Sheriff Woody

FIGURE – Toy story character : Sheriff Woody [1]

1. Reach for the sky !

Buzz Lightyear

FIGURE – Toy story character : Buzz Lightyear [2]

2. Don't yank my string !

Introduction and Motivation

Hidden Surface Removal(HSR)

- Hidden surface removal is one of the basic problem in Computer Graphics.
- In case of multiple 3D surfaces, at the time of rendering which surface will be rendered and which one is not?
- The ultimate goal is reducing computational time.

Introduction and Motivation

Hidden Surface Removal(HSR)

- Hidden surface removal is one of the basic problem in Computer Graphics.
- In case of multiple 3D surfaces, at the time of rendering which surface will be rendered and which one is not ?
- The ultimate goal is reducing computational time.

Different Names

- Hidden Surface Determination (HSD)
- Occlusion Culling (OC)
- Visible Surface Determination (VSD)

HSR

Hidden surface removal is a problem in computer graphics that scarcely needs an introduction : When Woody is standing in front of Buzz, you should be able to see Woody but not Buzz ; When Buzz is standing in front of Woody . . . well, you get the idea.
The magic of hidden surface removal is that you-can often compute things faster than your intuition suggests. Here's a clean geometric example to illustrate a basic speed-up that can be achieved. You are given n non vertical lines in the plane, labelled $L_1, L_2, \ldots, L_n$ with the i^{th} line specified by the equation $y = a_i.x + b_i$. We will make the assumption that no three 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 that 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 $j \neq i$. We say line L_i is visible if there is some x coordinates at which it is uppermost intuitively, some portion of it can be seen if you look down from $y = \infty$.
Give an algorithm that takes n lines as input and in $O(nlogn)$ time returns all of the ones that are visible.

Input-Output

Input

A set of lines

- no. of lines
- slope(m) and intercept(b) pairs

Input-Output

Input

A set of lines

- no. of lines
- slope(m) and intercept(b) pairs

Output

Visible set of line(s)

- visible line(s) equation(y=m*x+c)
- intersection points

Input-Output

Input

A set of lines

- no. of lines
- slope(m) and intercept(b) pairs

Output

Visible set of line(s)

- visible line(s) equation(y=m*x+c)
- intersection points

Line : pair of slope and intercept with y-axis

Assumptions

- None of the lines must be vertical($slope \neq \infty$).
- No three lines all meet at a single point.

Problem Visualization

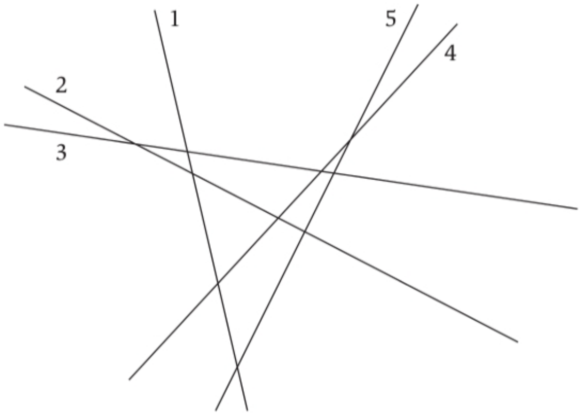


FIGURE – A set of lines

Problem Visualization

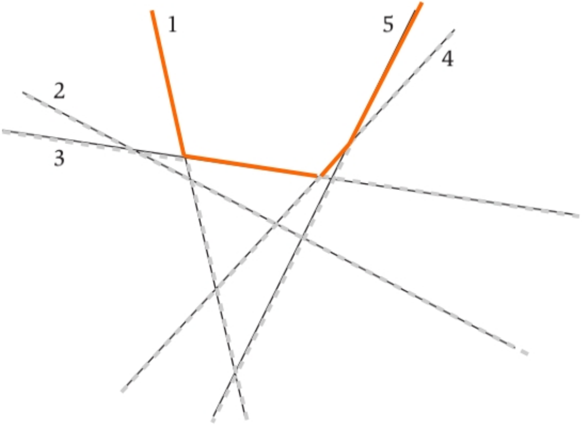


FIGURE – A set of lines

Observations

Before going to conventional approach we need to understand some important observations.

- If we start by sorting the lines in order of increasing slope. Notice that the first and last lines in this order will always be visible.
- If we have two lines with different slope then both lines are visible, lower slope line visible to left with respect to intersection point and higher slope line visible to right region.
- If we have two visible line and we add 3^{rd} line to check whether it is visible or not ? So we will find intersection point and check where it is lying in the left region or in the right.

Algorithm

Algorithm 0: Conventional Approach

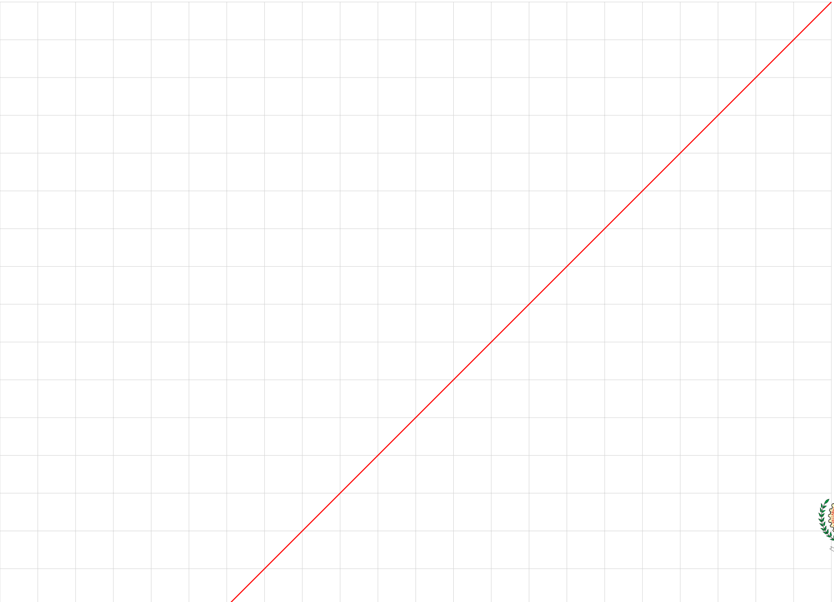
```
input  : A set of line(s)
output : visibleSet(lines, intersection points)
visibleSet=empty;
def line():
    label;
    slope;
    intercept;
def point():
    x-coordinate;
    y-coordinate;
Function addLine(l: line):
    return line added or not;
foreach i in the input set do
    if i==1 then
        visibleSet.addLine(line);
    else if i==2 then
        visibleSet.addLine(line1, line2);
        visibleSet.addPoint(line1, line2);
    else if i==3 then
        visibleSet.addLine(line1, line2, line3);
        visibleSet.point(line1, line2, line3);
    else
        visible.addLine(line);
```

Analysis

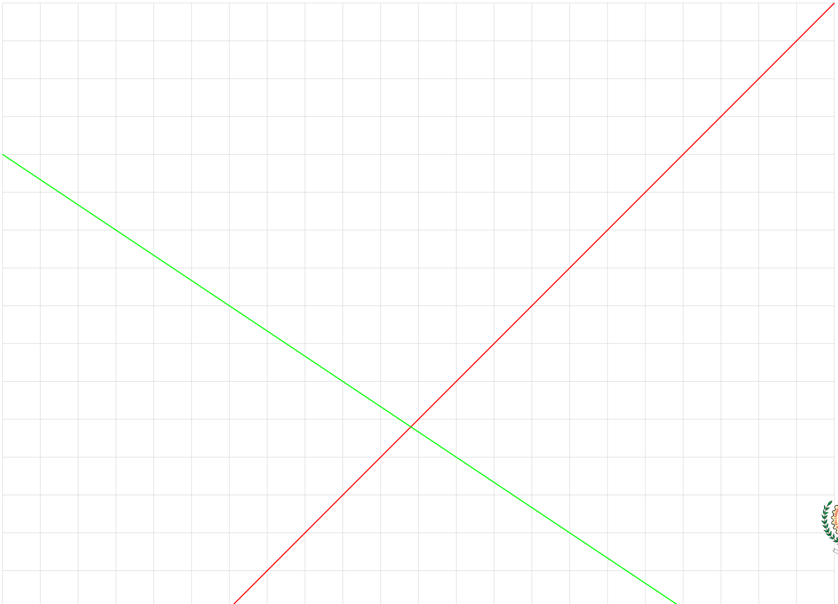
Analysis

1. taking a set of lines .... $O(1)$
2. adding line by line ... $O(n)$
3. for each line in visible set check that new line is visible or not $O(n)$ .... $O(n^2)$
4. Time complexity $= O(1) + O(n^2) = O(n^2)$

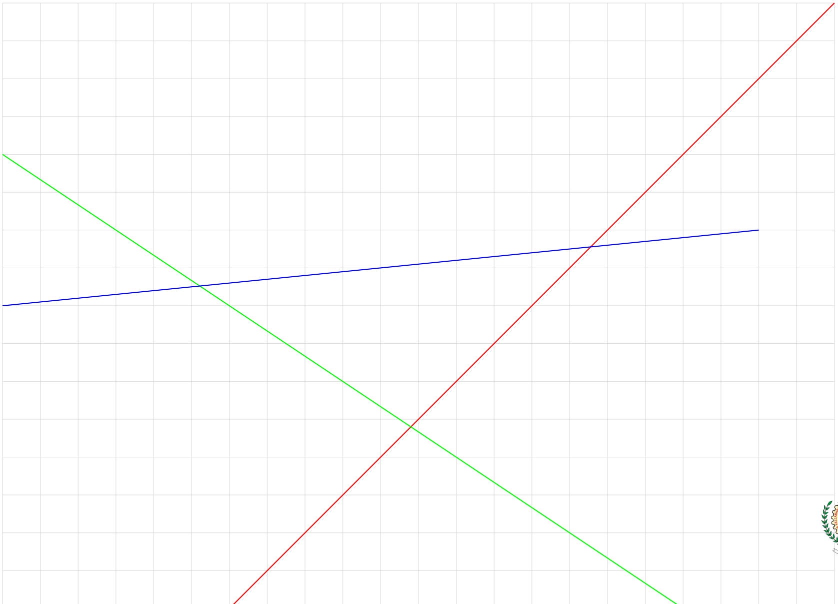
Example



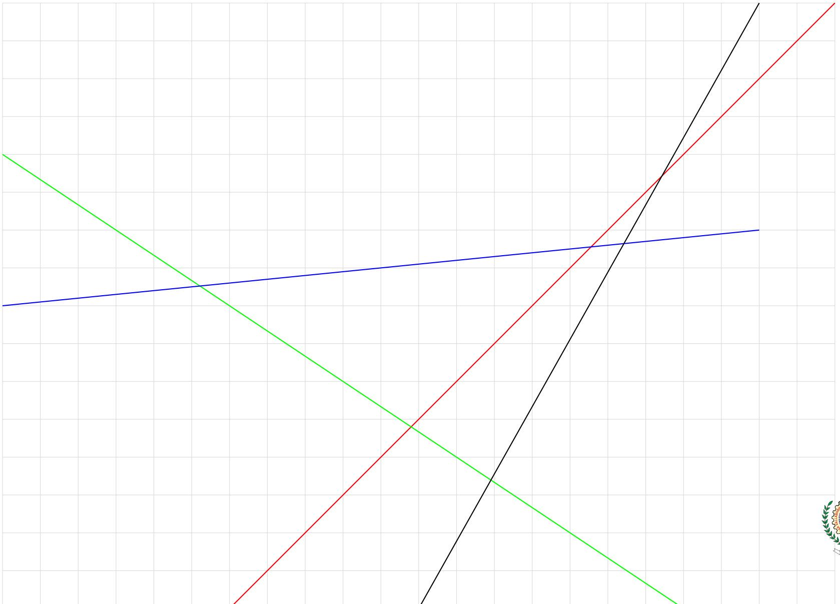
Example



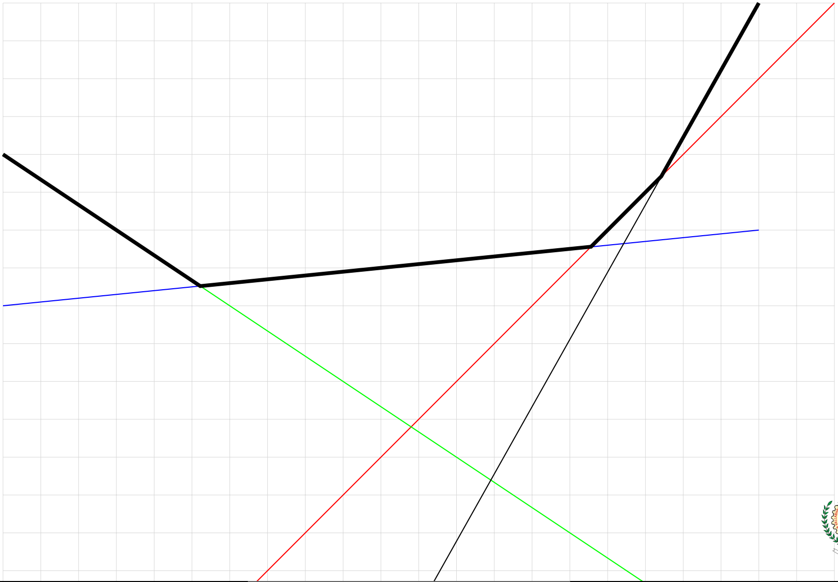
Example



Example



Example



Algorithm

Algorithm 1: Divide and Conquer

```
input  : A set of line(s)
output : visibleSet(lines, intersection points)
visibleSet=empty, n=no. of lines in set;
if n==1 then
    visibleSet.addLine(line);
else if n==2 then
    visibleSet.addLine(line1, line2);
    visibleSet.addPoint(line1, line2);
else if n==3 then
    visibleSet.addLine(line1, line2, line3);
    visibleSet.point(line1, line2, line3);
else
    mid = n/2;
    v1=visible(0, mid);
    v2=visible(n-mid, n);
    merge(v1,v2);
Function merge(set: visibleSet1, set: visibleSet2):
    return a set of visible lines, a set of intersection points;
```

Analysis

Analysis and Proof

When we divide our problem into sub-problems so finally those problems will divided in our base cases. At time of merging we need to consider slope and intersection point.

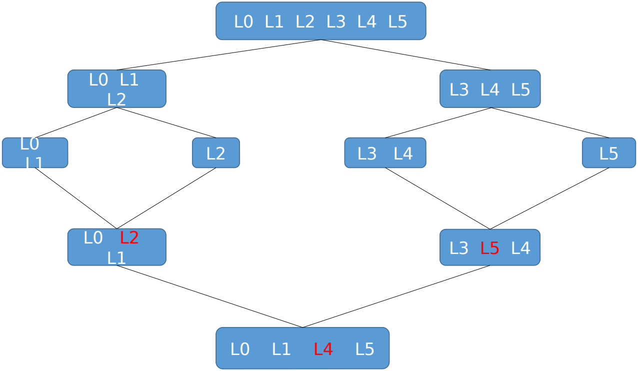
1. Suppose our time complexity is $T(n)$
2. time complexity for sub-solution is $T(n/2)$
3. in the algo we're dividing problem to 2 sub-problems which is $T(n/2)+T(n/2) = 2T(n/2)$
4. when we merge sub-problems which is $O(n)$
5. so finally our recurrence relation is $T(n) = 2 * T(n/2) + O(n)$
6. Time complexity $T(n) = 2 * T(n/2) + O(n)$ which is $O(nlog_2 n)$

Example

Sample Input

Line No.	x1	y1	x2	y2	m	c	equation
Lo	0	6	6	0	-1	6	Y = -x + 6
L1	0	6	-4	0	1.5	6	Y = 1.5x + 6
L2	0	3	3	0	-1	3	Y = -x + 3
L3	0	2	-5	0	0.4	2	Y = 0.4x + 2
L4	0	-6	8	0	0.75	-6	Y = 0.75x – 6
L5	0	8	-11	0	0.72	8	Y = 0.72 + 8

Example


L0 L1 L2 L3 L4 L5
L0 L1
L2
L3 L4 L5
L0
L1
L2
L3 L4
L5
L0 L2
L1
L3 L5 L4
L0 L1 L4 L5

Demo

- Woody : This time I will appear on screen.
- Buzz : This is my time I will appear.
- Producer : shhhhhhh....... ! Stop ! Let me see !

Demo


Problem solved!
Now You can see us.
... to infinity and beyond!

Contributions

Akash Banchhor(P18CO011)	Problem Study Implementation Analysis
Dilip Puri(P18CO008)	Problem Study Implementation Analysis, Presentation
Nishant Singh(P18CO012)	Problem Study Test Cases Presentation

Timeline

13-14 Aug	**Problem Study**
15-16 Aug	**Algorithm Making**
17-21 Aug	**Implementation** **Algorithm changes**
22-24 Aug	**Analysis and Report**
25-27 Aug	**Final report and presentation**