CS 7353: Analysis of Algorithms Homework 4: Visible Lines Problem

Noah Schrick

March 8, 2022

Contents

1 Problem Introduction

The Hidden Surface Problem is widely used in computer graphics. When displaying images, not all images should be visible (for example, if one object

is in front of another). To avoid overlapping images from being displayed and causing collisions, computations are performed to remove all or parts of an object from the field of vision. This problem is a simplified version focusing on linear lines in a 2D plane. Rather than handling shifting perspectives, this problem will be further simplified to focus on the perspective as looking down from y=∞.

The problem statement can be defined as follows:

You are given n nonvertical lines in the plane, labeled L_1,...,L_n, with the ith line specified by the equation y=a_ix+b_i. We will make the assumption that no three of the lines all meet at a single point. We say line L_i is uppermost at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0: $a_i x_0 + b_i$ ¿ $a_j x_0 b_j$ for all j≠i.

We say line L_i is visible if there is some x-coordinate at which it is uppermost-intuitively, some portion of it can be seen if you look down from "y=∞".

Determine which lines are visible and at which restricted domains.

2 Program Preface

This assignment was given with instruction to implement this in C++, or justify why Python should be used instead. This problem was solved using C++ on a Linux system. Attached with submission is a zip folder that contains:

- A CMakeLists.txt file for compiling
- An "images" folder that contains:
    1. Various images included in this report
- A "src" folder that contains:
    1. A Line.cpp and Line.h file for the Line class and associated functions
    2. A HiddenLines.cpp and HiddenLines.h file for the problem instance class and associated functions
    3. A GNUPlot .hpp file for plotting
    4. The main file
- A "data" folder that contains:
    1. A CSV file for defining the line functions to use

- A "build" folder that contains:
    1. A build.sh script to simplify the build process
    2. A run.sh script to simplify running the program
    3. Various CMake files
    4. The compiled binaries for the program and associated libraries

This program requires the installation of GNUPlot for plotting purposes, but modification can be performed to remove the plotting. Removing the plotting does not alter the solution computation. In addition, this program promises no guarantee of working on other Operating Systems - no testing was conducted on any other platform besides the local Linux machine.

3 Programming Approach

3.1 Line Class

To aid in the derivation of the solution, the first step was to create a Line class. This class has members of slope, y-intercept, an ID, the x-value at which visibility starts, and the x-value at which visibility ends. The ID is associated with the order from the CSV file, and the visibility values are both initialized to 0.

Functions included with this class are getters and setters for various members, and operator overloads for == and < to aid in the sorting of Lines based on slope when using sets. Also included is a function to find the x-value of the intersection between two lines.

3.2 HiddenLines Class

A separate class was created for handling the problem space. The HiddenLines class contains few members of its own, and is primarily function-based. The single member that this class has is a vector of all lines for the given problem.

This class contains four functions: a function to construct the problem, a function to get the vector of lines, a function to generate the solution, and a function to print the solution.

3.2.1 Constructing the Problem

For modularity, this problem uses a CSV file for constructing the problem. Any and all lines can be stored in a CSV file in the data folder. This CSV file is of format (slope,y-intercept). This file is read and parsed, then inserted into a set to order the lines by slope. After all lines are imported, the set is then copied to a vector for ease of use for other functions. Basic error checking is performed only in the sense of ensuring that there are the same number of slopes and y-intercept entries.

3.2.2 Generating the Solution

This problem uses a divide-and-conquer approach. The base cases are for line vectors of size 1 or 2. When a line vector is of size 1, it is returned with no further computation. When a line vector is of size 2, the two lines have their slopes compared and their intersection point found. The line with the smallest slope is visible up until the intersection point, and the line with the greater slope is visible starting at the intersection point. The lines have their underlying visibility range members adjusted, and they are returned to be merged with the other divided half.

The merge process walks through each halved list and compares lines between the halves. The lines between the vectors have their slopes and intersection points compared, similar to the aforementioned process, but a separate function for removing the invisible lines is also called after the halves are merged. The removal process returns a new vector container that only holds the visible lines, after further comparisons of lines and intersection points.

3.2.3 Printing the solution

The solution is printed by walking through the final container and printing each line's ID, their visibility start point, and their visibility end point. If the start or end point is the maximum or minimum value of a double, then it is converted to a string of positive or negative infinity, respectively. In addition, GNUPlot prints the original problem space, and the final visible lines. As a note, extra work is needed to convert the final solution to a piecewise plot. At the current moment, all lines that are visible are printed across the entire domain, rather than their restricted domain.

4 Results

Figure 1 shows the output of the terminal after the program completes. This solution was manually verified with Desmos by plotting each line and comparing their intersections. Figure 2 displays the plotting of the original problem with GNUPlot. Figure 3 displays the plotting of the only visible lines, though as mentioned in Section 3.2.3, further work needs to be conducted to plot the piecewise functions correctly.

```
[noah@NovaArchSys build]$ ./run.sh
Constructing Problem...

---------------Sorted Lines by slope:---------------
Line 6 has slope -9 and a y-intercept of -38
Line 5 has slope -1.45 and a y-intercept of 5.4
Line 4 has slope -0.947 and a y-intercept of -1.368
Line 3 has slope -0.059 and a y-intercept of -6.412
Line 2 has slope 0.36 and a y-intercept of -12.32
Line 1 has slope 0.808 and a y-intercept of -6.692
Line 8 has slope 2.364 and a y-intercept of 6.545
Line 7 has slope 5.833 and a y-intercept of -1.5
----------------------------------------------------

Generating Solution.

Solution is:
Line ID 6 visible from x=-inf to x=-5.748344
Line ID 5 visible from x=-5.748344 to x=-0.300210
Line ID 8 visible from x=-0.300210 to x=2.319112
Line ID 7 visible from x=2.319112 to x=inf

Press ENTER to continue...
```

Figure 1: Path Walking to State 14

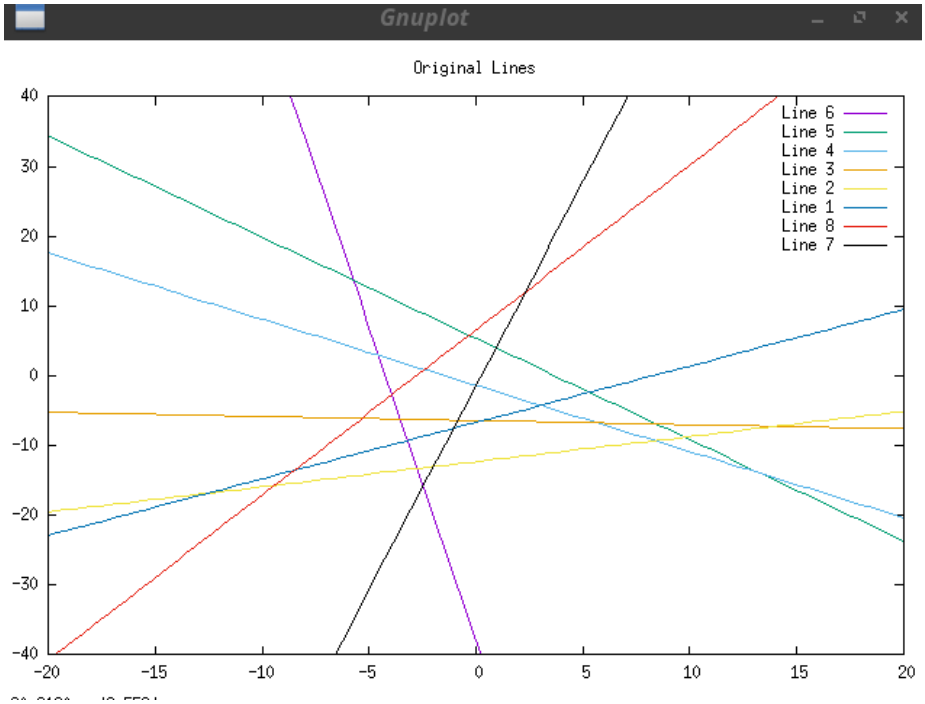

Figure 2: Path Walking to State 14

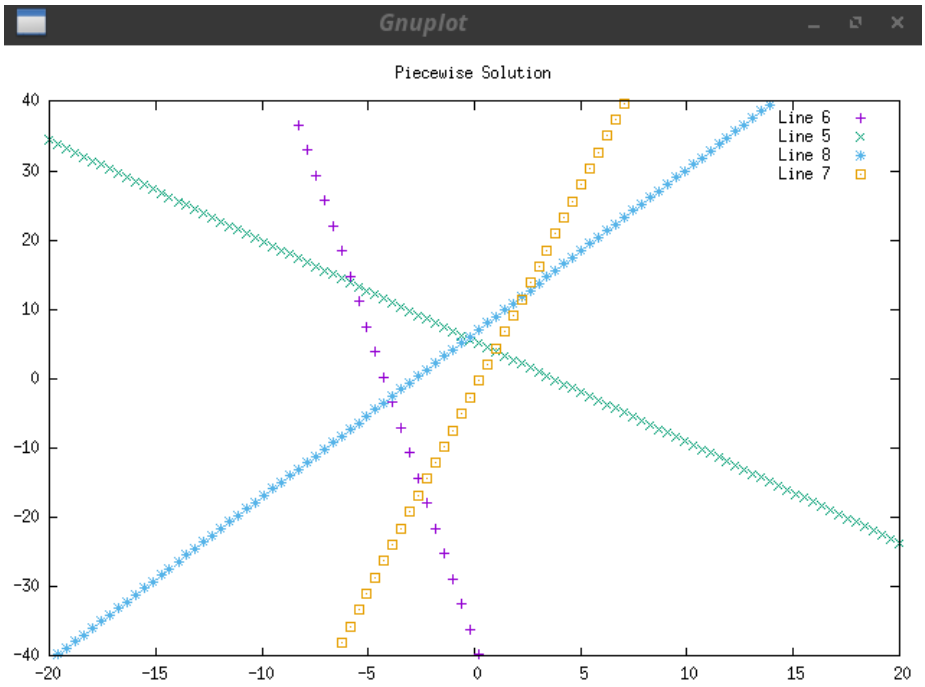

Figure 3: Path Walking to State 14