Final Report of CSE 590 Project: Collision Detection of 3D Printer

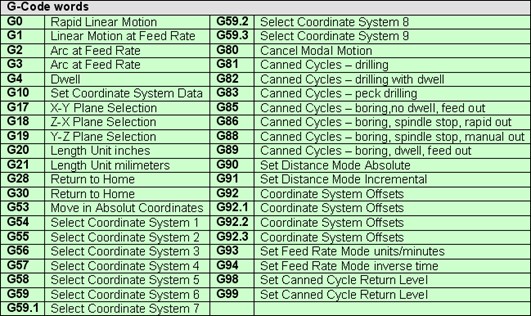
Gautam Othyoth Ganapathiyadan (50208358)

**Abstract—3D Printing or Additive Manufacturing is a CAD/CAM Manufacturing process by which nozzle movement is controlled to shape objects. The programming language used to control the nozzle is the G code. However malicious G code can cause collision of the printer nozzle and printed object and this can cause damage. In this project, we design an algorithm to detect such malicious G codes. The Verilog program designed will take in a set of 2 3D points - derived from the G code written. A 3D line is made using these points and they are compared with other lines synthesized from earlier points to check for intersection. This project aims to provide a comprehensive check for malicious gcodes by detecting the intersecting lines if any.**

1. PROJECT BACKGROUND AND INTRODUCTION

Additive Manufacturing which is also known as 3D printing is a mainstream manufacturing process in many industries. The printer nozzle is controlled to build a 3D physical object in layers. The 3D printer takes a set of instructions on how to move, where to move and what path to follow to shape the raw piece and drop the final 3D printed material.

3D printing is done with a programming language called G code. A list of sample G-codes are given in Fig1. These G codes are used to move the nozzle to shape the raw piece.



**Fig. 1.:** G-code commands.

If a gcode entered can cause the printer nozzle to potentially collide with the printed object during operating, this can damage the 3D printer. The aim of this project is to successfully formulate a program which can identify such malicious g codes and prevent the damage from happening.

This project concentrates on validation of the gcodes entered. The program employed checks if the printer nozzle collides with the printed object. The program takes a set of 2 3D points and constructs a line using those points. This line is checked for intersection with all the lines synthesized from previous inputs. The intersection check is not performed with previous inputs which have been found as malicious.

The following actions are performed based on the result of the algorithm -

1. **Line is malicious** – The LineID of this input is noted and this line is stored in the memory array but with malicious flag as 1.

2. **Line is not malicious** – The Line is stored in the memory array with malicious flag 0.

After all the inputs are taken, All the malicious LineIDs are printed in result.txt file.

The input to the program are stored in gcode.txt in the following format -

X1 Y1 Z1 X2 Y2 Z2.

Eg : 0 0 2 5 3 2

where X1,Y1,Z1 are the x,y and z coordinates of the first 3D point and X2, Y2, Z2 are the x,y and z coordinates of the second 3D point.

The output of the program will be the list of LineIDs of all the malicious inputs.

The algorithm used to detect malicious gcodes are Line Segment Intersection Method.

This program can then be loaded into an FPGA board where we can use this to validate any colliding inputs for the gcodes written in a 3D printer.

1. DESCRIPTION OF DESIGN METHOD

The Design method used for detecting malicious gcodes is Line Intersection method. A line is constructed from the 2 3D points, This line is compared with the lines constructed from the previous inputs to check for intersection.

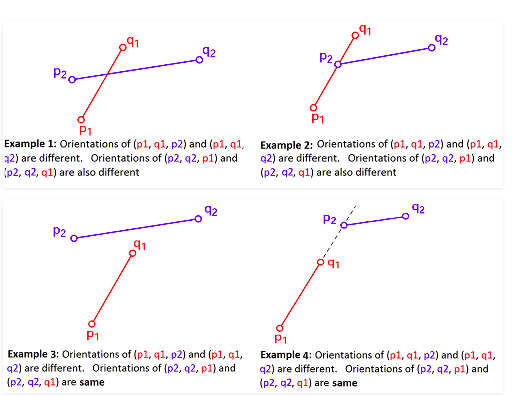
Line Segment Intersection Algorithm -

For 2D line with (p1,q1) and (p2,q2) as 2 end points, we use the regular line intersection algorithm which is as below -

2 Line Segments intersect if and only if one of the following 2 conditions are verified -

Case 1: (p1,q1,p2) and (p1,q1,q2) have different slopes and

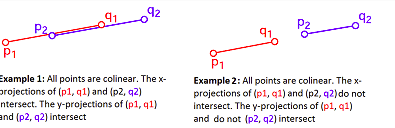
(p2,q2,p1) and (p2,q2,q1) have different slopes.



Case2: (p1,11p2),(p1,q1,q2),(p2,q2,p1) and (p2,q2,p1) are all collinear and

1. The x-projections of (p1,q1) and (p2,q2) intersect

2. The y-projections of (p1,q1) and (p2,q2) intersect



For 3D, The method used for 2D is extended to 3D by projecting the 3D lines into X, Y and Z planes respectively.

So the 2D line segment check algorithm is run 3 times with inputs (x,y), (y,z) and (x,z) respectively for Z, X and Y plane projections to check if the line segments intersect in 3D. Only if the line segment intersects in all planes will it intersect in 3D. Hence only if all 3 runs give a positive will we deem 2 line segments as intersecting.

Inputs for X plane – (y1,z1) and (y2,z2)

Inputs for Y plane - (x1,z1) and (x2,z2)

Inputs for Z plane - (x1,y1) and (x2,y2)

Exception Handling for 3D Line Projection into 3 planes -

The above 3D Line Segment Intersection method by projecting the 3D point can have certain exceptional cases where this check fails, so we incorporate some exceptional cases in our program to handle them.

1. False Negatives -

If 2 3D Line segments intersect they have to intersect in all 3 planes. There can be no exceptions in this case. Hence, this algorithm will not give any false negatives. So all malicious lines will be caught, there is no need to handle any exceptions here.

2.False Positives -

There can be 2 lines oriented in a particular way where they don’t intersect but intersect in all 3 planes, this can give rise to false positives. These exceptions can be handled by including 1 more check for line segments that are found to be intersecting by the projection method.

We use vector algebra to check if we get a solution for ‘a’ in the below equation to check if the lines intersect-

a(V1 X V2) = (P2-P1)XV2

where V1 is the Directional Vector of 3D line1, V2 is the Directional Vector of 3D line2. P1 is the first point of the line1 and P2 is the first point of line2.

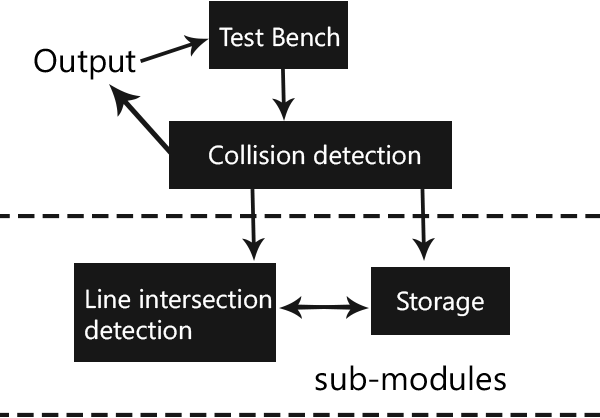
We perform Vector division to see if the scalar a gives a unique solution.

If a gives a unique solution then that means the lines intersect.

A combination of 2D line segment intersection algorithm, 3D line segment projection into 3 planes and Vector Algebra method to check for 3D line intersection.

1. VERILOG SOFTWARE FLOW CHART AND DESCRIPTION

Given below is the flowchart of verilog design for the 3D Line Segment Intersection Algorithm.



**Fig. 2.: FlowChart of Verilog Design**

We are implementing the Line Segment intersection method to detect malicious points. Line segments made from incoming points are stored in a memory array. Each incoming set of points are compared with previous points to find out malicious lines.

Detailed module design is as given below -

1. TEST BENCH – The Test Bench reads the gcode.txt file for each clock cycle to get the coordinates – x1, y1, z1, x2, y2, z2. These coordinates are passed as input to the underlying module – Collision Detect. It then gets the wire output values - out\_val, lineID from the Collision Detect Module. The Simulation to view the results is run by the TEST BENCH where we can see the corresponding out\_val for the in\_val. The out\_val will be 1 if the line segment is a malicious one, that is if it intersects with any of the previous line segments. The Test Bench also outputs a list of the LineIDs of all the malicious inputs in a text called results.txt
2. COLLISION DETECTION – The Collision Detection module takes in the coordinates as the input from the Test Bench. It primarily has 2 functions –
   1. Store the line segment formed using the coordinates in a memory module. This is implemented in the Storage Sub - Module
   2. Check if any line segments made using the incoming coordinates intersect with the ones stored in the memory module. This is implemented in the Line intersection detection module.
3. SUB-MODULES –
   1. Storage – It stores the coordinates that make the line segments that come as input from the Test Bench.
   2. Line Intersection Detection – It compares if the line segment made from the incoming coordinates intersect with the line segments in the Storage Sub Module. It sets the out\_val to 1 if such a case is detected.

Division Operation – A custom division operation is used as opposed to normal division to render the code synthesizable.

The Verilog logic used is as below -

p1= 0;

for(i=0;i <32;i=i+1) begin

p1 = {p1[30:0],a1[31]};

a1[31:1] = a1[30:0];

p1 = p1-b1;

if(p1[31] == 1) begin

a1[0] = 0;

p1 = p1 + b1; end

else

a1[0] = 1;

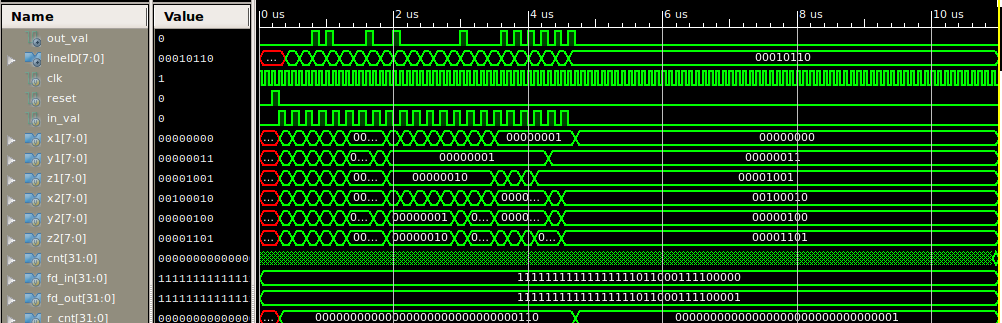
end

where a1 and b1 are inputs to be divided and the result is finally stored in a1.

1. SIMULATION RESULT

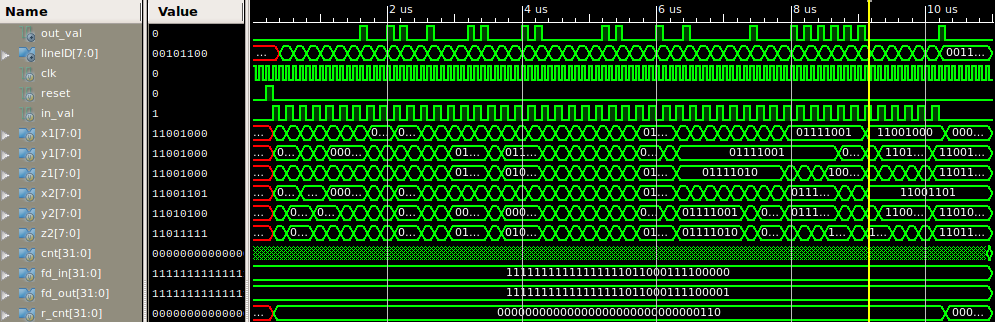
We look at some of the Simulation results for the various input conditions for analysis -

1. Simulation for the 22 values given as a part of gcode\_3d.txt as provided by the TAs -



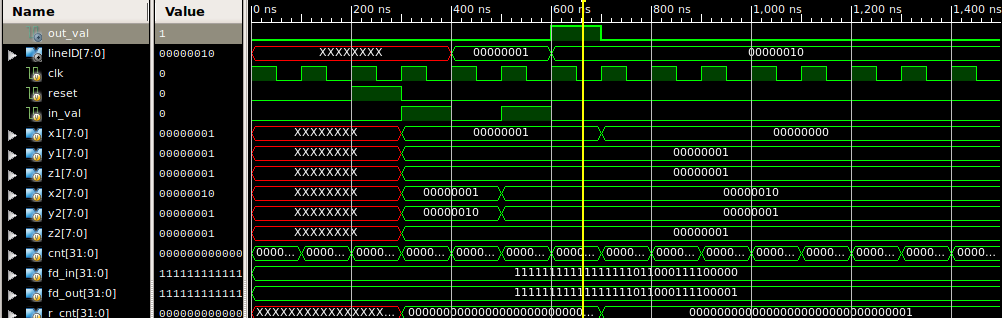
Here we can see that the out\_val is becoming 1 for lines that are intersecting with the previous non-malicious lines. A New LineID is generated every time an in\_val comes in with an input of a pair of 6 points. This is then checked with all the previous lines and seen if it is malicious or not. The lineIDs which have 0 as out\_val are non-malicious.

2. Simulation for the 50 values given as a part of gcode\_3d\_2.txt as provided by the TAs -



The program can only take 50 inputs at a time. Lines which intersect and are malicious have out\_val as 1.

3. Simulation for Boundary Value Cases – Lines that intersect as L.

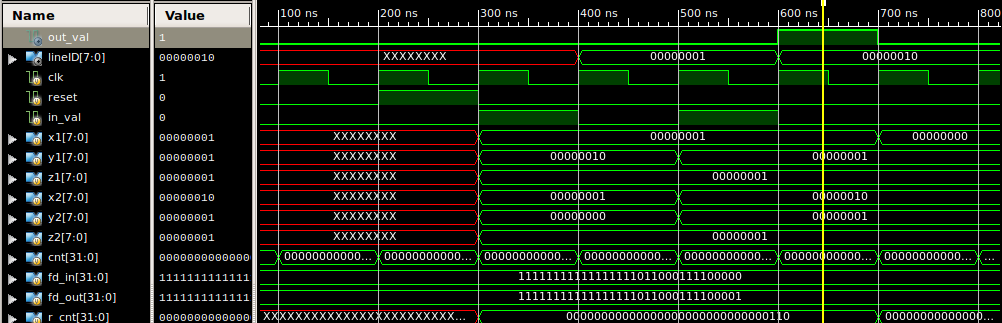


Here we see the case where the lines intersect as L. The out\_val in this case is 1 meaning the lines intersect. The input used here is -

1 1 1 1 2 1

1 1 1 2 1 1

4. Simulation for Boundary Value Cases – Lines that intersect as T.

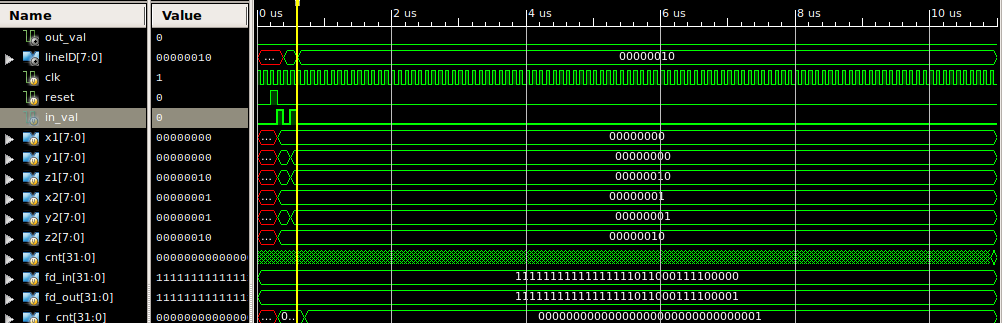


Here we see the case where the lines intersect as T. The out\_val in this case is 1 meaning the lines intersect. The input used here is -

1 2 1 1 0 1

1 1 1 2 1 1

5. Exceptional Handling Case – Lines that don’t intersect but intersect in 3D Line Projection -

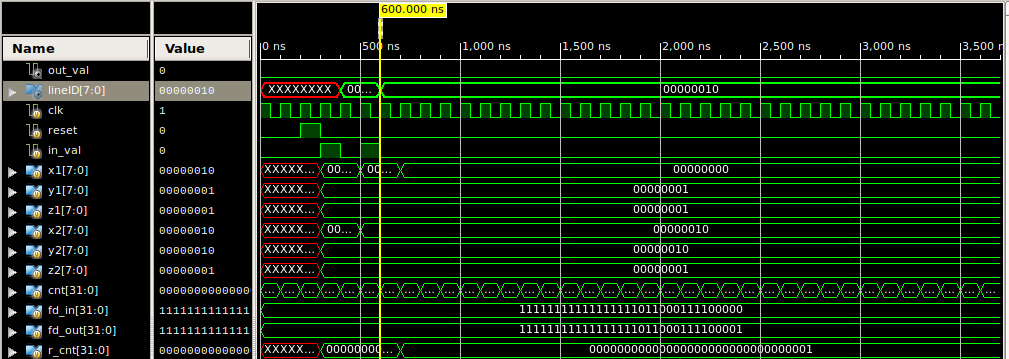


Here we see that the out\_val is 0 as the lines don’t intersect. Although this passes the Line Segment Intersection Algorithm, it is found to be non intersecting at cross-product vector algebra check. The input used is -

0 1 0 1 0 2

0 0 2 1 1 2

6. Simulation with Parallel Lines -

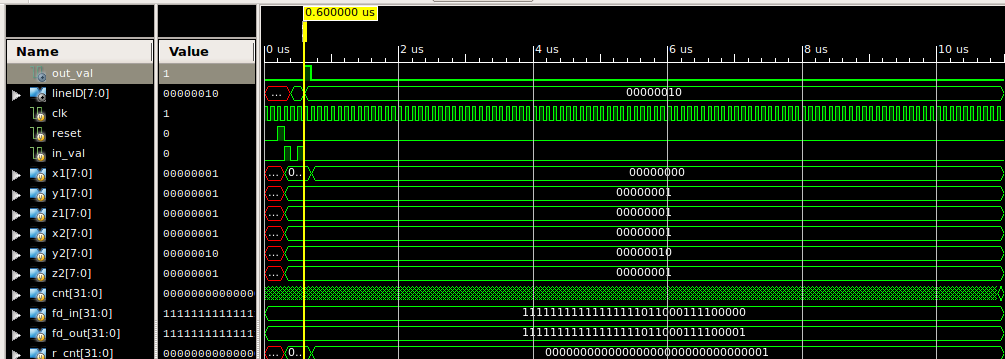


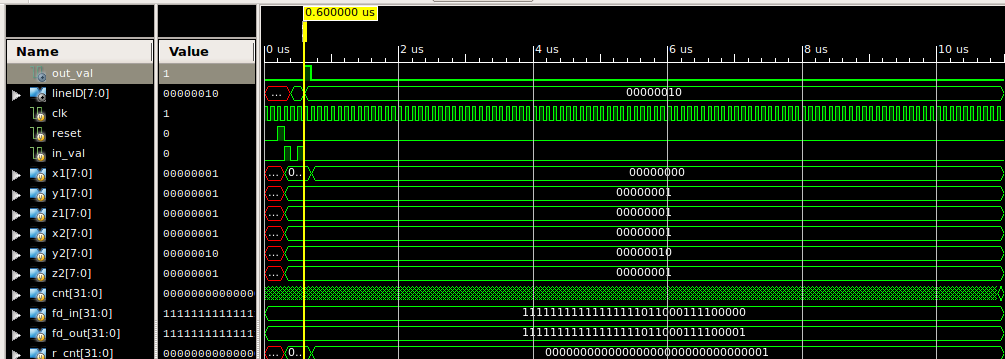
Here we see that the lines don’t intersect as they are parallel. The input used are -

1 1 1 1 2 1

2 1 1 2 2 1

7. Simulation with duplicate lines -





Here the lines are the same, so they are shown as malicious with out\_val as 1. The input used is -

1 1 1 1 2 1

1 1 1 1 2 1

1. DISCUSSION

3D printing is a very popular manufacturing process in a lot of industries. Gcode is the programming language

that is used to perform 3D printing.

One of the main challenges with 3D printing is malicious gcode causing the Printer Nozzle to collide with the printer object. This can cause damage to the printer.

This project is aimed at eliminating the above collision problem by designing a 3D line intersection algorithm to check if the line segments intersect and find the malicious gcodes.

The malicious inputs can be found through line segment intersection algorithm in 3D. The malicious inputs are found and are printed in a separate file.

Verilog is used to perform this program and the simulations are analyzed to observe the results of the program.

The simulation is run by the testbench which in turn calls the collision detect module.

The code is synthesizable and can be used in an FPGA board.

REFERENCES

[1] Sungsuk Kim and Sun Ok Yang. Transforming algorithm of 3d model data into g-code for 3d printers in distributed systems. In *International Conference on Computer Science and its Applications*, pages 1074–1078. Springer, 2016.

[2] <http://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/>

[3] <http://mathforum.org/library/drmath/view/62814.html>

[4] https://www.wikipedia.org