Advanced Modelling and Infilling Algorithms for 3D Printing

Richard Tynan
Cistercian College Roscrea

Stand Number: _____

January 2014

Comments

Contents

1 Abstract

Infill patterns are a vital part of 3D printing objects, especially when using today's consumer-grade plastic extrusion printers. The job of an infill pattern is to act as a support structure for the model from the inside. This negates the need to fill the model solidly, saving materials and time.

In this project I aim to design and effectively implent several new, possibly more effecient, infill algorithms. Currently, infill algorithms create regular two dimensional patterns upwards through the model. While these are simple to implement, they fail to take advantage of the printer's ability to handle complex three dimensional shapes. It is proposed that three dimensional shapes may produce a more evenly distributed and stronger support for models.

To approach the creation of these infill patterns, I created software which applied these infills to models. I then printed models containing these new infills to test how effective they are. For fair comparison between new and current infills, the primary controlled variable was that all models were printed with the same volume of plastic to be used in the infill. Resulting prints showed that some of the new infills performed very well, providing more support and giving more strength to the model, than currently availible infills, even for the same amount of plastic used.

In conclusion new infill patterns can be more effective and efficient if implemented correctly. To extend upon this work, direct implementation in the current tool-chain of printing software is recommended.

2 Glossary

I have placed the glossary near the start of the report because many of the terms are used throughout.

2.1 FDM Terms

FDM
Fused Deposition Manufacturing, commonly known as a form of 3D printing. By laying down layers of material which adhere to each layer underneath, a model is built up. The material used, commonly plsatic, is extruded out the nozzle, forming a thin filament known as extrudate. A combination of the nozzle and table moving places the extrudate on different parts of the model.

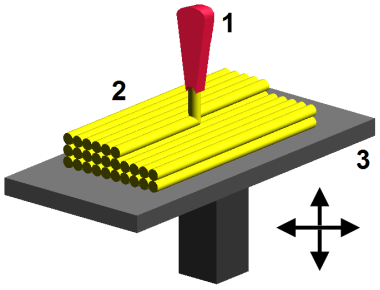


Figure 1: 1 - nozzle ejecting molten material (plastic), 2 - deposited material (modelled part), 3 - controlled movable table *via Wikipedia*

Infill Definition contained in the Introduction.

G-code G-code is the language which tells the nozzle how to build the model. It contains instructions for each layer, to move the nozzle, the speed to move at and how much material to extrude.

Slicer A slicer is a program which takes a 3D CAD model as input, "slices" it into layers and then outputs G-code instructions to print each layer.

Bridging
Because extrudate is molten when extruded, it must be placed on the model. If it is not, it will sag or fall down before hardening. If a printer must "bridge" over a gap, the must bring the extrudate across the gap to the other side, sticking it to both sides of the gap. Successful bridging is required to print flat tops of models, for example on cubes. One of

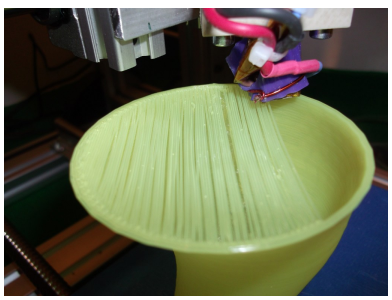


Figure 2: An extreme example of bridging *via [1]*

the roles of an infill is to support the bridge, making the gap that must be bridged smaller.

Forty-Five Degree Rule In a similar vein to bridging, this rule deals with sagging extrudate. If an overhang is of too steep an angle, extrude may fall off the edge of each layer. If a face of a model is below 45° to the horizontal, extrudate will often fall. Bridging is an exception to this rule, as the extrude ends up being support on both sides.

2.2 Software

Wolfram Mathematica Used to handle the equations relating density to measurements of cells and produce answers.

OpenSCAD A Solid 3D CAD modeller, OpenSCAD is used to construct the infills and place them on the inside of the models. They are placed inside the models by overlapping the model and infill, and keeping the intersection of the two (referred to as intersecting the infill with model later on in the report).

CGAL The *Computational Geometry Algorithms Library*, a library for C++, is used to take the model as input and pass on information to OpenSCAD, such as vertices, size, etc.

3 Introduction

3.1 Background

The main objective of this project was to develop software to allow the experimentation of 3D infilling algorithms on simple 3D models. Then to practically test the effectiveness of these algorithms on a 3D printer, analyse them in comparison to one another and determine the ease at which they could be effectively implemented in today's 3D printing Computer Aided Design (CAD) tool-chain.

I first became interested in general Computer Aided Manufacture (CAM) several years ago, and after a while, realised that it would be feasible to build my own 3D printer, because of the RepRap Project[4]. After becoming involved with the 3D printing community, I became very interested in the development and design of the software and concepts behind the process of taking a model and producing a printable object from it.

Martin Baumers' work[2] on topology optimisation and printing models created with these algorithms really focussed my interests on altering and creating models without human intervention. The key role that additive manufacturing plays here is to allow these complex designs to be produced, as most topologically optimised models can not be manufactured by traditional means.

One of the major aspects of 3D printing by the Fused Deposition Modelling (FDM) process is the infill printed inside objects. After reading an article by Gary Hodgson[3], I became interested in the development of new, more efficient infills. In his article, Hodgson gives examples of some new ways in which infills could be made more efficient and robust. However, he ran only one test, which was a model in which he manually placed spheres to simulate one of his ideas. He then printed this test model. I began to think about how I could make a more robust testing suite, implement multiple infill algorithms which would be applied to any model of choice and how I could test these algorithms many times.

3.2 The Importance of Infills

The infill forms one of the two main parts of any 3D print which is done by FDM. The other part, the shell, is the outside of the object. Of course an object could be printed with no infill, however most ordinary models, even if they have been designed with FDM in mind, still require some form of infill.

There are two main reasons to use infills:

- **Self Support** If the object has a "roof" or top without support, this would sag downwards as the molten extrudate (extruded material) would have no support and collapse.

- **External Support** If the object requires a certain amount of strength for its intended function, being hollow could be too weak, and it may give in to external forces, snapping or buckling.

The question may be asked why not fill in the object solidly? As in have an infill which takes up all inside space with material and leaves no air whatsoever. Again, there are two main reasons why not to infill the model solid:

- **Material** Infilling solidly is often a waste of material, as it is simply not needed. This increases the raw material cost of making the object.
- **Time** 3D printers by their nature have very small nozzles, often around 0.5mm for plastic FDM machines. This allows them to be detailed, but also means more time would spent covering an area entirely in extrudate, as the small nozzle would need to be brought over all of the area, at a certain fixed speed. Besides time, this could also increase the cost of making an object.

3.3 Current Available Infills

My first step was to look at the already available infills, some of which are shown.

The most widely used infill pattern, mainly for its simplicity is the *rectilinear* pattern. There are several other widely available infill patterns. However a common theme of all of these patterns is that they are simply 2D designs which are extruded upwards with the print. While these designs are effective, they do not take advantage of the nature of 3D printers, in the sense that they are not exactly 3D.

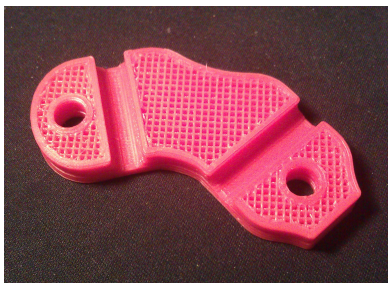


Figure 3: An unfinished printed part showing the interior regular grid of the rectilinear infill

Infills tend to have a density parameter. This density is usually expressed as a percentage and ranges from 0%, meaning no infill or hollow, to 100%, meaning completely solid fill.

3.4 Conception of 3D Infills

After the initial development of the software, I had four infills designed and spent some time making them more efficient and printable. My four infills are as follows:

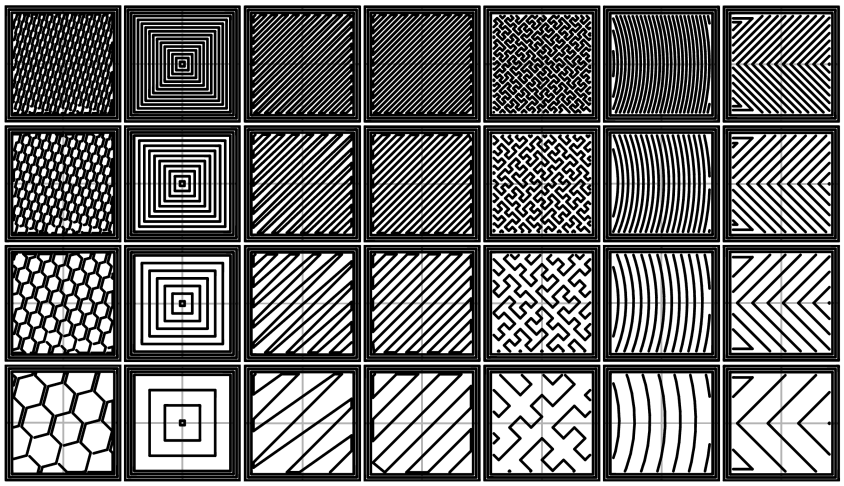


Figure 4: Showing various infill patterns at varying densities which are available with, *slic3r*, a popular printing slicer program.
Fill Density - Top to Bottom: 80%,60%,40%,20%.
Pattern - Left to Right: Honeycomb, Concentric, Line, Rectilinear, Hilbert Curve, Archimedean Chords, Octagram Spiral
Source: Slic3r Manual[5]

1. **Sphere Packing** A simple infill, following one of Hodgson's suggestions. It creates a regular cubic packing of spheres of a specified radius and then intersects this packing with the model, filling the insides with whole and partially cut spheres.

2. **Truncated Octahedron Tessellation** This infill was theorised and mentioned briefly by Hodgson, and in my implementation creates a tessellation of truncated octahedra based on a specified edge length of the truncated octahedron. The tessellation is then placed inside the model in a similar manner to the sphere packing.

3. **Rhombic Dodecahedron Tessellation** After thinking of ways to improve the truncated octahedron infill, I began to research other polyhedra which can tessellate 3D space. The rhombic dodecahedron is a better shape for printing for several reasons, specified later. My implementation creates the rhombic dodecahedra based on the length between their points (the vertices where four acute angles meet). Again, the tessellation is intersected with the model.

4. **Dynamic Vertex Support** The last infill is based on Hodgson's thoughts on using the vertices of the model to generate internal edges. As he did not specify any particular pattern or design which could be used, I designed my own algorithm. It selects vertices which are all a

specified distance away from each other and then joins them to meet in the centre of the object with struts of sort.

4 Design & Analysis of Algorithms

In this section, there will be detailed descriptions of each infill algorithm, the concepts behind them and their implementation.

4.1 Sphere Packing

4.1.1 Description

The sphere packing infill is quite simple. A cubic packing of spheres, which is then intersected with the model. Each sphere is specified a radius and each one is two radii apart from its nearest neighbour.

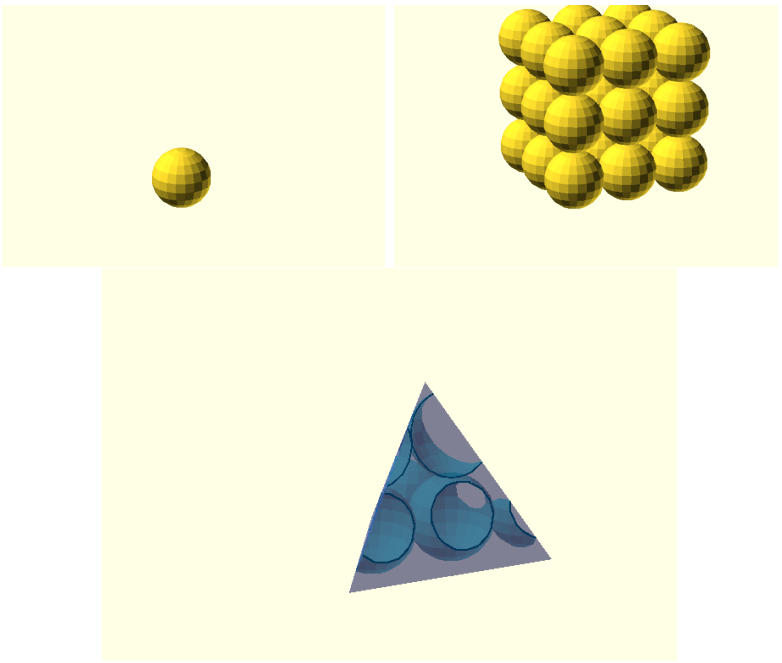


Figure 5: First a sphere is created, cubic packed and finally placed inside the model's shell

4.1.2 Density

As each sphere is of radius r, the volume of space taken up by plastic will be the volume of a sphere of radius r, minus the volume of sphere radius $(r - n)$, where n is the width of the extrudate. For our purposes assume n equals the nozzle width. As the density is ratio of the volume taken up by plastic to the volume it occupies and each sphere occupies a cube of edge

length $2r$, we can derive this equation:

$$\rho = \frac{\frac{4}{3}\pi r^3 - \frac{4}{3}\pi(r-n)^3}{(2r)^3}$$

We now have infill density (on average) expressed in a value between 0 and 1 and in terms radius and nozzle width. Nozzle width will be known by the person printing, and they will select a density value, ergo a radius can be calculated. Unfortunately, using Mathematica to express r yields this rather long equation:

$$r = \frac{\sqrt[3]{\pi}\ \sqrt[3]{3\sqrt{36\,n^6\,\rho^4 - 12\,\pi\,n^6\,\rho^3 + \pi^2\,n^6\,\rho^2}\ + 18\,n^3\,\rho^2 - 9\,\pi\,n^3\,\rho + \pi^2\,n^3}}{6\,\rho} -$$

$$\frac{54\,\pi\,n^2\,p - 9\,\pi^2\,n^2}{54\sqrt[3]{\pi}\ \rho\sqrt[3]{3\sqrt{36\,n^6\,\rho^4 - 12\,\pi\,n^6\,\rho^3 + \pi^2\,n^6\,\rho^2}\ + 18\,n^3\,\rho^2 - 9\,\pi\,n^3\,\rho + \pi^2\,n^3}} + \frac{\pi\,n}{6\,\rho}$$

Here is a sample table of density values with $n = 0.4$ as that is the nozzle width of my printer (measured in mm), again computed with Mathematica. Two of the three answers to this third degree polynomial are complex, so the real root is taken.

Infill Density	Radius (mm)
0.05	12.1575
0.10	5.86436
0.15	3.75885
0.20	2.699
0.25	2.05603
0.50	0.620997
0.75	0.227767
1.00	0.203149

4.2 Truncated Octahedron Tessellation

4.2.1 Description

This infill algorithm relies on the ability of truncated octahedra to tessellate space uniformly[6]. A tessellation is made and intersected with the model.

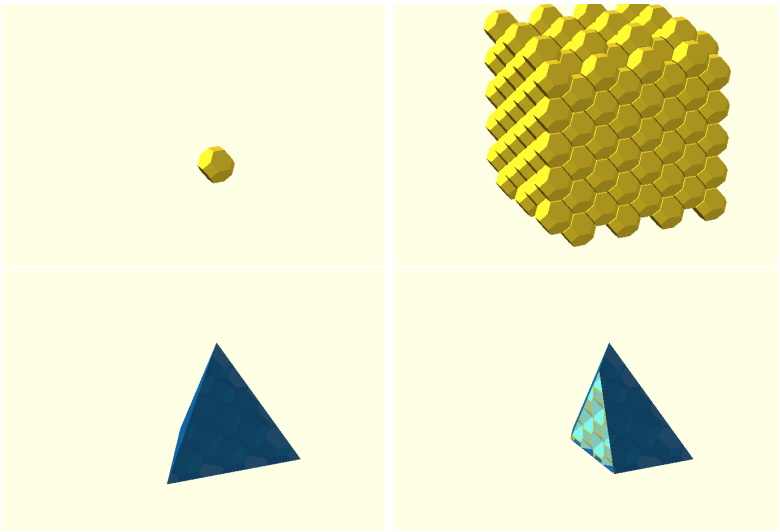


Figure 6: A truncated octahedron is created, tessellated and finally placed inside the model's shell, with a section view on bottom right

4.2.2 Density

The volume of a truncated octahedron of edge length a is $8\sqrt{2}a^3$ (source: [6]). The density of a hollow truncated octahedron can be got similarly to the sphere packing. However there are a few details to note.

- As each truncated octahedron touches another, the width of plastic they create together should equal the nozzle width. Therefore, the width of $n/2$ should taken off on each side to get the size of the inner truncated octahedron. Of course, taking $2(n/2)$ simply results in n.
- The edge length of the inner truncated octahedron (a_i) is not simply the outer edge length minus nozzle width ($a_o - n$). The distance from the square face of a truncated octahedron to its opposite square face is $2\sqrt{2}a$ (source: [6]), which I'll refer to as height since that is how it is orientated in my program. We know that the square face of the inner truncated octahedron should be $n/2$ away from the outer shape's corresponding square face, so we can say that the inner truncated octohedron's height is $2\sqrt{2}a_o - 2(n/2)$. Logically, it follows that this

equation is true: $2\sqrt{2}a_o - 2(n/2) = 2\sqrt{2}a_i$. Solving, we get

$$a_i = a_o - \frac{n}{2\sqrt{2}}$$

Taking the above into account, the density can be expressed as follows:

$$\rho = \frac{8\sqrt{2}a_o^3 - 8\sqrt{2}(a_o - \frac{n}{2\sqrt{2}})^3}{8\sqrt{2}a_o^3}$$

Once again, this results in a rather long equation when expressing a_o in terms of n and ρ, so here is a table of various densities, again taking $n = 0.4$:

Infill Density	a_0 (mm)
0.05	8.34225
0.10	4.09791
0.15	2.6819
0.20	1.97289
0.25	1.54661
0.50	0.685515
0.75	0.382179
1.00	0.141421

4.3 Rhombic Dodecahedron Tessellation

4.3.1 Description

This infill algorithm once again relies on a polyhedron's ability to tessellate space, this time the rhombic dodecahedron[7]. Again a tessellation is created and intersected with the model.

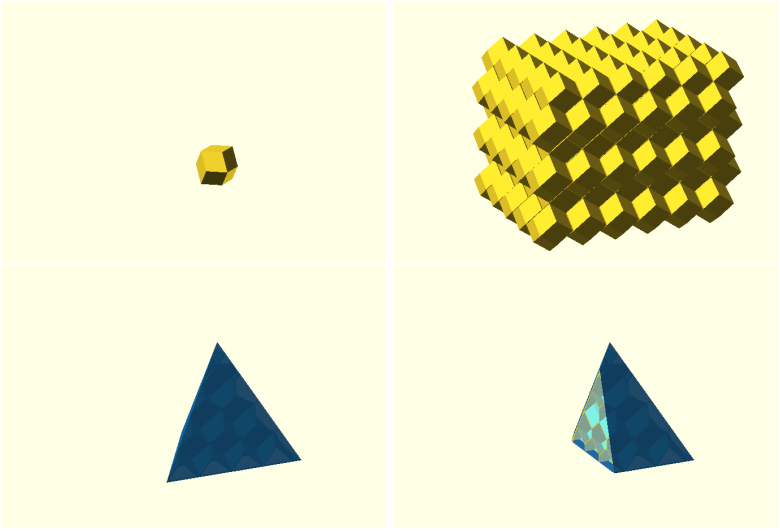


Figure 7: A rhombic dodecahedron is created, then tessellated, and finally placed inside the model's shell, with a section view on bottom right

4.3.2 Density

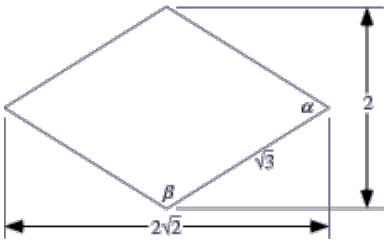


Figure 8: Rhombic face of a Rhombic Dodecahedron[7]

My program takes the distance between the two vertices where the acute angles of four rhombi meet on the rhombic dodecahedron to draw it. This measurement (h) is twice the shorter measurement on one of the rhombic faces (b), which is shown as 2 on the diagram. It is clear that the two lengths ($a : b$) are in ratio $2\sqrt{2}/2$. Therefore the following must hold for rhombic faces on rhombic dodecahedra:

$$\frac{2\sqrt{2}}{2} = \frac{a}{b}$$

Subbing $b = h/2$ and simplifying:

$$\frac{h}{\sqrt{2}} = a$$

a is important because the volume of rhombic dodecahedron can be expressed in terms of edge length of their rhombic faces s. Once a and b are found, simple trigonometry yields $s^2 = (a/2)^2 + (b/2)^2$. After subbing in $\frac{h}{\sqrt{2}} = a$ and $b = h/2$ and simplifying:

$$s = \sqrt{\frac{3h^2}{16}}$$

From source [7], the volume of a rhombic dodecahedron can be expressed as $\frac{16}{9}\sqrt{3}s^3$. Again, the density can be expressed in a similar manner, the difference between the volumes of an outer and inner rhombic dodecahedron, over the space taken up. The h measurement of the inner rhombic dodecahedron will equal the outer one's h measurement minus n.

$$\rho = \frac{\frac{16}{9}\sqrt{3}(\sqrt{\frac{3h^2}{16}})^3 - \frac{16}{9}\sqrt{3}(\sqrt{\frac{3(h-n)^2}{16}})^3}{\frac{16}{9}\sqrt{3}(\sqrt{\frac{3h^2}{16}})^3}$$

Again, expressing h in terms of ρ and n yields this long equation.

$$h = \frac{\sqrt[3]{\sqrt{n^6\rho^4 - 6n^6\rho^3 + 13n^6\rho^2 - 12n^6\rho + 4n^6} + n^3(-\rho^2) + n^3\rho}}{\sqrt[3]{2}\,(\rho - 2)} - \frac{\sqrt[3]{2}\,(9n^2 - 9n^2\rho)}{9(\rho-2)\sqrt[3]{\sqrt{n^6\rho^4 - 6n^6\rho^3 + 13n^6\rho^2 - 12n^6\rho + 4n^6} + n^3(-\rho^2) + n^3\rho}} - \frac{n}{\rho - 2}$$

So I've compiled another table to show some values.

Infill Density	h (mm)
0.05	23.5954408
0.10	11.5906365
0.15	7.58556093
0.20	5.58018328
0.25	4.37446737
0.50	1.93892884
0.75	1.08096575
1.00	0.2

4.4 Dynamic Vertex Support

4.4.1 Description

While this infill algorithm is quite simple, it could be rather effective. It iterates through every vertex, checking how far away said vertex is from any other vertex that has already been chosen. If the vertex is the specified distance d or more away from every other chosen vertex, it is appended to the chosen list.

This list is used to draw strut-like objects from the chosen vertices to the centre, which support the object.

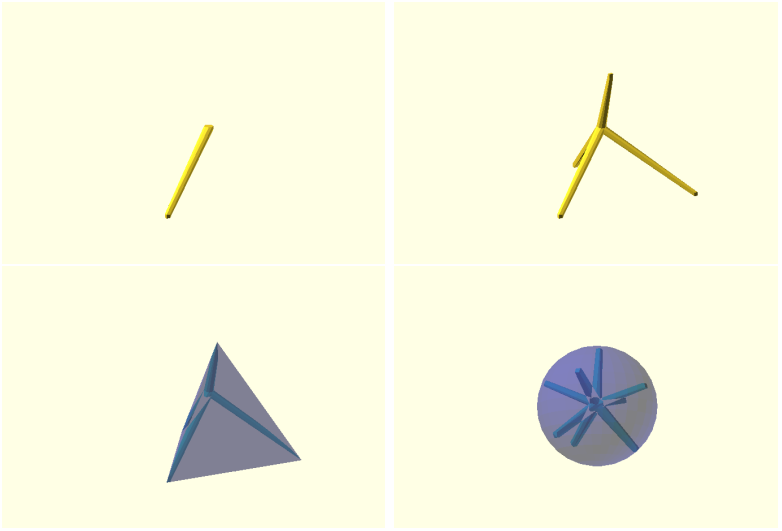


Figure 9: Struts like the one on top left are connected to vertices, to result in a support structure on the top right, then placed inside the model's shell (bottom left). This infill algorithm is particularly useful for spherical, curved and organic-shaped models, for example the sphere on the bottom left.

4.4.2 Density

This infill algorithm is quite different to the others and much more difficult to estimate mathematically, however computationally it is feasible. As such I offer no formal estimates.

4.4.3 Vertex Choosing Computational Complexity

As estimating the infill density would be too inaccurate if done by hand, I instead choose to look at the computational complexity of the algorithm, when choosing vertices to support. While the other algorithm's running times consist almost solely of the time it takes to render the infill patterns

and intersect them with the model, this algorithm has the vertex choosing added on to that.

Worst-case scenario, the algorithm will add every single vertex it meets onto the list of chosen vertices, and then have to iterate over them again. Iteration through all of the vertices is simply v times, where v is the number of vertices. However each chosen vertex must be iterated over each time. It follows that the complexity will be the sum $\{1 + 2 + 3 + \cdots + v - 1\}$, ie. at $v = 1$ there will be no chosen vertices to iterate over, at $v = 2$ there will be one, up until the final vertex where there will be $v - 1$ chosen vertices to iterate over.

Using arithmetic sum formula, the complexity of $O(\frac{1}{2}v(v-1))$ is found. This running time is not particularly good as it increases faster than v does, but is usable for models with less than 40,000 or so vertices. Improvements could probably be made using heuristic methods.

5 Experimental Methods - Printing

As the main aim of this project is to see how printable the new infills are, the practical application consists of printing them using test models. The main test model was a cube of 40mm side length. To be able to see the infill, the printing was stopped before the models had finished. Some other models where also completed to see how infill affects the outside appearance. Below are photos of the process.

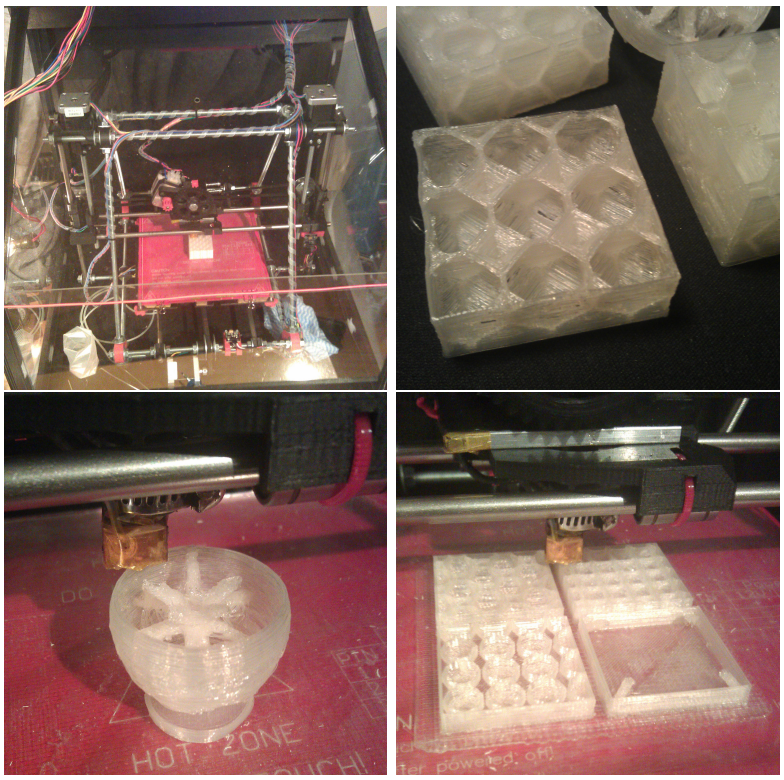


Figure 10: Clockwise from top right:

The printer;
The results of some early prints, pictured foreground is the first rhombic dodecahedron tessellation infill print with early prints of others around it;
A sphere printing with the vertex support infill;
The four new infills, all (except the vertex support infill) having a density of 10%.

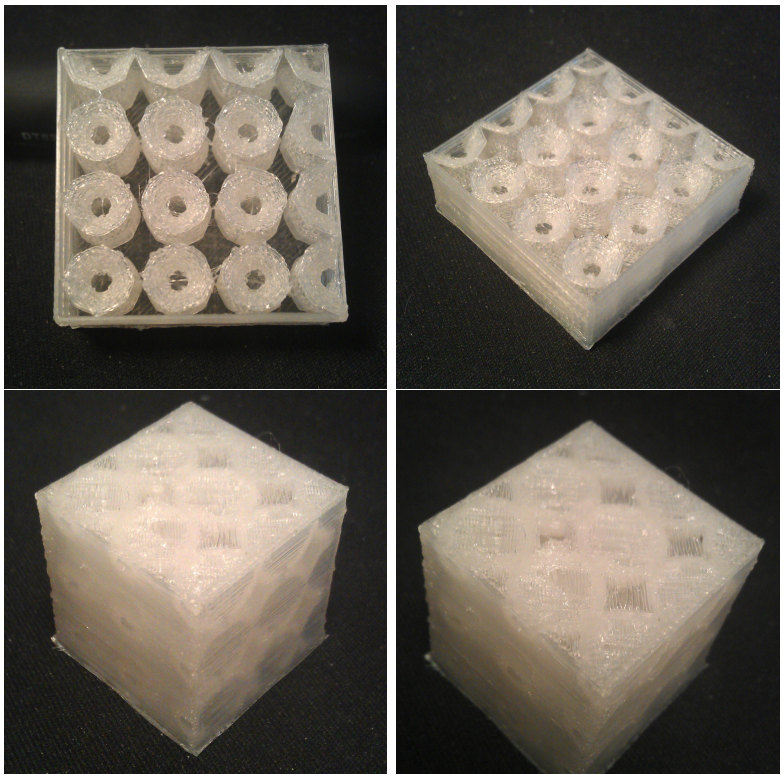
6 Results & Conclusions

In this section I will include pictures of resulting prints, each at 10% fill density. I will then follow up with a list of conclusions about each infill algorithm, observations and suggestions for improvement.

I have included the some rectilinear infill prints also at an infill density of 10% at the end of this section to allow comparison.

Also note, that each of the completed cubes was printed with one top layer (the top "roof" of the cube), to show how effective the bridging is. Because of this, there are some holes in this top layers. Usually however, most printers decide to have 3 top layers to ensure no holes are present.

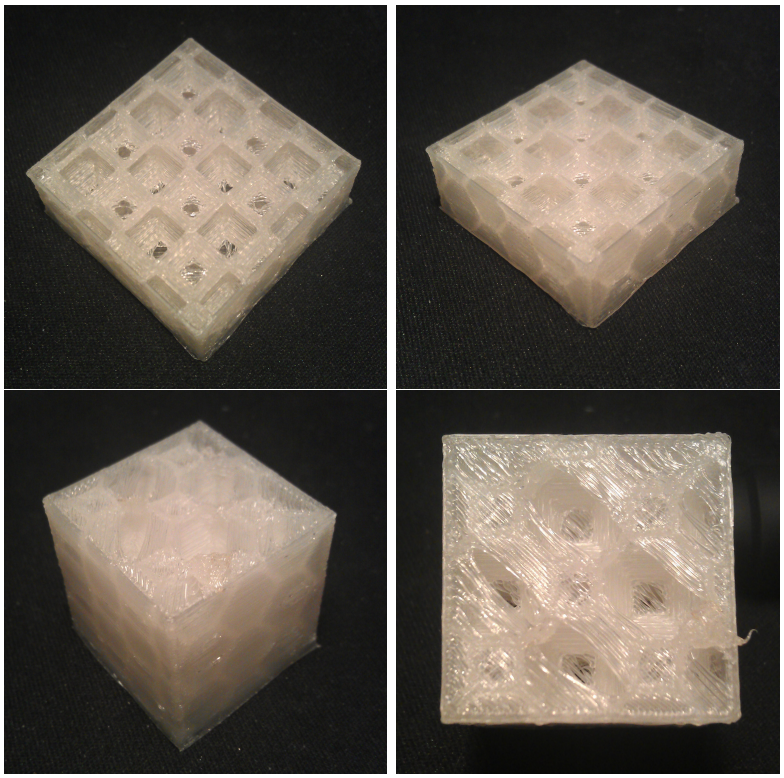
6.1 Sphere Packing



- As can be seen from the top pictures, small spheres are easily printed.
- From top right, we can see that there are issues with getting the spheres to intersect, if they are rendered with a low amount of fragments. This could be easily fixed if sphere packing was implement in a slic3r and written directly into the G-code.

- The gaps which the printer has to bridge on the top layer are of a similar size to the gaps left on rectilinear infill at 10%. The worst-case scenario is that the spheres are cut through their centres by the top of the model. This would leave circular gaps of radius r, but they are comparable to the rectilinear gaps of the same density.
- Unlike the rectilinear pattern, this infill leaves no long unsupported vertical areas down the side of the model, see bottom left. It does however make little contact on the sides where the sphere's surface touches the shell, for example, on the left side of the bottom left picture.
- This infill algorithm would be trivial to implement direct to G-code in slicer program, because of its regular and simple nature.

6.2 Truncated Octahedron Tessellation



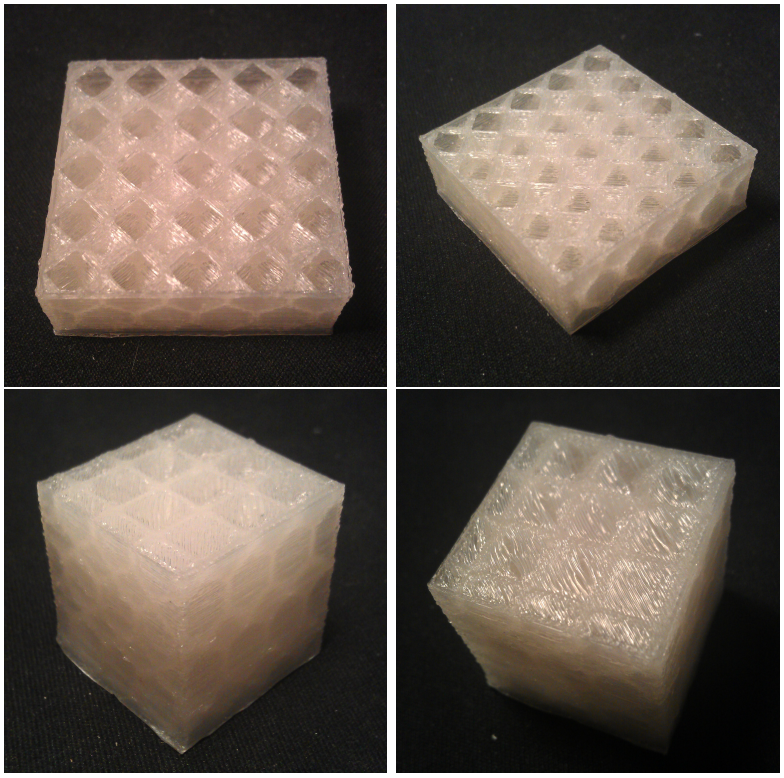
- The truncated octahedra mostly print quite well when hollow, tessellated and small, as seen in the top pictures. According to [8], the dihedral angle between a square face and a hexagonal face is $125°15'51''$. As the square is horizontal, the hexagonal face must be $125°15'51'' - 90° =$

$35°15'51''$ from the vertical, which is $90° - 35°15'51'' = 54°44'9''$ above the horizontal. This is well above the 45° rule and there the hexagonal faces print well.

- The top and bottom of the truncated octahedra are completely horizontal and therefore have some issues being printed. As seen in the top left picture, there are visible holes where the flat surfaces failed to bridge and collapsed. While the octahedra could be rotated slightly to ensure no surface is horizontal, some surfaces would still be far less than 45° degrees to the horizontal and would also fail to print well.
- Worst-case scenario the truncated octahedra are cut through their centres and leave gaps in regular octagon shapes of edge length a_o. As can be seen in bottom right, this caused some major issues while bridging the top layer. These gaps are much larger than other infills.
- Again, unlike the rectilinear pattern, this infill nicely supports the side walls of the model, leaving no long vertical gaps in support, as seen in bottom left.
- This infill algorithm would be fairly easy to implement direct to G-code in slicer program, because of its regular nature.

6.3 Rhombic Dodecahedron Tessellation

- The rhombic dodecahedra print very well. The orientation in which the rhombic dodecahedra are arranged had been chosen to ensure no faces were horizontal. This results in the faces of greatest slope being 45°. This is clear because each congruent rhombic face is parallel to its opposite face and also the length of a rhombic face away from it. Cutting the rhombic dodecahedron through its centre and the centre of four such faces will expose a square shaped section view. In the chosen orientation for printing, this square is rotated 45°, meaning each of its sides (and consequently the faces of the rhombic dodecahedron) are exactly 45° to the horizontal, which is just allowable by the 45° rule. In practice they print very well because of their size and compliance to this rule.
- In the worst case, the rhombic dodecahedra would be cut through their centre by the top layer, leaving a gap of support in the shape of the previously mentioned square. This square has an edge length of the longest dimension of one of the rhombic faces, which was denoted by a in the *Design & Analysis of Algorithms* section. As can be seen in the bottom left, the top layer prints almost perfectly, bridging excellently.
- This infill has some of the smallest gaps in support on the side walls of the model.



- This infill algorithm would be fairly easy to implement direct to G-code in slicer program, because of its regular nature.

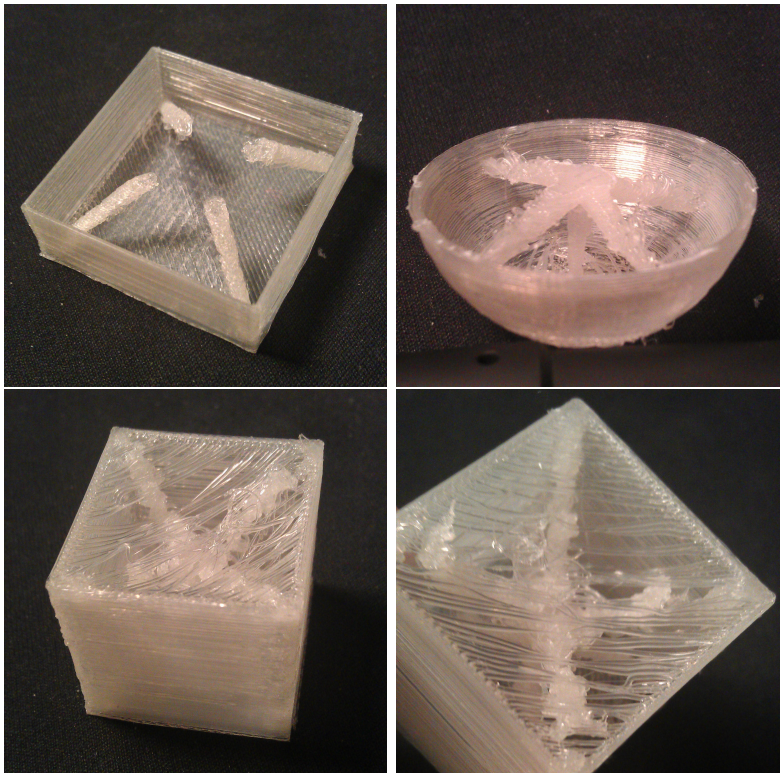
6.4 Dynamic Vertex Support

- My earlier statement that this infill algorithm would be much more useful for curved, organic and spherical models is justified in the bottom pictures. Because the cube's flat top face only contains vertices on its corners, the gap for bridging was far too large and the surface collapsed in. On top left there is a sectioned sphere which was printed with this infill quite successfully.

- The top left picture shows a cube stopped during the print. It shows that the struts are printed quite well, even at 45°. However, there may be cases in other prints where this angle is to small and the struts might not print.

- The fact that the side walls are not supported by this infill is another reason that it should only be used with curved surfaces.

- This infill algorithm would be particularly difficult to implement direct to G-code in slicer program.

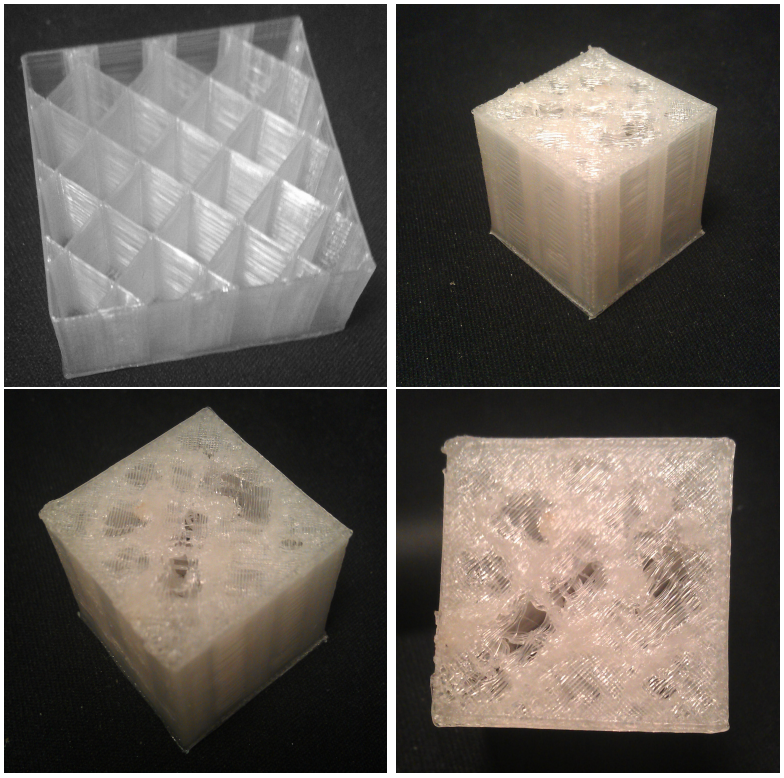


6.5 Rectilinear (Comparison)

- The gaps left by this infill caused some briding problems for the top layer, as shown on bottom left.
- On the print's side walls, long vetical bars of differing shade can be seen. These are large gaps where the infill does not touch or support the side walls, which would create weaker points here on the part.
- This infill prints almost twice as fast as the sphere packing and two tessellation infills. The reason for this is because of the relative simplicity of the rectilinear pattern versus the others. However, it does have the advantage of being directly output in G-code by the slicer, meaning the printer will have move efficient movements to print it. If the other infills were almost output by the slicer program straight into G-code they would be much more comparable to the rectilinear infill's speed.

6.6 Final Conclusion

The truncated octahedron infill and sphere packing both performed reasonably well, with some drawbacks, such as the truncated octahedron's bridging issues and the spheres' difficulty joining to each other.



The dynamic vertex support algorithm was an interesting experiment and while not performing particularly well, could certainly be improved in many different ways. For example, joining struts to faces, perhaps multiple times on each face, or maybe creating a set number of struts, which would then be evenly distributed.

It is clear that the rhombic dodecahedron tessellation print was the better performing infill pattern out of the four new infills. It does appear much stronger than the rectilinear infill, because of the lack of gaps along the side walls, and it is very easily printable. The uniform tessellation supports the model excellently, and overall this infill has exceeded my expectations.

I believe I have achieved my aim to prove the viability of 3D infill algorithms. To extend this work, the infills would need to be implemented in a slicer program, for two reasons.

Firstly, to increase the efficiency of the software. Currently, editing the input model and adding a 3D model of the infill is only possible for small, simple models, and takes a reasonably long amount of time to run. If the algorithms were part of a slicer program, they could be inserted into very complex models layer by layer.

Secondly, the time taken to print would be greatly reduced. The slicer program could effectively plan the most efficient path for the print head, as slicer programs do for currently existing infills like the rectilinear pattern.

Integration into a slicer program will now be much easier for the infills for which I derived density to cell size relations. I plan to some day extend my work by implementing the better performing algorithms in a slicer.

7 Acknowledgements

First and foremost I would like to thank Gary Hodgson, for inspiring this project. Without his article and subsequent correspondence, this project would not be.

I'd also like to thank Dr. Phil Maguire from the Departement of Computer Science, NUI Maynooth, for his help in the early stages of planning the project and the many discussions we had.

The RepRap project and its creator, Dr. Adrian Bowyer, for without which there would be no printer to test the infills on.

I would also like to thank my family, friends and my science teachers, for their time put into organising, proofreading and assistance with the exhibition.

8 References

[1] 3D Printing Era, *Printing Overhangs and Bridges*
http://www.3dprintingera.com/3d-printing-overhangs-and-bridges/

[2] Martin Baumers *Additive Manufacturing and 3D Printing Research Group, University of Nottingham*
https://www.nottingham.ac.uk/engineering-rg/manufacturing/3dprg/people/martin.baumers

[3] Gary Hodgson, *Thoughts on Fill Algorithms*,
http://garyhodgson.com/reprap/2012/01/thoughts-on-fill-algorithms/

[4] *The RepRap Project*,
http://reprap.org

[5] Slic3r Manual, *Infill Patterns and Density*,
http://manual.slic3r.org/InfillPatternsAndDensity.html

[6] Wolfram Mathworld, *Truncated Octahedron*,
http://mathworld.wolfram.com/TruncatedOctahedron.html

[7] Wolfram Mathworld, *Rhombic Dodecahedron*,
http://mathworld.wolfram.com/RhombicDodecahedron.html

[8] Wikipedia, *Truncated Octahedron*,
https://en.wikipedia.org/wiki/Truncated_octahedron