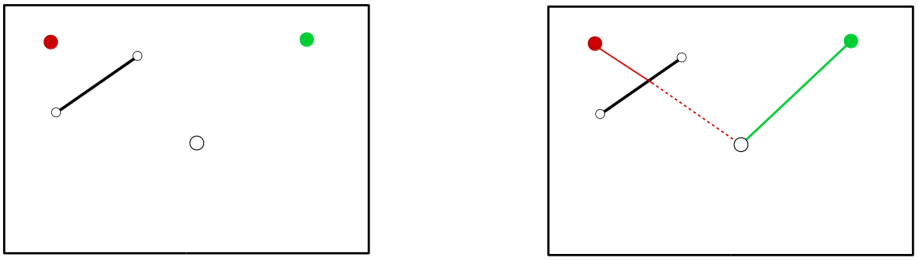
Un punto A es **visible** desde un punto B
si y solo si
El segmento que une A y B no intersecta con ningún segmento de los obstáculos.

Problema: Iluminación

El prolema de **iluminación** consiste en decidir el nivel de luz que recibe un punto. Esto está dado por las luces que lo iluminan directamente:

Un punto P es iluminado directamente por una luz L
si y solo si
L es **visible** desde P

La iluminación total de un punto se define como la suma de los **aportes** de cada luz que lo ilumina directamente, por lo que es necesario computar la visibilidad de cada luz para un punto. Esto se ilustra en la Figura 1.



(a) Configuración inicial

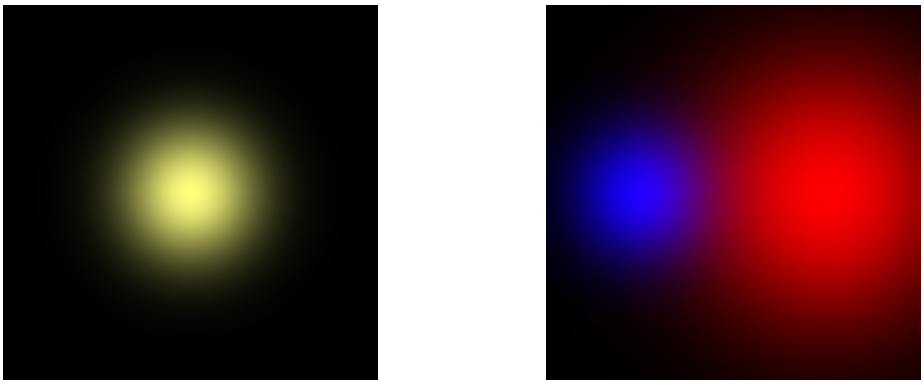
(b) Configuración con rayos de luz

Figura 1: Ejemplo de iluminación de un punto con luces roja y verde

Para efectos de esta tarea, asumiremos que las luces son puntuales, y que el **aporte** de una luz está dado por la siguiente fórmula:

$$C \cdot e^{\frac{-d^2}{2 \cdot I^2}}$$

Donde d es la distancia entre P y la luz, I es la intensidad de la luz, y C es el color de la luz en RGB. Esto puede verse en la Figura 2.



(a) El aporte de la luz decae al aumentar la distancia

(b) La suma de los aportes mezcla los colores de estas luces de distintas intensidades

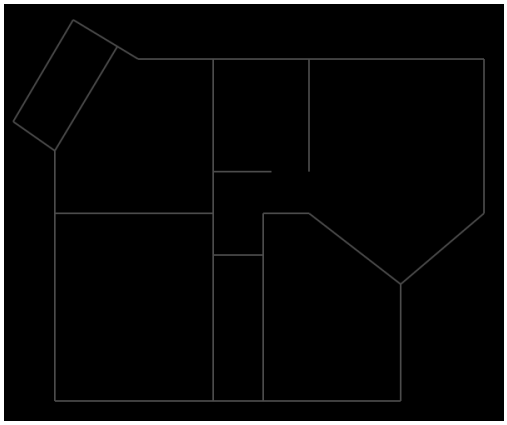
Figura 2: Ejemplo de cómo la distancia e intensidad afectan el aporte de una luz

Generación de imágenes

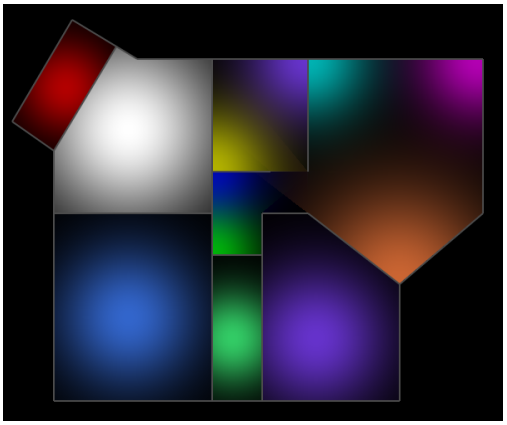
De lo anterior se puede construir el algoritmo para iluminar el plano de una imagen, de la siguiente forma:

```
for all pixel P in image do
    P.color ← black
    for all light L in map do
        if L is visible from P then
            P.color ← P.color + contribution of light L at point P
        end if
    end for
end for
```

Lo que da el resultado que puede verse en la Figura 3.



(a) Los obstáculos aquí son las paredes de una casa



(b) Se calcula el color de cada pixel segun las luces que le llegan

Figura 3: Ejemplo del algoritmo de iluminación en acción

Es claro ver que este algoritmo es $\mathcal{O}(PLS)$, donde P es la cantidad de píxeles, L la cantidad de luces, y S la cantidad de segmentos obstáculo.

Tu deber es implementar en C este algoritmo, pero además organizar los obstáculos en una estructura de datos de manera de poder calcular **visibilidad** en $\mathcal{O}(log(S))$, y así poder calcular la iluminación de una imagen en $\mathcal{O}(PL \cdot log(S))$

Para esto deberás utilizar la estructura de datos **KD-Tree** para dos dimensiones.

Análisis

Como se mencionó anteriormente, tendrás que implementar la estructura de datos llamada **KD-Tree**.

Deberás escribir un informe autocontenido en el que expliques a rasgos generales las decisiones que tuviste que tomar a la hora de construir tu árbol. Debes respaldar esto con visualizaciones de tu árbol construidas usando la interfaz gráfica. (Tanto de divisiones de ejes como de *bounding boxes*)

Además, deberás incluir un análisis empírico de cómo afecta la estructura del árbol el rendimiento del código. Para esto, tendrás que medir los tiempos de ejecución tanto te la creación del arbol como de la iluminación de la imagen al cambiar:

- El punto en el que se corta un eje (mediana, o algo que tenga sentido)
- La profundidad o cantidad de elementos a la que se deja de cortar.
- Que hacer con los segmentos que quedan a ambos lados del corte (dividirlos, replicarlos)

Este análisis deberá estar acompañado por los datos dispuestos en gráficos según sea pertinente.

Input

Tu tarea debe recibir el siguiente input:

```
./plotter <test.txt> <out.png>
```

Donde:

- `<test.txt>` es el archivo con la descripción de la escena
- `<out.png>` es el nombre de la imagen de output del programa

El archivo `<test.txt>` posee la siguiente estructura:

- La primera línea contiene el alto y el ancho de la imagen, `H W`
- La siguiente línea contiene la cantidad de luces `L`
- Las siguientes `L` líneas corresponden a cada luz de la imagen, las cuales contienen su posición `x y`, su color `RGB` y si intensidad `I` de la forma `x y R G B I`
- La siguiente línea contiene la cantidad de segmentos `S` que serán los obstáculos.
- Las siguientes `S` líneas corresponden a ambos extremos de cada segmento de recta, de la forma `xi yi xf yf`

Output

El output de tu programa es mostrar correctamente la iluminación del plano en la interfaz gráfica, detallado en la siguiente sección.

Watcher

Es la librería que permite visualizar el problema y generar la imagen de output. Es extremadamente útil para debuguear el código y es necesaria para generar el output. Las funciones provistas para su uso son las siguientes:

- `watcher_open(height, width)`: Abre una ventana para mostrar una imagen de las dimensiones dadas.
- `watcher_set_color(R, G, B)`: Selecciona un color en formato RGB para las siguientes operaciones de dibujo.
- `watcher_draw_segment(xi, yi, xf, yf)`: Dibuja un segmento de recta de un punto a otro del color seleccionado. Puedes usar esto para dibujar los obstáculos, además de otras líneas que te sirvan para debugear tu árbol (como los ejes de división, o las *bounding boxes* de los árboles)
- `watcher_paint_pixel(row, col)`: Pinta el pixel del color seleccionado.
- `watcher_snapshot(filename)`: Genera una imagen PNG con el contenido actual de la ventana. Úsala para sacarle fotos a tu árbol.
- `watcher_close()`: Cierra y libera los recursos de la ventana.

Evaluación

Tu código será evaluado con diferentes tests y se evaluará la imagen final que haya en la interfaz al cerrar el `watcher`, ignorando las líneas creadas usando `watcher_draw_segment`

La nota de tu tarea se descompone como se detalla a continuación:

- 60% A que tu código genere las imágenes correctamente
- 40% A tu informe, específicamente:
  - 10% a la explicación del árbol.
  - 10% a cada ítem del análisis empírico (30% total)

Para esta tarea **se aplicará descuento por ortografía**, a criterio del corrector.

Tu código será probado con escenas con distintas cantidades de píxeles, luces, y obstáculos. Se espera que tome tiempo menor que la versión sin KD-Tree, por lo que será cortado si demora mucho.

Entrega

- **Lugar:** GIT - Repositorio asignado (asegúrate de seguir la estructura inicial de éste).
  - En la carpeta *Programa* debe encontrarse el código.
  - En la carpeta *Informe* debe encontrarse un archivo de formato PDF de nombre **exactamente** `Informe.pdf` con el análisis de la tarea.
  - Se recogerá el estado en la rama *master*.
- **Hora:** 1 minuto pasadas las 23:59 del día de la entrega.
- Se espera que el código compile con `make` dentro de la carpeta *Programa* y genere un ejecutable de nombre `plotter` en esa misma carpeta.
- No se permitirán entregas atrasadas.

Bonus

- **Buen uso de espacio y del formato (+10% a la nota del *Informe*)** :
  La nota de tu informe aumentará un 10% si tu informe está bien presentado y usa el espacio y formato de manera de transmitir mejor la información. A juicio del corrector.

- **Manejo de memoria perfecto (+10% a la nota de *Código*):**
  Se aplicará este bonus si **valgrind** reporta en tu código 0 leaks y 0 errores de memoria, considerando que tu programa haga lo que tiene que hacer.

- **Análisis de una solución alternativa (+1.5 a la nota *final*)** :
  Tu nota final de la tarea puede aumentar si presentas otra forma de resolver el problema, incluyendo análisis de complejidad en términos de memoria y tiempo de ejecución. El puntaje se obtendrá si y solo si el análisis es correcto y se presenta una solución más eficiente en notación $\mathcal{O}$. Para efectos de este análisis puedes considerar que el aporte de una luz no depende de la intensidad ni de la distancia.

Anexo: Geometría

Un segmento de recta entre los puntos P_a y P_b puede expresarse como la interpolación lineal de ambos puntos. Es decir,

$$S_{ab}(t) = t \cdot P_a + (1-t) \cdot P_b \qquad t \in [0,1]$$

Sea $S_{12}(t_1)$ el segmento que une los puntos P_1 y P_2 y $S_{34}(t_2)$ el segmento que los puntos P_3 y P_4.

Al igualarlos se tiene:

$$S_{12}(t_1) = S_{34}(t_2) \qquad t_1, t_2 \in [0,1]$$

Lo que equivale a

$$\begin{aligned} t_1 \cdot P_1 + (1-t_1) \cdot P_2 &= t_2 \cdot P_3 + (1-t_2) \cdot P_4 \\ t_1 \cdot P_1 + P_2 - t_1 \cdot P_2 &= t_2 \cdot P_3 + P_4 - t_2 \cdot P_4 \\ t_1 \cdot P_1 - t_1 \cdot P_2 + t_2 \cdot P_4 - t_2 \cdot P_3 &= P_4 - P_2 \\ t_1 \cdot (P_1 - P_2) + t_2 \cdot (P_4 - P_3) &= P_4 - P_2 \end{aligned}$$

De manera matricial

$$\begin{aligned} \begin{bmatrix} (P_1 - P_2) & (P_4 - P_3) \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} &= \begin{bmatrix} (P_4 - P_2) \end{bmatrix} \\ \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} &= \begin{bmatrix} (P_1 - P_2) & (P_4 - P_3) \end{bmatrix}^{-1} \cdot \begin{bmatrix} (P_4 - P_2) \end{bmatrix} \end{aligned}$$

Viendo los valores en las coordenadas:

$$\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} (P_{1_x} - P_{2_x}) & (P_{4_x} - P_{3_x}) \\ (P_{1_y} - P_{2_y}) & (P_{4_y} - P_{3_y}) \end{bmatrix}^{-1} \cdot \begin{bmatrix} (P_{4_x} - P_{2_x}) \\ (P_{4_y} - P_{2_y}) \end{bmatrix} \tag{1}$$

Recordar que

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\Delta} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

con $\Delta = ad - cb$.

En este caso, si Δ es 0, los segmentos son paralelos, por lo que no es posible que intersecten.

De otro modo, es necesario resolver el sistema en (1).

Los segmentos intersectan entonces si se cumple que ambos $t_1, t_2 \in [0,1]$