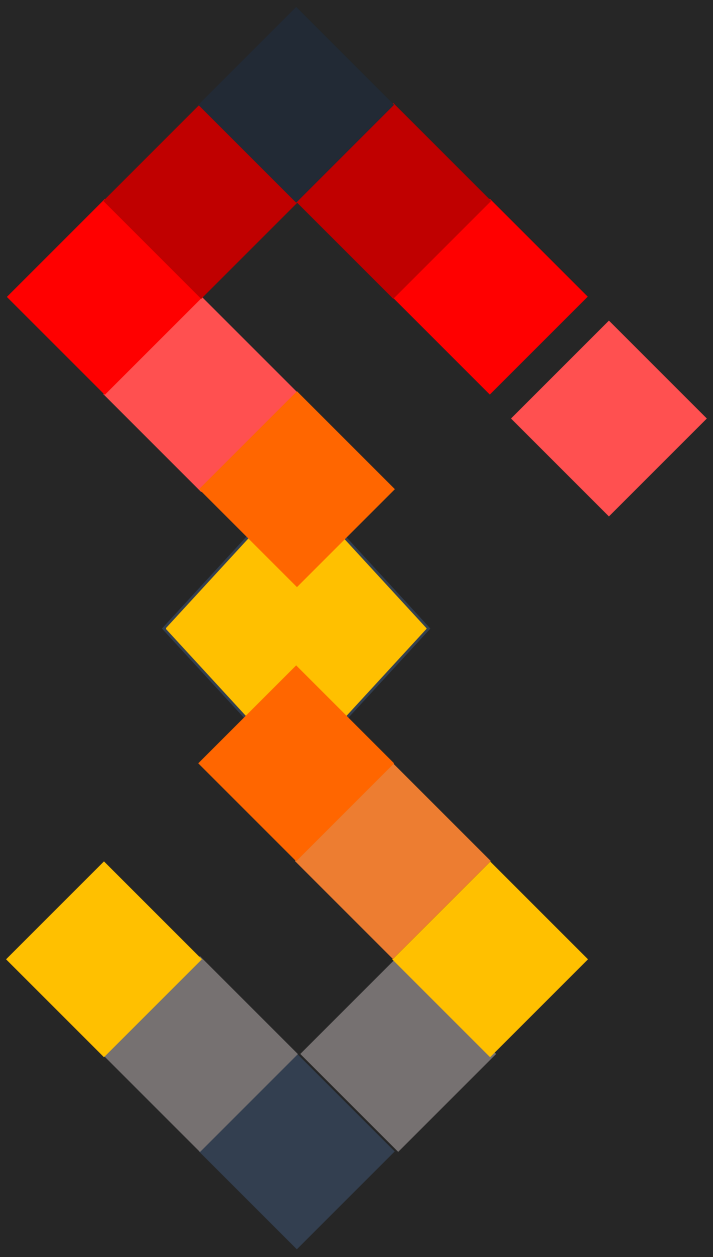


Slicer

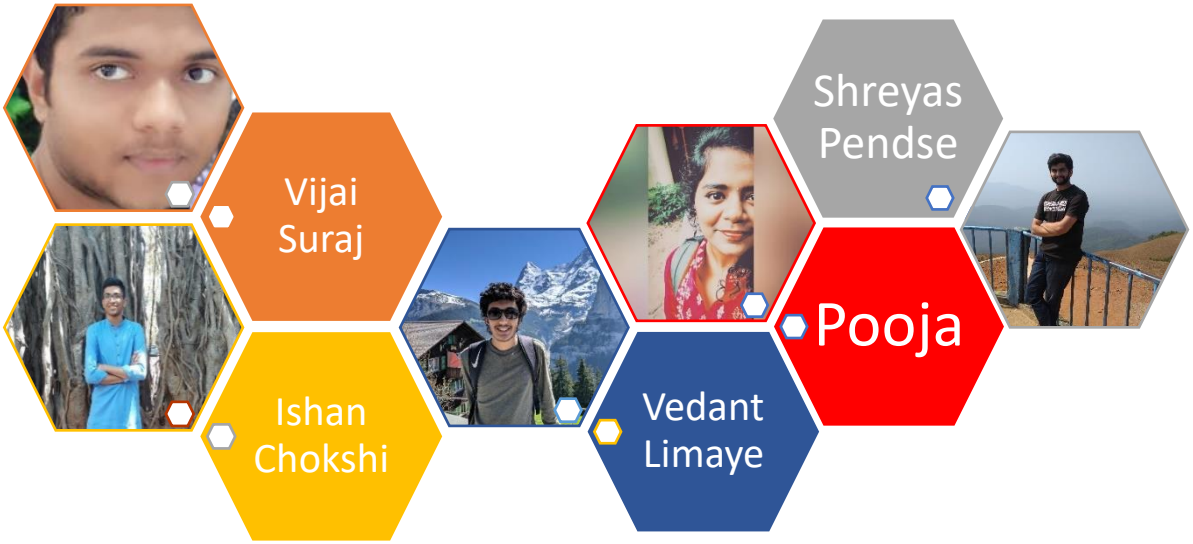
report

Slicer

Project Report 2020-2021

*3D*Printing Club

Center For Innovation
Indian Institute of Technology
Madras



EP19B009
vijaisuraj2001@gmail.com
Born in Chennai
Has lived in 4 cities and studied in 8 schools
Loves to game and watch anime

ME19B047
shreyaspendse01@gmail.com
Born in Pune
Fond of cricket and football
Follows new tech

BE19B018
ishan.c1671@gmail.com
Loves playing football and basketball
Loves to sing too

CE19B035
addagatlapooja@gmail.com
From Telangana
Hobbies are dancing, listening to music
Enjoys editing pictures

ME19B185
vslvedant@gmail.com
1st year mech student
Comes from Pune
Plays TT and swims in Insti
Happy to be part of CFI

Table of Contents

Overview of the Project

Slicer is a software that converts the 3D object into specific instructions for the printer. The slicer converts .stl or .amf mesh file into g-code.
This is a learning-based project where the end goal is to explore all possible features of a slicer. We aim to achieve this, and more, by developing our own slicer. Later on, we will work on the shortcomings of available slicers in the market and make our slicer better.

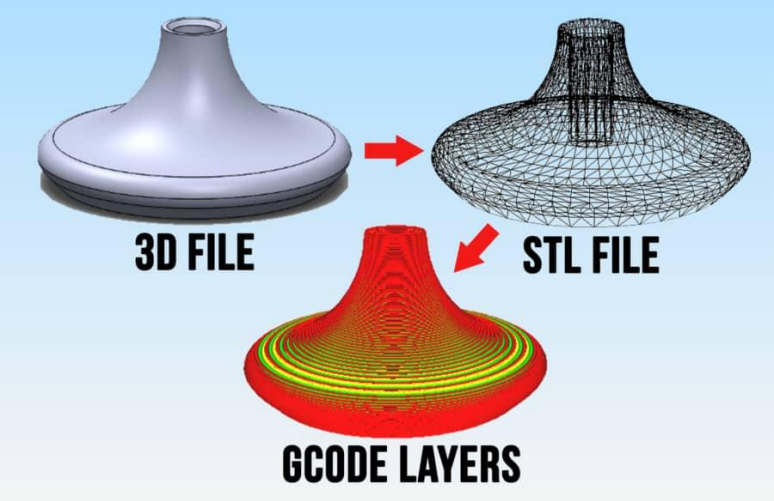
Learning Objectives

- In depth understanding of a slicer
- Extracting info and parameters from sliced layers
- Try out UI platforms to test out codes to extract these parameters
- Get familiar with the background work of a slicer like importing .stl files, print settings, generation of G-code file.
- Go through GitHub repositories of Cura and Slic3r to get a fair idea of how the backend code for a slicer is made.
- Develop our own slicer
- Implement common features into our slicer
- Test it against some of the popular slicers available and improvise our slicer

Background Work Done

What do Slicers do?

3D slicer programs, sometimes just called slicers, convert digital 3D files (STL file) into G-code which 3D printers understand. They calculate a path for the nozzle to follow based on the part and the user-specified settings.



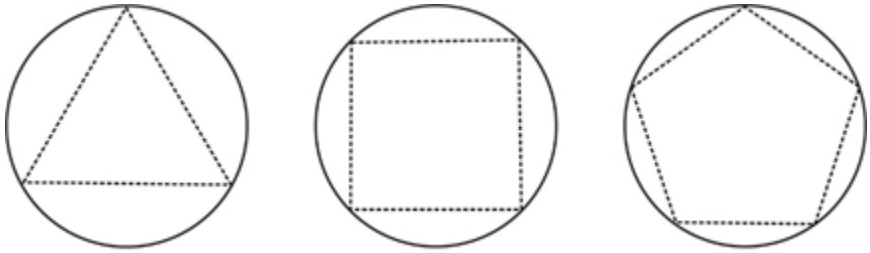
3D model of our object is created with the assistance of a CAD software, which is saved as an STL file. This STL file is then converted into G Code by a slicer.

How does a Slicer work?

Before how a slicer works is understood, we'll have to first understand how a model is stored in an STL file.
The main purpose of saving our model as a STL file, and not in any other format, is to encode the surface geometry of the object, by limiting the detail to manageable amounts. This is achieved by a simple concept called Tessellation.

Tessellation

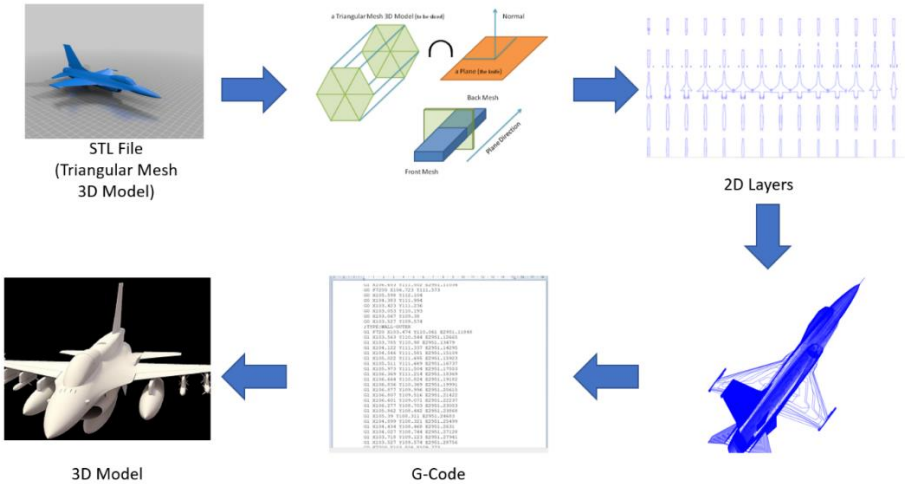
Tessellation is the process of tiling a surface with one or more geometric shapes such that there are no gaps or overlaps.
Tessellation is done in order to approximate the curves and geometry of the 3D model, since the nozzle of 3D printers can't travel in curved paths and can only travel in straight lines.
The level of tessellation determines the accuracy of the print.



As the order of Tessellation increases, the number of triangles used in the approximation increases, and the accuracy of the approximation increases as well. A regular infinite sided polygon (with infinite number of triangles) is the best approximation of a circle.

Slicing

In our case, tessellation of the surface of our model is used to capture the geometry of our model. The vertices of the triangles and the components of the unit normal of the of those triangles, also called the facets, is stored in the STL file.
This process is what has made 3D printing possible. Most models have very complex curves, but by using tessellation, we can adjust the number of facets used to define the same geometry, thus approximating the curve to varying complexities. Lowering the complexity is key to achieving 3D printing.



This image provides a brief overview of the steps involved in slicing a 3D model and generate g code.

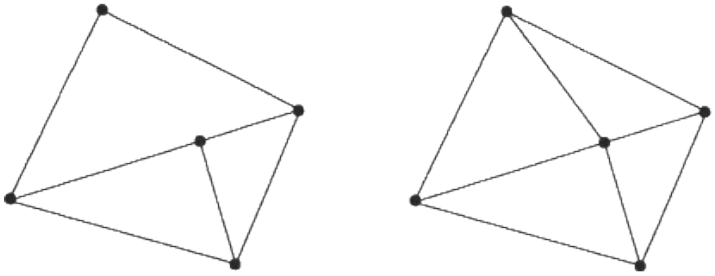
The next step of the process, after generating the STL file, is slicing the object into layers. This is achieved by slicing the triangles in the same plane. If the triangle and the slice plane intersect a line segment is formed by the triangle and the slice plane. The intersections are then converted into continuous contours or curves. This is then arranged in a 2D matrix and a g-code file is generated as per the matrix. This file contains the instruction for the 3D printer.

Rules for Slicing STL

There are special rules for tessellation and storing information for STL files. Below are some of them.

Vertex-Vertex rule

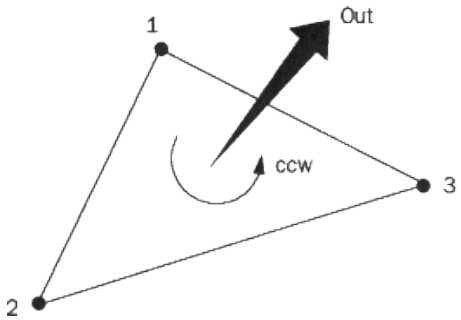
In STL formatting, this rule states that each triangle must share 2 vertices with its neighboring triangles. This means that a vertex of one triangle must not lie on the side of another triangle.



Vertex-Vertex rule violation(left).
Each triangle is sharing 2 vertices with each of its adjacent triangles(right).

Orientation rule

This rule states that the orientation of the facet must be defined clearly. This orientation is specified in two ways. First, the direction of normal vector should point outwards. Moreover, the vertices should be listed in counter-clockwise order when looking at the object from the outside (or right-hand rule).



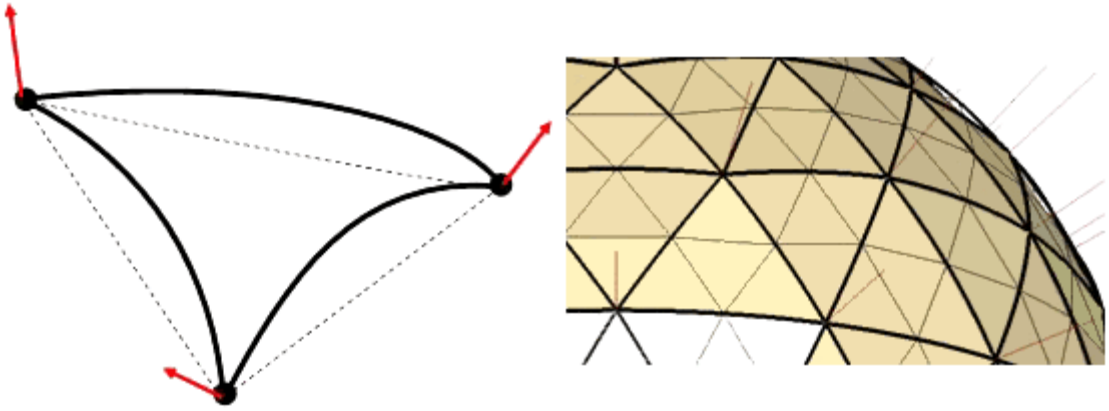
Each triangle in the STL file is specified with its normal vector along with its vertices listed in the counter-clockwise order.

All-positive Octant rule

This rule states that all the coordinates of the triangle vertices must be positive (nonnegative and nonzero). If the 3D part is allowed to lie anywhere in the coordinate space, then negative coordinates may occur. The problem is, when storing negative coordinates, one must use “signed floating pointing numbers”. These numbers may require an additional bit to store a sign (+/-). Hence, it is important to ensure that all coordinates are positive. When all the coordinates are positive, it also means that more space will be saved.

Other File Formats: AMF

Another popular choice of file format for 3D printers, was invented as STL is bloated, slow, error-prone, and incapable of storing color, material, and texture information.
It is an XML based format with native support for geometry, scale, color, materials, lattices, duplicates, and orientation, giving it superiority over STL when it comes to innovative printing.
It also uses similar algorithm to STL files but instead use curved triangles, making it easier to describe curved surfaces without using as many facets as STL format.



The AMF format uses curved triangles, which means precise encoding without sacrificing file size

The rest of the procedure are the same as STL file format: the layers are sliced by planes. The curves intersecting with the slicing plane gives the shape and geometry of the 3D object.

Print Settings

When using slicers, we can choose from a variety of print settings to adjust how your part is printed. Adjusting these settings can change the quality, strength, and total print time. Here are some of the most commonly available print settings.

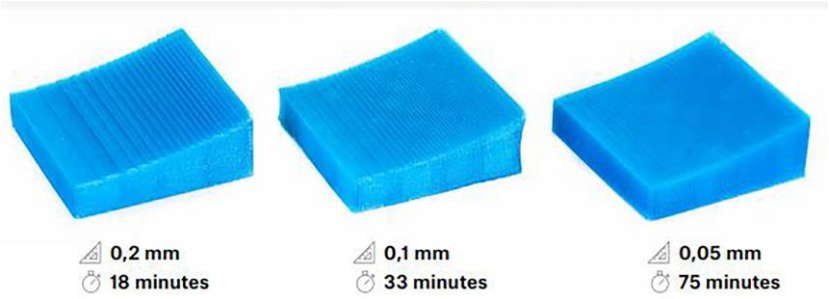
Nozzle and build plate temperature

These temperatures depend on the material being used for printing. Here are some of the common materials and their optimum temperatures.

Material	Nozzle Temperature	Build Plate Temperature
PLA	180 - 230 ^{o}C	20 - 60 ^{o}C
ABS	210 - 250 ^{o}C	80 - 110 ^{o}C
PETG	220 - 250 ^{o}C	50 - 75 ^{o}C
Nylon	240 - 260 ^{o}C	70 - 100 ^{o}C
TPU	210 - 230 ^{o}C	30 - 60 ^{o}C

Layer height

It adjusts how thick each layer is. Typically layer height ranges from 0.1 - 0.3 mm, with 0.2 mm being standard for most printers. Smaller layer results in smoother parts, but due to the increased number of layers, the print takes more time.



By optimizing layer height to suit the requirements it's possible to speed up the printing process.

Wall thickness

Also referred to as "shell thickness", this setting determines the strength of the print. Each shell thickness is equivalent to the nozzle width. More the number of shells, the stronger the print.

It can also depend on the material being used; to ensure usability.

MATERIAL THICKNESS RECOMMENDATIONS

MATERIAL	PLA	ABS	NYLON	VEROWHITE	TRANSPARENT	ABS-LIKE	RUBBER-LIKE	VISICLEAR
RECOMMENDED Minimum thickness (mm)	1.5	1.5	1.5	1.0	1.0	1.0	2.0	1.0
ABSOLUTE Minimum thickness (mm)	0.8	0.8	0.8	0.6	0.6	0.6	0.8	0.6

Infill percentage

Most 3D prints aren't hollow, they aren't solid either. The interior mostly contains patterns called infill that strengthen and support the layers above. The shape of print and infill percentage can be bumped up for additional reinforcement.
Infill patterns can be chosen according to the shell shape. The best patterns are those which are along the direction of the force.
Honeycomb is one of the most used shapes because although they are not always aligned along the force, they provide an equal amount of strength in all directions.

Flow

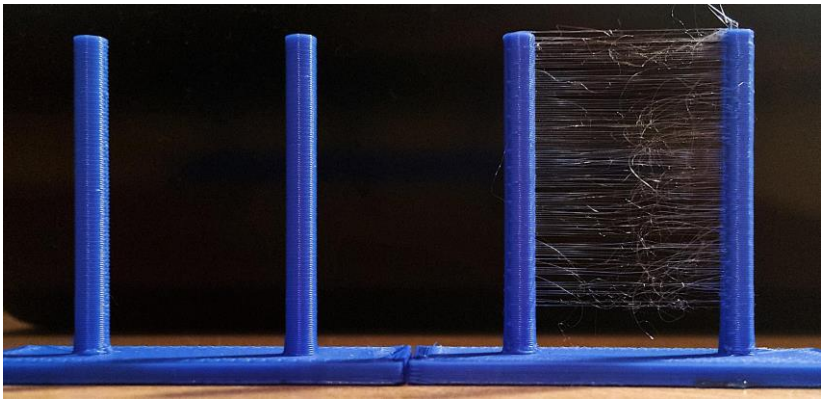
This option is used to change the rate at which the material is fed to the extruder. Typically, it is set at 100% but it can be changed according to the print quality and material. Over and under extrusions are common defects that arise due to faults in flow percentage.

Print Speed

Print speed is how fast the nozzle moves while it's printing. It can be adjusted according to the part being printed or the material used. Rigid plastics like PLA can handle higher speeds (60 mm/s or greater) but if flexible materials like TPU or TPE are used it's better to slow down the nozzle speed to 30 mm/s.

Retraction

Sometimes while printing, the nozzle may have to travel from one point to another without extruding any material. During this travel excess material can leak out of the nozzle creating unsightly string and blob. To prevent this, retraction reverses the extrusion motor slightly, relieving pressure on the molten material during the travel.



Enabling retraction will increase the print time slightly, but can drastically improve the print quality, making the extra time worth it. Retraction enabled and disabled on left-side and right-side print respectively.

Support Structure

Overhangings are a challenge for 3D printers because the plastic will droop down as soon as they leave the nozzle without a layer below. Enabling support generates an additional support structure that is printed at the same time as the part.
The process of removing support structures can leave behind imperfections, therefore it is advisable to take time and optimally tune the support settings.

This print depicts a poor choice of orientation and support structures, because by orienting this structure differently it's possible to completely eliminate the need for support structures.

Minimum layer time

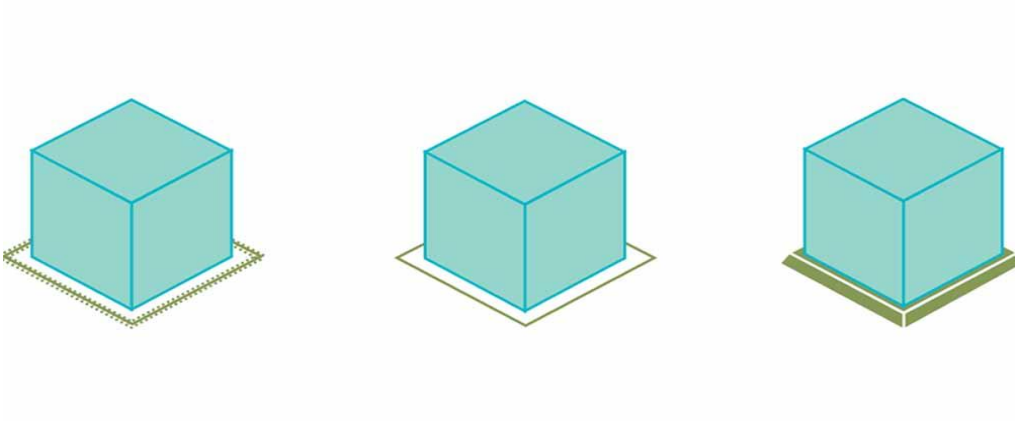
Instead of using fans, another way to improve cooling is increasing the minimum time for each layer. When printing small parts, the plastic may be extruded over a layer that hasn't cooled yet. By adjusting this setting, we can give the printed layers more time to dry.

Lift head

If enabled, the nozzle head is lifted away from the print if the minimum layer time hasn't been reached. This setting should be used if the layer being printed is extremely small, allowing it to be exposed to air flow

BUILD PLATE ADHESION

Sometimes the part may have problems sticking to the build plate. There are different ways to get the part sticking to the build plate like using PEI sheets or using adhesives, but slicers also have options which can help solve this problem. There are typically three options to choose from: skirt, brim and raft.



Skirt, brim and raft (from left to right)

Skirt

A skirt is a small length of material extruded right at the beginning of the print. This setting improves adhesion to the build plate by keeping the bottom surface consistent.

Brim

A brim can be used to improve adhesion of the print by modifying the first layer to increase the surface area. Brims can be difficult to remove so use only when necessary.

Raft

When the raft option is enabled, a platform is generated on top of which the part is printed. It is especially useful when the lowest layer is required to be smooth. It also uses up a lot of plastic and can drastically increase the print time. It should be noted that rafts reduce the effect of heat beds.

OTHER SETTINGS

Horizontal expansion

Once the molten material leaves the nozzle, it tends to expand slightly. This effect can be quite annoying if the parts have to be put together. For example, if a hole of diameter 10mm is printed, the hole may end up with diameter 9.6mm. Adding a horizontal expansion of -0.2mm will be negated.

Combing

Travel moves which don't extrude material can cause defects if it moves over a printed layer. By enabling combing, these intersections are made less visible because the nozzle will move over infill as much as possible. This setting will not increase the print by much but might significantly improve your print quality.

G Code

G-code essentially tells the nozzle where to go and how much plastic to extrude along the way. It takes many instructions to print along the way. A small part that takes 1 hour to print can have more than 100,000 lines of G-code.

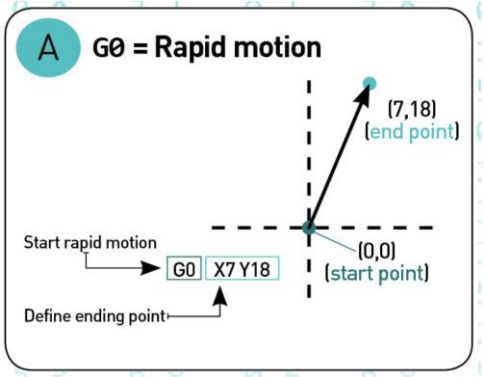
Startup G-code

Before you start a print, instructions for starting position, extruding some material before printing or any other actions can be given to improve the print quality of the first layer.

Ending G-code

When a print finishes, an instruction can be given to the 3D printer to follow routinely after every print. There should at least be codes to instruct the 3D printer to cool down the nozzle and build plate, as well as switching off the fans and motors. It is recommended that you lift the nozzle a few millimeters so that the nozzle doesn't get stuck to the print.

Some Important G Code Commands



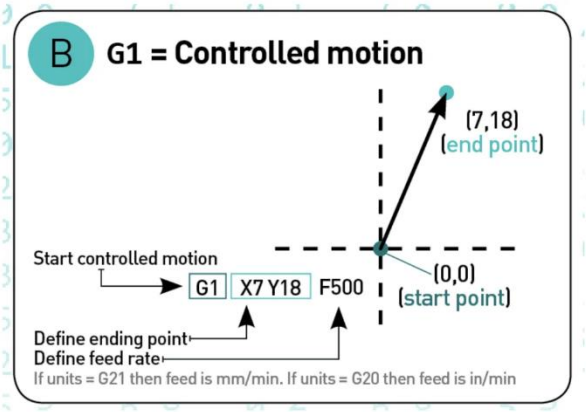
1. G0: rapid motion

The G0 command tells the print head to move at maximum travel speed from the current position to the coordinates specified by the command without extrusion of any material.

An example of G0 command. (image source: all3dp.com)

2. G1: controlled motion

The G1 command tells the print head to move at a specified speed from the current position to the coordinated specified by the G-code command. The speed is specified by the Feed rate parameter F. The head will move in a coordinated fashion such that both axes complete the travel simultaneously. The printer can extrude material while executing this G-code command with extrusion specified by the extrusion parameter E.



An example of G0 command. (image source: all3dp.com)

3. G17 / G18 / G19: Set Plane

These G-code commands set the plane in which the nozzle should move. Typically, G17 is the default for most machines and it denotes the X-Y plane. G18 denotes the Z-X plane and G19 denotes the Y-Z plane.

4. G20 / G21: Set Units

These G-code commands set the units. G20 denotes inches while G21 denotes millimetres.

```
G20

G0 X7 Y18
```

means "move rapidly to X=7 inches and Y=18 inches" while

```
G21

G0 X7 Y18
```

means "move rapidly to X=7 mm and Y=18 mm".

5. G28: Homing

A G28 command tells the machine to go to its home position. A home position can be defined by the G28.1 command as follows.

```
G28.1 X0 Y0 Z0
```

6. G90: Absolute mode

Absolute mode tells the machine to interpret coordinates as absolute coordinates. This means the G-code command below will send to x = 10.

```
G0 X10
```

7. G91: Relative mode

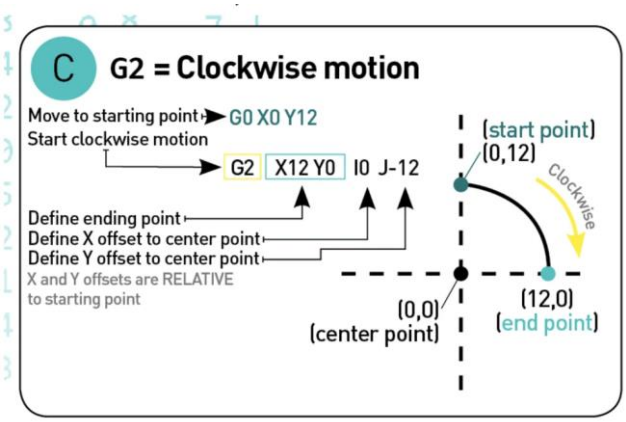
The relative mode is the opposite of the absolute mode. G91 tells the machine to interpret coordinates as relative coordinates. If the machine is currently at X=10, then the following G-code commands tell the machine to move 10 units in the X direction from its current position. At the end of the operation, the machine head will be located at X=20.

```
G91

G0 X10
```

8. G2: Clockwise motion

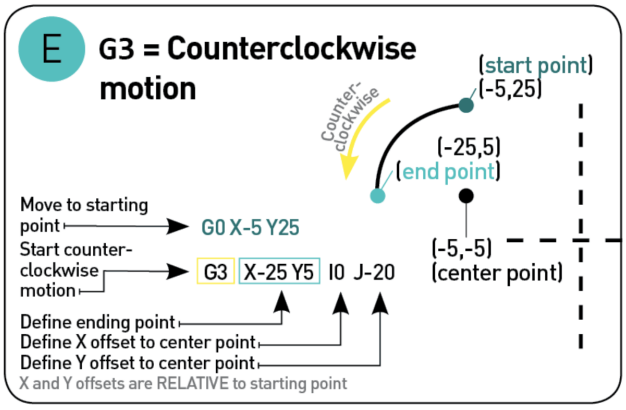
G2 tells the machine to move clockwise starting from its current location. The endpoint is specified by the coordinates X and Y. The centre of rotation is specified by the parameter I, which denotes the X offset of the current position from the centre of rotation. J denotes the Y offset of the current position from the centre of rotation.



An example of G2 command. (image source: all3dp.com)

9. G3: Counter clockwise motion

The G3 command creates a circular motion but in the counter clockwise direction.



An example of G3 command. (image source: all3dp.com)

10. Code comments

Comments and explanation are written in the code by including a semi-colon in front of the English text.

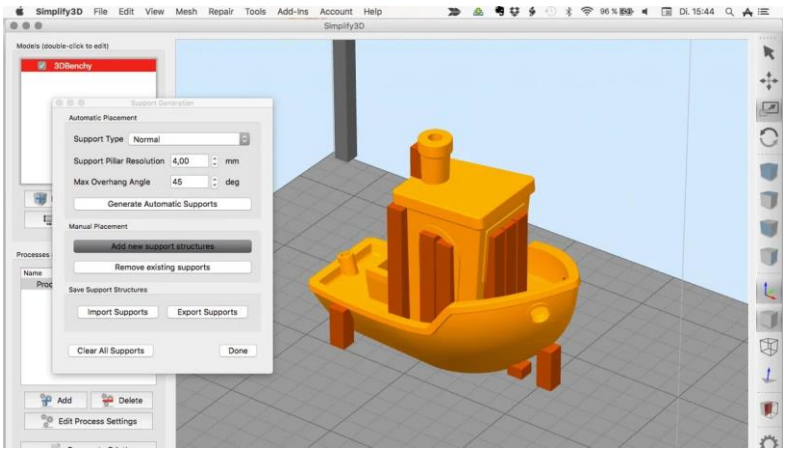
```
G0 X-25 Y5  ; rapid movement to X=-25 and Y=5
```

……

Slicer Recommendations

Using slicer software is an important step in the process of 3D printing. There are many slicers to choose from. It takes a lot of guesswork to choose the right slicer for yourselves. Most of the slicers are free, open-source, and will work with any desktop 3D printer.
Some are user friendly, and others might take some getting used to. But advanced users might want to choose a more powerful and complex slicer to fine-tune every setting.

Simplify3D



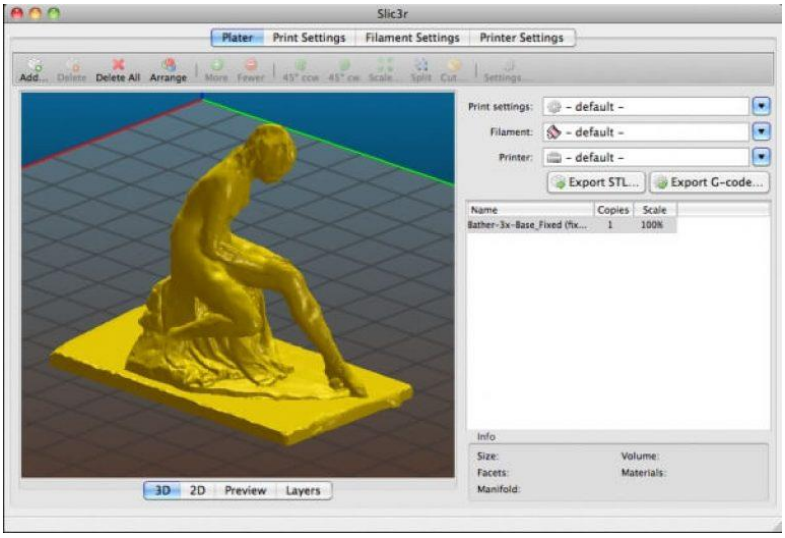
Pros:

- ✔ Helps in achieving high print quality
- ✔ Relatively high processing speed
- ✔ Provides freedom in terms of customization of settings
- ✔ Plenty of tools to support complex geometries and difficult 3D files

Cons:

- ✘ Beginners can find too confusing to use because of too many controls
- ✘ It is pretty expensive for beginners

Slic3r



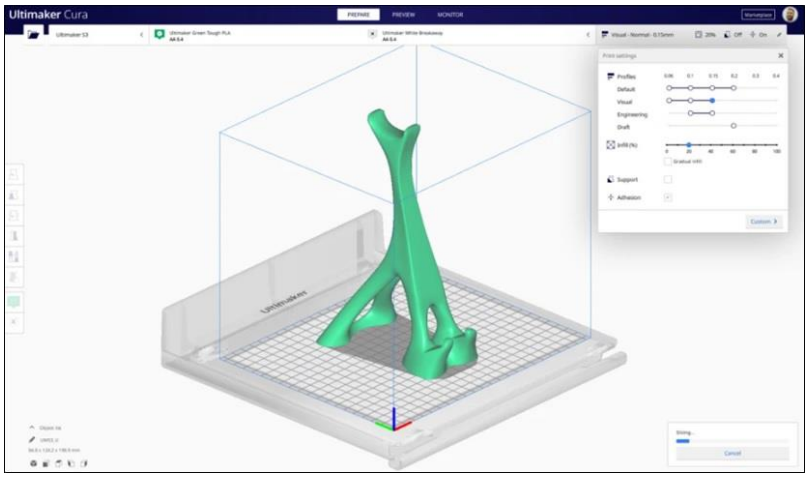
Pros:

- ✔ The software is open-source and free-to-use
- ✔ The software integration with Octoprint helps users directly upload files on Octoprint
- ✔ Honeycomb infill pattern can be used

Cons:

- ✘ A bit lengthy for beginners

Cura



Pros:

- ✔ The software is open-source and free-to-use
- ✔ The developer updates in a weekly basis

Cons:

- ✘ It can be annoying and time-consuming to uncollapse and change settings
- ✘ The processing of models is very slow

……

Tasks and Level of Completion

- This is a high prep project, lot of skill set is required before writing the code itself, so we have gone through all basics of slicing and a slicer.
- A rough windows app/UI, where we can import .stl(s) is ready
- Gone through the concepts about how to slice a 3D model into layers; and representing those layers in terms of triangular entities
- Pseudo Code for the slicer is ready (which has been presented from the next onwards).
- Analysis of errors in latest versions of slicers is done so as to try to overcome these issues while making our own slicer.

Pseudo Code

Here is a code, and explanations wherever required, to demonstrate the algorithm implemented by slicers.

```
#include <fstream>

#include <string>

#include <vector>

#include <math.h>

#include “glm/glm/glm.hpp”

#include “glm/glm/gtc/matrix_transform.hpp”

#include “glm/glm/gtc/type_ptr.hpp”

#define DEG_TO_RAD(x) (x*0.0174532925199f)

using namespace std;

/*

This code is merely for an understanding of structure of a Slicer program. This code may or may not work.

*/

struct v3 {

float x, y, z;

v3(float _x=0, float _y=0, float _z=0) : x(_x), y(_y), z(_z) {}

float dotproduct(const v3 &v) const { return x*v.x + y*v.y + z*v.z; }

void transform(const glm::mat4 &mat) { glm::vec4 v = glm::vec4(x, y, z, 1.0f), vt; vt = mat*v; x=vt.x; y=vt.y; z=vt.z;}

v3& operator-=(const v3 &pt) { x-=pt.x; y-=pt.y; z-=pt.z; return *this;    }

v3 operator-(const v3 &pt) { return v3(x-pt.x, y-pt.y, z-pt.z); }

v3 operator+(const v3 &pt) { return v3(x+pt.x, y+pt.y, z+pt.z); }

v3 operator/(float a) { return v3(x/a, y/a, z/a); }

v3 operator*(float a) { return v3(x*a, y*a, z*a); }

float normalize() const { return sqrt(x*x+y*y+z*z); }

};

v3 operator-(const v3 &a, const v3 &b) {return v3(a.x-b.x, a.y-b.y, a.z-b.z); }

v3 operator+(const v3 &a, const v3 &b) {return v3(a.x+b.x, a.y+b.y, a.z+b.z); }

struct LineSegment {

LineSegment(v3 p0=v3(), v3 p1=v3()) { v[0]=p0; v[1]=p1; }
```

```
v3 v[2];
};
class Plane {
public:
Plane() : mDistance(0) {}
float distance() const { return mDistance; }
float distanceToPoint(const v3 &vertex) const {    return vertex.dotproduct(mNormal) – mDistance; }
void setNormal(v3 normal) { mNormal = normal; }
void setDistance(float distance) { mDistance = distance; }
protected:
v3        mNormal;    // normalized Normal-Vector of the plane
float    mDistance;  // shortest distance from plane to Origin
};
struct Triangle    {
Triangle(v3 n, v3 v0, v3 v1, v3 v2) : normal(n) { v[0]=v0; v[1]=v1; v[2]=v2; }
Triangle& operator-=(const v3 &pt) { v[0]-=pt; v[1]-=pt; v[2]-=pt; return *this;}
void transform(const glm::mat4 &mat) { v[0].transform(mat); v[1].transform(mat); v[2].transform(mat);}
// @return -1 = all triangle is on plane back side
//          0 = plane intersects the triangle
//          1 = all triangle is on plane front side
//         -2 = error in function
int intersectPlane(const Plane &plane, LineSegment &ls) const {
// a triangle has 3 vertices that construct 3 line segments
size_t cntFront=0, cntBack=0;
for (size_t j=0; j<3; ++j) {
float distance = plane.distanceToPoint(v[j]);
if (distance<0) ++cntBack;
else ++cntFront;
}
if (3 == cntBack) {
```

```
return -1;
}
else if (3 == cntFront) {
return 1;
}
size_t lines[] = {0,1,1,2,2,0}; // CCW Triangle
std::vector<v3> intersectPoints;
for (size_t i=0; i<3; ++i) {
const v3 &a = v[lines[i*2+0]];
const v3 &b = v[lines[i*2+1]];
const float da = plane.distanceToPoint(a);
const float db = plane.distanceToPoint(b);
if (da*db<0) {
const float s = da/(da-db); // intersection factor (between 0 and 1)
v3 bMinusa = b-a;
intersectPoints.push_back(a+bMinusa*s);
}
else if (0==da) { // plane falls exactly on one of the three Triangle vertices
if (intersectPoints.size()<2
intersectPoints.push_back(a);
}
else if (0==db) { // plane falls exactly on one of the three Triangle vertices
if (intersectPoints.size()<2
intersectPoints.push_back(b);
}
}
if (2==intersectPoints.size()) {
// Output the intersecting line segment object
ls.v[0]=intersectPoints[0];
ls.v[1]=intersectPoints[1];
return 0;
}
```

```
return -2;
}
v3 v[3], normal;
};
class TriangleMesh {
public:
TriangleMesh() : bottomLeftVertex(999999,999999,999999), upperRightVertex(-999999,-999999,-999999) {}
size_t size() const { return mesh.size(); }
// move 3D Model coordinates to be center around COG(0,0,0)
void normalize() {
v3 halfBbox = (upperRightVertex – bottomLeftVertex)/2.0f;
v3 start = bottomLeftVertex + halfBbox;
for (size_t i=0; i<mesh.size(); ++i) {
Triangle &triangle = mesh[i];
triangle -= start;
}
bottomLeftVertex = halfBbox*-1.0f;
upperRightVertex = halfBbox;
}
void push_back(const Triangle &t) {
mesh.push_back(t);
for (size_t i=0; i<3; ++i) {
if (t.v[i].x < bottomLeftVertex.x) bottomLeftVertex.x = t.v[i].x;
if (t.v[i].y < bottomLeftVertex.y) bottomLeftVertex.y = t.v[i].y;
if (t.v[i].z < bottomLeftVertex.z) bottomLeftVertex.z = t.v[i].z;
if (t.v[i].x > upperRightVertex.x) upperRightVertex.x = t.v[i].x;
if (t.v[i].y > upperRightVertex.y) upperRightVertex.y = t.v[i].y;
if (t.v[i].z > upperRightVertex.z) upperRightVertex.z = t.v[i].z;
}
}
v3 meshAABBSize() const {
```

```
return v3(upperRightVertex.x-bottomLeftVertex.x, upperRightVertex.y-bottomLeftVertex.y,
upperRightVertex.z-bottomLeftVertex.z);
}
std::vector<Triangle>& getMesh() { return mesh; }
const std::vector<Triangle>& getMesh() const { return mesh; }
v3 getBottomLeftVertex() const { return bottomLeftVertex; }
v3 getUpperRightVertex() const { return upperRightVertex; }
// Mesh COG point should be at (0,0,0)
int transform(const glm::mat4 &mat) {
for (size_t i=0; i<mesh.size(); ++i) {
Triangle &triangle = mesh[i];
triangle.transform(mat);
}
return 0;
}
protected:
std::vector<Triangle> mesh;
v3 bottomLeftVertex, upperRightVertex;
};
// read the given STL file name (ascii or binary is set using 'isBinaryFormat')
// and generate a Triangle Mesh object in output parameter 'mesh'
int stlToMeshInMemory(const char *stlFile, TriangleMesh *mesh, bool isBinaryFormat) {
if (!isBinaryFormat) {
ifstream in(stlFile);
if (!in.good()) return 1;
char title[80];
std::string s0, s1;
float n0, n1, n2, f0, f1, f2, f3, f4, f5, f6, f7, f8;
in.read(title, 80);
while (!in.eof()) {
in >> s0;                    // facet || endsolid
if (s0=="facet") {
```

```
in >> s1 >> n0 >> n1 >> n2;           // normal x y z
in >> s0 >> s1;                      // outer loop
in >> s0 >> f0 >> f1 >> f2;          // vertex x y z
in >> s0 >> f3 >> f4 >> f5;          // vertex x y z
in >> s0 >> f6 >> f7 >> f8;          // vertex x y z
in >> s0;                            // endloop
in >> s0;                            // endfacet
// Generate a new Triangle with Normal as 3 Vertices
Triangle t(v3(n0, n1, n2), v3(f0, f1, f2), v3(f3, f4, f5), v3(f6, f7, f8));
mesh->push_back(t);
}
else if (s0==”endsolid”) {
break;
}
}
in.close();
}
else {
FILE *f = fopen(stlFile, “rb”);
if (!f) return 1;
char title[80];
int nFaces;
fread(title, 80, 1, f);
fread((void*)&nFaces, 4, 1, f);
float v[12]; // normal=3, vertices=3*3 = 12
unsigned short uint16;
// Every Face is 50 Bytes: Normal(3*float), Vertices(9*float), 2 Bytes Spacer
for (size_t i=0; i<nFaces; ++i) {
for (size_t j=0; j<12; ++j) {
fread((void*)&v[j], sizeof(float), 1, f);
}
fread((void*)&uint16, sizeof(unsigned short), 1, f); // spacer between successive faces
```

```
Triangle t(v3(v[0], v[1], v[2]), v3(v[3], v[4], v[5]), v3(v[6], v[7], v[8]), v3(v[9], v[10], v[11]));
mesh->push_back(t);
}
fclose(f);
}
mesh->normalize();
return 0;
}
// Take an input Triangle Mesh ‘mesh’ and fill the output
// parameter ‘slicesWithLineSegments’ with line segments for
// each slice
int triMeshSlicer(
const TriangleMesh                    *mesh,       // the const input mesh
std::vector<std::vector<LineSegment>>    &slicesWithLineSegments,
// the result slices
const float                           sliceSize) {// slice size in 3D Model digital units
Plane plane;                                  // The intersection plane
plane.setNormal(v3(0, 0, 1));                    // normal does not change during slicing
const v3 aabb = mesh->meshAABBSize();              // as the model for it’s 3D axis-aligned bounding-box
const size_t nSlices = 1+(int)(aabb.z/sliceSize);    // compute number of output slices
const std::vector<Triangle> &m = mesh->getMesh();    // get a const handle to the input mesh
const float z0 = mesh->getBottomLeftVertex().z;       // find the minimal z coordinate of the model (z0)
for (size_t i=0; i<nSlices; ++i) {                 // start generating slices
std::vector<LineSegment> linesegs;               // the linesegs vector for each slice
plane.setDistance(z0+(float)i*sliceSize);       // position the plane according to slice index
for (size_t t=0; t<m.size(); ++t) {              // iterate all mesh triangles
const Triangle &triangle = m[t];          // get a const handle to a triangle
LineSegment ls;
if (0==triangle.intersectPlane(plane, ls)) {// the plane does intersect the triangle
linesegs.push_back(ls);                  // push a new Line Segment object to this slice
```

```
}
}
slicesWithLineSegments.push_back(linesegs);       // push this vector to the slices vector
}
return 0;
}
// GIV is a free viewer: http://giv.sourceforge.net/giv/
// output a single GIV ascii file with the slices, layed out in a matrix form
int    exportSingleGIVFormat(
const std::vector<std::vector<LineSegment>> &slicesWithLineSegments,
const v3                                    &aabbSize) {
char filename[256];
FILE *f=NULL;
float dx=0, dy=0;
f=fopen("out.marks", "w");
if (!f)    return 1;
const size_t nSlices = slicesWithLineSegments.size();
const size_t slicePerRow = (size_t)sqrt((float)nSlices);
for (size_t i=0; i<nSlices; ++i) {
const std::vector<LineSegment> &lss = slicesWithLineSegments[i];
dx = (float)(i%slicePerRow)*(aabbSize.x*1.05f);
dy = (float)(i/slicePerRow)*(aabbSize.y*1.05f);
for (size_t j=0; j<lss.size(); ++j) {
fprintf(f, "\n\n$line");
fprintf(f, "\n$color blue");
fprintf(f, "\n%f %f", dx+lss[j].v[0].x, dy+lss[j].v[0].y);
fprintf(f, "\n%f %f", dx+lss[j].v[1].x, dy+lss[j].v[1].y);
}
}
fclose(f);
return 0;
}
```

```
// GIV is a free viewer: http://giv.sourceforge.net/giv/
// output a single GIV ascii file with the slices, layed out in 3D form
int    exportSingleGIVFormat3D(
const std::vector<std::vector<LineSegment>> &slicesWithLineSegments,
const v3                              &aabbSize) {
// Generate a 3D View using OpenGL Math Library
glm::detail::tvec3<float> eulerAngles(0, 0.0f, 45.0f);
glm::detail::tquat<float> quaternion = glm::detail::tquat<float>(DEG_TO_RAD(eulerAngles));
// This glm func wants values as radians
float angle = glm::angle(quaternion);
glm::detail::tvec3<float> axis = glm::axis(quaternion);
glm::mat4 View        = glm::rotate(glm::mat4(1.0), angle, axis);
glm::mat4 Projection  = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.f);
glm::mat4 Model       = glm::lookAt(
glm::vec3(1, 1, 1),    // Eye point (where am I?)
glm::vec3(0, 0, 0),    // Look at Center point (What am I looking at?)
glm::vec3(0, 1, 0));// UP vector of the camera (Where is my UP vector?)
glm::mat4 MVP = Projection * View * Model;
// Start Output
char filename[256];
FILE *f=NULL;
v3 p0, p1;
f=fopen("out3D.marks", "w");
if (!f)    return 1;
const size_t nSlices = slicesWithLineSegments.size();
const size_t slicePerRow = (size_t)sqrt((float)nSlices);
for (size_t i=0; i<nSlices; ++i) {
const std::vector<LineSegment> &lss = slicesWithLineSegments[i];
for (size_t j=0; j<lss.size(); ++j) {
p0=lss[j].v[0];
p1=lss[j].v[1];
p0.transform(MVP);
```

```
p1.transform(MVP);
fprintf(f, “\n\n$line”);
fprintf(f, “\n$color blue”);
fprintf(f, “\n%f %f”, p0.x, p0.y);
fprintf(f, “\n%f %f”, p1.x, p1.y);
}
}
fclose(f);
return 0;
}
int main(int argc, char **argv) {
float    sliceSize = 1.0f;
char    modelFileName[1024];
bool    isBinaryFormat = true;
glm::detail::tvec3<float> eulerAngles(0,0,0);
for (int i=1; i<argc;) {
if (0==strcmp(argv[i], “-slicesize”)) {
sliceSize = atof(argv[i+1]);
i+=2;
}
else if (0==strcmp(argv[i], “-model”)) {
strcpy(modelFileName, argv[i+1]);
i+=2;
}
else if (0==strcmp(argv[i], “-ascii”)) {
isBinaryFormat = false;
i+=1;
}
else if (0==strcmp(argv[i], “-eulerangles”)) {
eulerAngles.x = atof(argv[i+1]);
eulerAngles.y = atof(argv[i+2]);
eulerAngles.z = atof(argv[i+3]);
```

```
i+=4;
}
else {
++i;
}
}
// Parse the file and generate a TriangleMesh
TriangleMesh mesh;
if (stlToMeshInMemory(modelFileName, &mesh, isBinaryFormat)!=0)
return 1;
fprintf(stderr, “Mesh has %d triangles\n”, mesh.size());
// Optional Model Rotation Around COG Point using GLM Library
glm::mat4 mat = glm::mat4(1.0f);
glm::detail::tquat<float> quaternion = glm::detail::tquat<float>(DEG_TO_RAD(eulerAngles));
// This glm func wants values as radians
float angle = glm::angle(quaternion);
glm::detail::tvec3<float> axis = glm::axis(quaternion);
mat = glm::rotate(mat, angle, axis);
mesh.transform(mat);
// Generate Slices
std::vector<std::vector<LineSegment>> slicesWithLineSegments;
triMeshSlicer(&mesh, slicesWithLineSegments, sliceSize);
// Output in 2D Matrix form
exportSingleGIVFormat(slicesWithLineSegments, mesh.meshAABBSize());
// Output in 3D Layered Form
exportSingleGIVFormat3D(slicesWithLineSegments, mesh.meshAABBSize());
return 0;
}
```

Future Plans

- Get more grip on the Github repos of Cura and Slic3r. Working on building the mesh algorithms for the slicer.
- Implementation of Pseudocode and incorporation of the code into the App
- More changes to the UI of the app, making it user-friendly such as giving import .stl(s) file option to user{rather than hardcoded}, output spaces to show compiled data to users
- Compiling everything done to the point and focusing more on algorithms part for processing.
- Explore algorithms to overcome difficulties like overhangs, corners, small lengths or supports and support interface.

Inference and Conclusion

- 3D Modelling -> Slicer -> Printing is a basic flowchart of the 3D printing process. While modelling and printing are fairly straightforward processes, slicing is not. Modelling involves imprinting the design in your brain onto a computer software (requiring only modelling skillsets), and printing is basically a computer following pre-fed steps.
- Slicing is where the magic happens: it's the translator. It takes the human readable idea (model) and produces commands for the printer to follow (computer readable) which is known as G-code. 3D printer prints any object layer by layer from its bottom layer upwards.
- They analyse and visualize how every cross- section looks like. Then, it needs to compute what is the most efficient and optimum path that the printer head can move along. User ready slicer software let you modify structural features like number of walls the outer surface has, the filling pattern, whether interior should be hollow or filled, options to add supports to assist proper curing of print etc.
- The final output of the slicing process is G-Code. It has instructions to set filament temperature, printer head positioning and orientation, and control the release of filament.

.....

Bibliography

1. http://edutechwiki.unige.ch/en/Slicers_and_user_interfaces_for_3D_printers
2. https://the3dbros.com/3d-slicing-software-the-basics/
3. https://en.wikipedia.org/wiki/3D_printing
4. https://www.instructables.com/id/How-to-Build-Your-Own-3D-Printing-Slicer-From-Scra/
5. https://all3dp.com/2/the-best-printing-temperature-for-different-filaments/
6. https://www.fictiv.com/hwg/fabricate/recommended-wall-thickness-for-3d-printing (common defects and how to solve them)
7. https://3dprinterly.com/what-is-the-strongest-infill-pattern/
8. https://www.reddit.com/r/3Dprinting/comments/al68zq/the_importance_a_slicer_can_have_same_layer/
9. https://pick3dprinter.com/3d-slicer-software/
10. https://www.binpress.com/ultimaker-cura-3d-printing-software-review/
11. https://mycrazygoodlife.com/3d-printing-supports-rafts-skirts-and-brims/
12. https://all3dp.com/1/3d-printer-file-format/
13. https://all3dp.com/g-code-tutorial-3d-printer-gcode-commands/
14. https://www.cmac.com.au/blog/5-vital-things-about-stl-file-format-3d-printing#:~:text=There%20are%20special%20rules%20for,Below%20are%20some%20of%20them.&text=In%20STL%20formatting%2C%20this%20rule,the%20side%20of%20another%20triangle.