Student Project:

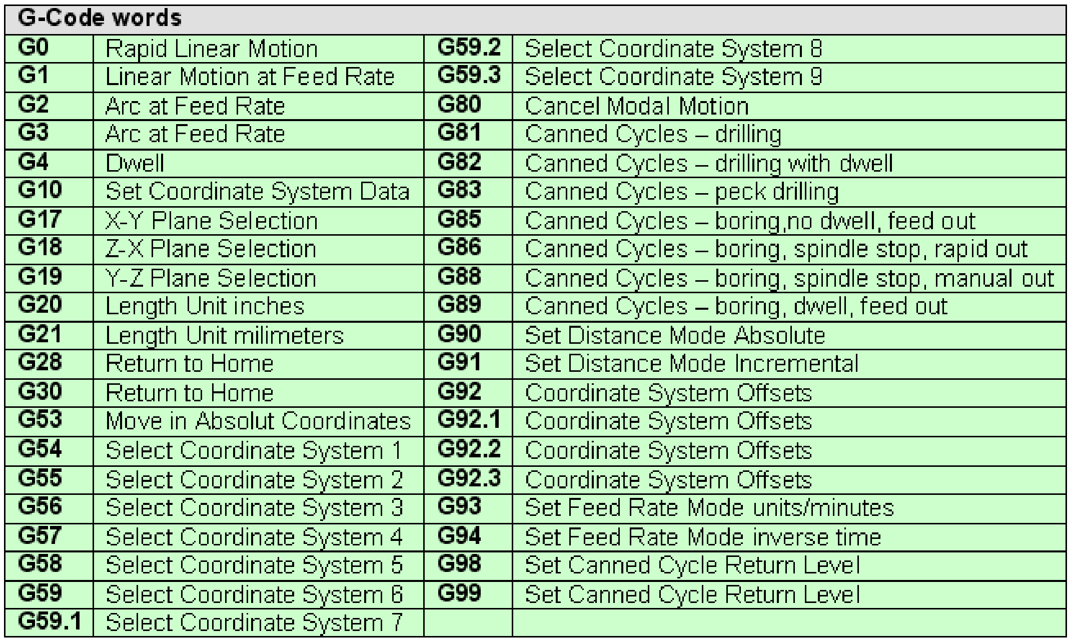
FPGA Implementation of Collision Detection for 3D Printers

1. Introduction

Additive manufacturing (AM), also known as 3D printing, has been becoming a mainstream manufacturing process in various industry fields. Specifically, it refers to a process by which 3D digital design data (in the cyber domain) is used to build up a 3D physical object in layers by depositing material (in the physical domain). The G-code is the programming language to control the nozzle movement of a 3D printer. However, some malicious G-code may induce collision between the printer nozzle and printed object on the platform to damage the printer. In this project, we aim to detect all the malicious G-code in the given printing files.

2. Background of G-code:

G-code is a language in which people tell computerized machine tools how to make something. The "how" is defined by instructions on where to move, how fast to move, and what path to follow. The most common situation is that, within a machine tool, an adding tool (nozzle) is moved according to these instructions through a toolpath and drops the material to leave only the finished workpiece. (From WiKi)



G-Code words			
G0	Rapid Linear Motion	**G59.2**	Select Coordinate System 8
G1	Linear Motion at Feed Rate	**G59.3**	Select Coordinate System 9
G2	Arc at Feed Rate	**G80**	Cancel Modal Motion
G3	Arc at Feed Rate	**G81**	Canned Cycles – drilling
G4	Dwell	**G82**	Canned Cycles – drilling with dwell
G10	Set Coordinate System Data	**G83**	Canned Cycles – peck drilling
G17	X-Y Plane Selection	**G85**	Canned Cycles – boring,no dwell, feed out
G18	Z-X Plane Selection	**G86**	Canned Cycles – boring, spindle stop, rapid out
G19	Y-Z Plane Selection	**G88**	Canned Cycles – boring, spindle stop, manual out
G20	Length Unit inches	**G89**	Canned Cycles – boring, dwell, feed out
G21	Length Unit milimeters	**G90**	Set Distance Mode Absolute
G28	Return to Home	**G91**	Set Distance Mode Incremental
G30	Return to Home	**G92**	Coordinate System Offsets
G53	Move in Absolut Coordinates	**G92.1**	Coordinate System Offsets
G54	Select Coordinate System 1	**G92.2**	Coordinate System Offsets
G55	Select Coordinate System 2	**G92.3**	Coordinate System Offsets
G56	Select Coordinate System 3	**G93**	Set Feed Rate Mode units/minutes
G57	Select Coordinate System 4	**G94**	Set Feed Rate Mode inverse time
G58	Select Coordinate System 5	**G98**	Set Canned Cycle Return Level
G59	Select Coordinate System 6	**G99**	Set Canned Cycle Return Level
G59.1	Select Coordinate System 7		

3. Input and Output

Input: The input is a file with a series of starting and ending (3D) voxel coordinate pairs, which provides a series of paths for the nozzle in the G-code. We will use integers for the voxel coordinates. We only ask for 2D, layer by layer collision detection for the project, and the 3D version as a bonus. Because in 3D space there would be more exceptions and it is slightly more complicated.

Example : pair1 (X_s,Y_s,Z),(X_e,Y_e,Z): (0,0,2),(5,3,2).

We guarantee the line segments are partitioned by layers and given by the ascending order.

We will prepare two set of these files, one for testing/debugging and the other for grading.

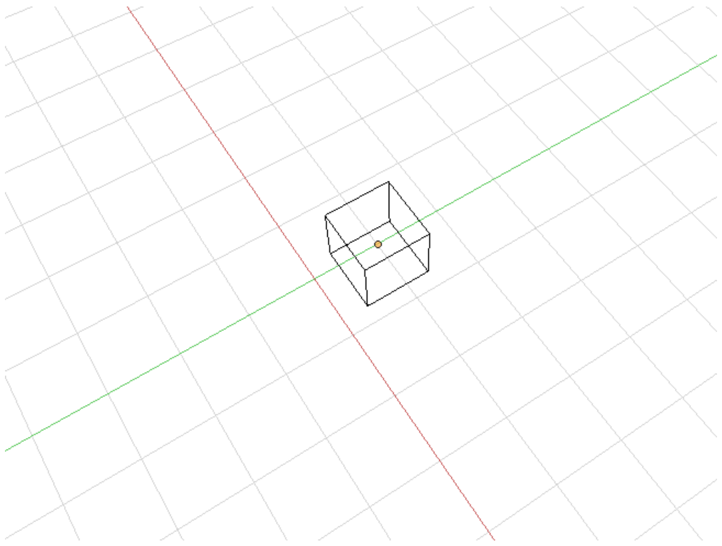
Output: Line index number (starting from 1) of all the malicious voxel coordinate pairs.

4. Verilog Design Flow

We illustrate two methods of Verilog design, voxel marking, and line segment interaction. You can choose any one to design your project.

Method 1) -Voxels Marking

This method is based on voxel concept (https://en.wikipedia.org/wiki/Voxel), which is shown in the following picture. We can record the voxels along the nozzle path which is defined by G-code. When encountering a malicious G-code command, drop it and continue the next operation.

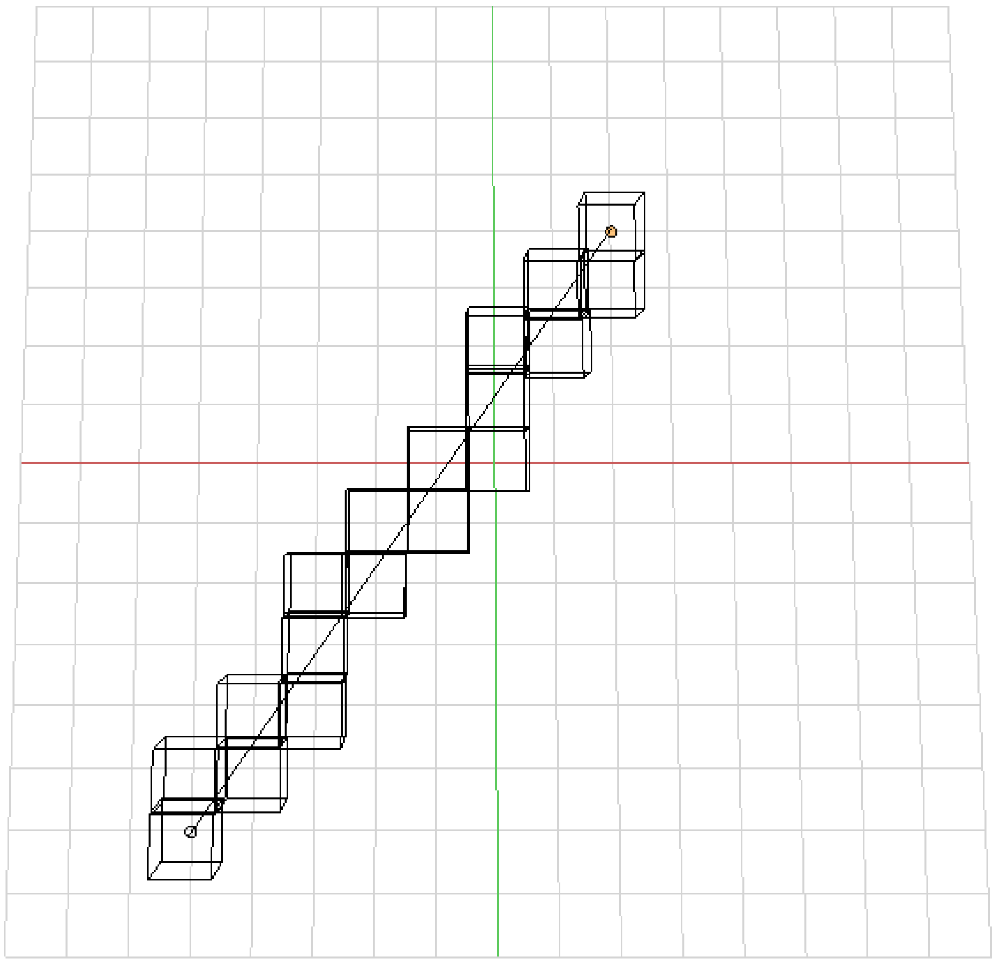


Note: Given a voxel coordinate (X,Y,Z) located at the center of said voxel. The voxel occupies the cubic space (X ±0.5, Y ±0.5, Z ±0.5).

The whole method is as the following:

Step 1: Design a memory module to record each voxel state. For example a 4x4x4 3D space would need 64 bits of memory to record.

Step 2: Initialize all voxels in the space as “0”. Feed the voxel coordinate pairs (6 integers) into the Verilog module. For the first segment, design your algorithm to find all the voxels which intersects with the line segment and set their state as “1”. An example is shown in the following figure.



Step 3: Continue to read other line segments sequentially into your design. If any voxel along the current line segment is already set as “1”, there will be a collision happening. **Don’t change any voxel state** and output this line

segment index number as a malicious one. Otherwise, update all the voxel states which intersects with the line segment.

Step 4: Repeat step 3 until there is no line segment left.

Method 2) -Line Segment Intersection

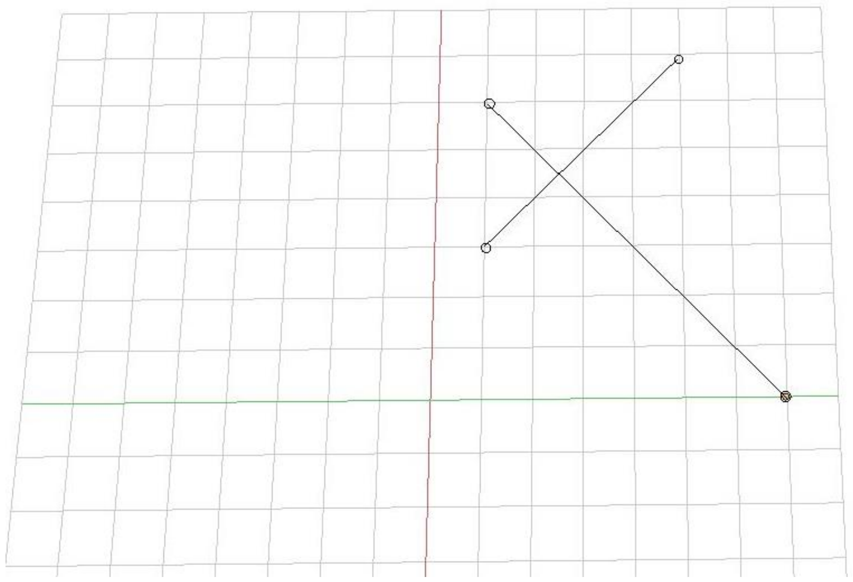
Step 1: Design a memory module to store the voxel coordinate pairs for each line segment on the same layer. Set current layer height as Z=0.

Step 2: Read one voxel coordinate pair (6 integers) into your Verilog module.

- If the layer height of this line segment is the same as Z, determine whether the current line segment has intersection point with other line segments whose layer height is also equal to Z (Hint: use cross product, refer to the link below). If there is intersection, output the line segment index number as malicious G-code. Otherwise, store the current line segment into memory.
- If the layer height is larger than Z, it means another layer is starting. Update Z to current layer height and clear the memory module. Store the current line segment into memory.

http://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/

Step 3: Repeat step 2 until there is no line segment left.



Note: feel free to design other methods outside of the above frameworks.

5. Evaluating Criterion

There are three checkpoints of this project.

a. **First four weeks (until 3/27):**
In the first checking, determine the proper Verilog module which would operates (either by voxels marking or line intersection, see the design section) on an incoming voxel coordinate pairs.

b. **Another four weeks (until 4/24):**
In the second checking, determine the proper Verilog module which would distinguish whether an incoming voxel coordinate pair is malicious.

c. **Another two weeks (until 5/8):**
In the third checking, determine the proper Verilog implementation which would leverage on the above two modules and return the output file.

We will compare the ground truth with the student’s results

6. Grading

a. Functionality of module design in each checkpoint. (50%)
b. A written report for the project. (50%)
c. Bonus (waive the final exam): Operate on the full 3D space, the voxel coordinate pairs would be given in 3D.
d. 1-2 selected groups will attend the IOT competition “FPGA for Internet of things” on behalf of UB: http://www.cse.cuhk.edu.hk/~byu/dac-iot/; They will get A in CSE 490/590.

Appendix:

Verilog top module:

```
module CollisionDetect (clk, reset, in_val, x1,y1,z1,x2,y2,z2, out_val, lineID)

input clk, reset, in_val;

input [7:0] x1, y1, z1, x2, y2, z2;

output out_val;

output [7:0] lineID;

/* Implementation your collision detection here*/


endmodule
```

Parameter Description:

clk [1 bit]: clock waveform of the design

reset [1 bit]: clear the design before program running

in_val [1 bit]: indicator of validity for input data, “1” is ready, “0” is not ready

x1 [8 bit]: unsigned type, position on x axis of starting point of segment

y1 [8 bit]: unsigned type, position on y axis of starting point of segment

z1 [8 bit]: unsigned type, position on z axis of starting point of segment

x2 [8 bit]: unsigned type, position on x axis of ending point of segment

y2 [8 bit]: unsigned type, position on y axis of ending point of segment

z2 [8 bit]: unsigned type, position on z axis of ending point of segment

out_val [1 bit]: indicator of validity for output data, “1” is ready, “0” is not ready

lineID [8 bit]: the segment index number of collision code