Adaptive layer heights in the FDM 3D printing process

Using 3D model complexity to automatically determine and adapt layer height

By Chris ter Beke

Abstract

Traditionally the FDM 3D printing process (or FDM for short) uses a fixed layer height throughout the construction of an object. This results in the same quality for the entire object regardless of surface details. To improve surface details the layer height for the whole model has to be decreased, resulting in a much longer print time. This research aims to use the shape of a 3D model to adapt the layer height and change it per layer. The goal is to get a surface quality close to that of a model that is entirely printed with a decreased layer height but without much of the added printing time. Finally a proof of concept is implemented in the Ultimaker Cura slicing software to bring the solution to market.

1 Introduction

In this section we will introduce the history and current state of 3D printing, the 3D printer manufacturer Ultimaker and the problem we are trying to solve.

1.1 FDM/FFF

3D Printing, or Additive Manufacturing, is a term that covers a lot of different technologies[1] which are involved in creating three-dimensional objects from a solid or liquid source material.

One of those technologies is Fused Deposition Modeling (FDM), or Fused Filament Fabrication (FFF). This is currently one of the more robust technologies and has evolved through several stages of open source and closed source development. It was originally invented by Scott Crump in the late 1980s and then patented by Stratasys[2]. When this patent expired in the early 2000s, Dr. Adrian Bowyer envisioned a low-cost FDM style 3D printer which he called RepRap[3].

1.2 Ultimaker

One of the largest manufacturers in the desktop segment is Ultimaker. Founded in 2011, the company expanded rapidly due to the unparalleled reliability of the Ultimaker Original 3D printer. The design was loosely based on the RepRap design, but with some important modifications to enhance print quality and speed. One of which was moving the extruder motor to the back of the printer and extending the feeding tube to the nozzle. This reduces weight on the XY carriage and thus allowing higher print speeds.

Two years later, the Ultimaker 2 was launched, which still is one of the most-selling 3D printers to date. It was still the same design, but had a more finished look and better electronics and firmware.

In the fall of 2016 the Ultimaker 3 was announced. This is the first printer from Ultimaker that features a dual extrusion setup, allowing for printing with multiple colors or materials, for example using a water soluble material to print support structures. While not

[1] https://www.iso.org/standard/69669.html
[2] https://www.google.com/patents/US5121329
[3] http://reprap.org/

being the first printer to have dual extruders, it worked far more reliable than the competitors. This makes the Ultimaker 3 the go-to desktop printer for makerspaces, SME and enterprise.

1.2.1 Cura

Cura is a free and open source software package that can convert 3D model files to G-code, a format that FDM 3D printers can understand. Cura was originally written by David Braam, who joined Ultimaker shortly after to continue to work on this slicer and other software for Ultimaker. Ultimaker kept Cura open source and has since then released a version 2.0 with a new interface that is targeted more towards beginner users. Currently a dedicated team of around 10 engineers and testers is working on Cura full-time.

1.3 Problem statement

Most slicers use a fixed layer height throughout the slicing process. Since layer height is one of the main factors for visual quality, this means that the entire printed object has the same quality, regardless of shape or complexity. Especially in curved areas this means that the layers can be very visible. There are two options to combat this: post-processing using sandpaper, or printing the entire model at a smaller layer height. Both are time consuming and not real solutions to the problem.

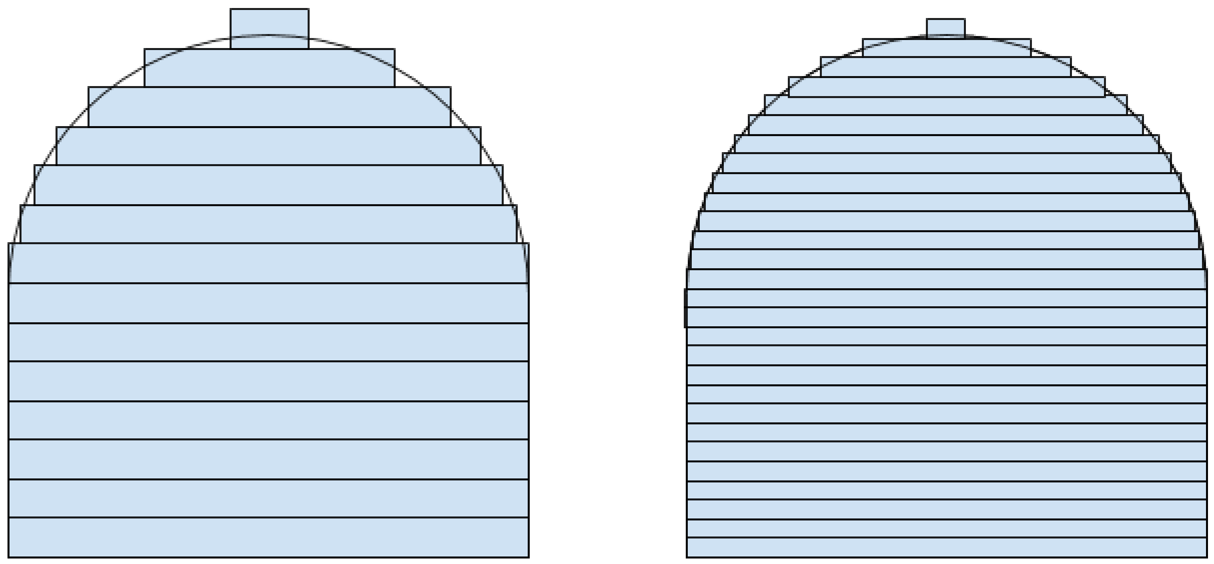


Figure n: Visible quality difference between larger and smaller layer heights.

The increased printing time of using a smaller layer height is a result of the fact that the time it takes to print a layer is defined by size of the layer, not by its height. So when reducing the layer height by a factor of two, the total print time is also increased by a factor of two. While this solves the problem for most curves, the steepest slopes will still show visible layering and need another decrease in layer height. This process can go on and on until the total print time is far beyond realistic expectations. Another side effect is that all straight sections are also printed with with smaller layers and these sections don’t need the increased quality.

Ultimaker is looking for a solution that improves visual quality only in sections of the print that require it and that does not add a lot of print time. A fully automated solution is preferred as the user should not need to have extra knowledge about which part of the model should be printed with increased quality and which should not.

2 Research questions

The following research questions are used to help us determine if the final prototype implementation was successful or not.

Research question
“Can we use the shape of a 3D model to adapt the layer height in order to improve the surface quality of the printed object without increasing the printing time too much?”

Secondary research question
“Does changing the layer height affect the print quality?”

3 State of the Art

In this section we will discuss similar technologies that are on the market and are used as inspiration for our own implementation. Notable is that these are open source and can be used and modified under the conditions of the original license. We will also discuss the Ultimaker 3, the machine that will be used for testing our algorithm.

3.1 Autodesk VariSlice™

Autodesk introduced the Ember 3D printing platform in 2016. They saw a possible improvement in DLP 3D printing by changing the quality depending on the shape of the 3D model. A proof of concept was written by several of their engineers and open sourced[4] under the name “VariSlice”.

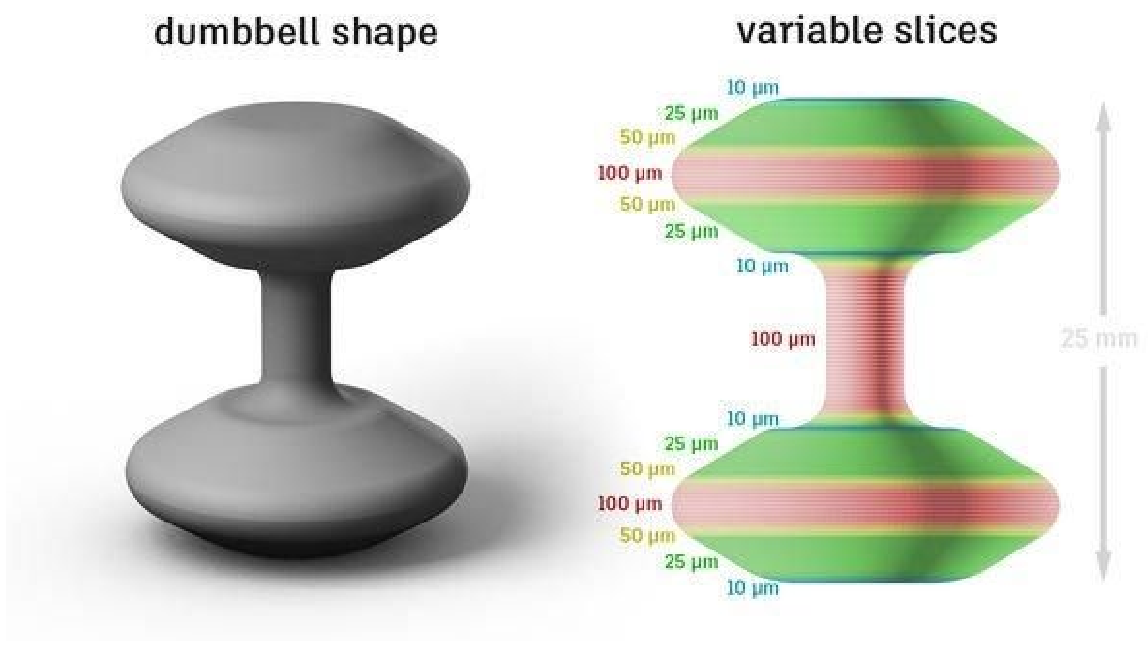


Figure [x]: Schematic representation of Autodesk VariSlice™.

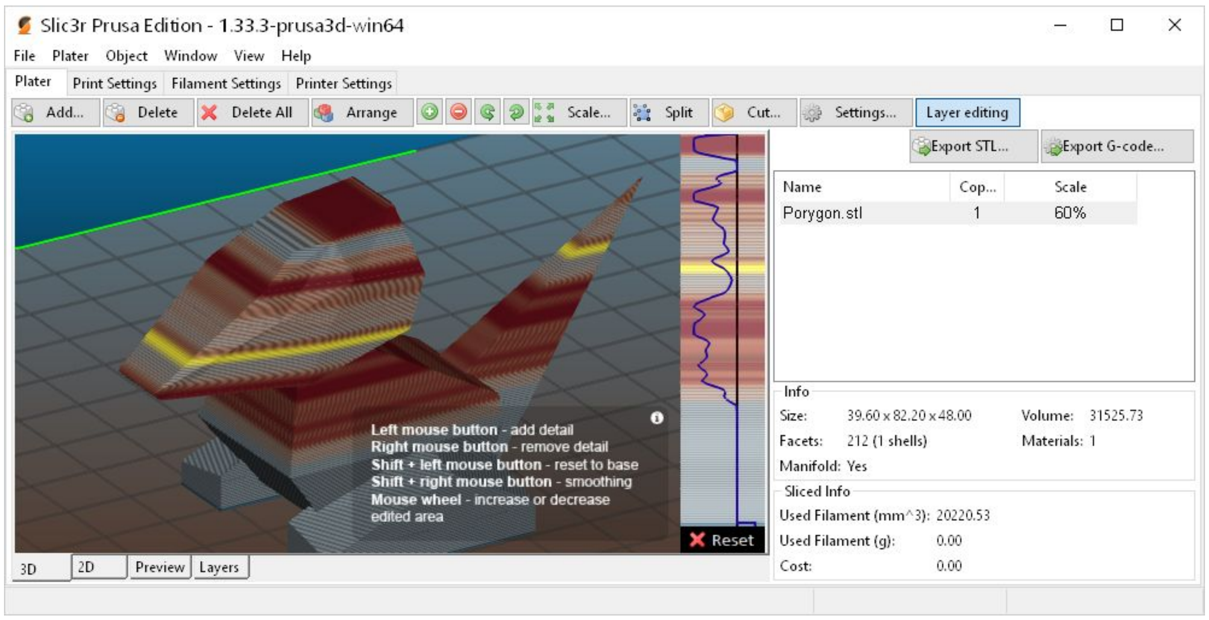
[4] Full source code available on http://www.instructables.com/id/Variable-Slicing-for-3D-Printing-on-Autodesk-Ember/

The main algorithm consists of two loops: one over the possible layer height range and one over the triangles of the 3D model that intersect a layer. It starts by finding all triangles that intersect a layer of the largest allowed height. From all those triangles it finds the one with the flattest slope, so the most horizontal triangle. If the layer height divided by the Tangent of that slope is larger than a predefined threshold, it will try a smaller layer height. This process continues until the threshold is met or the smallest allowed layer height is reached. Then it goes to the next layer, where the bottom of that layer is the top of the current layer. Relevant source code for this can be found in Appendix II.

Although the VariSlice code does the job, it is quite hard to understand due to lack of documentation and use coding standards. It is also not written in an optimal way, as many calculations are done several times while looping over all the triangles. Section 4.2 will describe further how the algorithm is optimized for our own implementation. It should also be noted that VariSlice is not at all an end-user application. Coding skills are needed to use the algorithm, and it only works for 3D printers using DLP technology.

3.2 Slic3r Prusa Edition

Slic3r Prusa Edition by the open source 3D printer manufacturer Prusa Research has introduced a variable layer height feature in 2017. Their solution uses a vertical bar in the user interface that the user can interact with to increase or decrease layer height. There is no automated detection of the required layer height per section. This means the user needs to know which sections of the model require a different layer height and what that layer height should be to get an optimal balance between visual quality and print time.


Slic3r Prusa Edition - 1.33.3-prusa3d-win64
File Plater Object Window View Help
Plater Print Settings Filament Settings Printer Settings
Add... Delete Delete All Arrange Scale... Split Cut... Settings... Layer editing
Export STL... Export G-code...
Reset
3D 2D Preview Layers

Left mouse button - add detail
Right mouse button - remove detail
Shift + left mouse button - reset to base
Shift + right mouse button - smoothing
Mouse wheel - increase or decrease edited area

Name	Cop...	Scale
Porygon.stl	1	60%

Info
Size: 39.60 x 82.20 x 48.00 Volume: 31525.73
Facets: 212 (1 shells) Materials: 1
Manifold: Yes

Sliced Info
Used Filament (mm^3): 20220.53
Used Filament (g): 0.00
Cost: 0.00

Figure [x]: Screenshot of Slic3r Prusa Edition variable layer height.

Prusa's implementation is variable layer heights is very visual. By default, all the layers have the same height. The user can bring up an interface element in which the layer height can be tweaked per section. These tweaks are also visualised on the 3D model so the result can be verified. Tweaking can be done by clicking the left and right mouse buttons on the variable layer height interface. This will raise or lower the layer height at the position of the mouse cursor. This results in a mountain-like graph as can be seen above.

Notable is the forced use of smooth transitions between layer heights. This is done to prevent sudden changes in flow rate from one layer to the next. Abrupt changes in flow rate can reduce the visual quality of the printed model, as well as reduce layer adhesion and thus part strength. By transitioning between the chosen heights slowly, these side effects can be minimized.

3.3 Ultimaker 3

The Ultimaker 3 will be used to test our slicing results on, so it is worth mentioning several properties of this machine. The preset base profiles available in Cura for the Ultimaker 3 will be used as a baseline for our slicing settings. These are important as most users use one of these preset profiles.

The Ultimaker 3 is a Bowden style 3D printer. This means that the extruder motor, the motor that pushes the filament through the hot end, is not attached to the print heat but to the back of the machine. A Bowden tube between the extruder motor and the print head ensures that the filament is pushed through correctly. This system has advantages and disadvantages compared to a so-called direct drive system, where the extruder motor is directly attached to the print head. The main benefit is reduces weight on the printhead, allow for faster movement and printing speeds. The biggest drawback is the delay in extrusion and retraction of the actual filament compared to the movements of the extruder motor. These delays are caused by the distance and tension in the tube.

Our variable layer height algorithm has to take this effect into account, as any changes in layer height result in changes of flow rate. When changing the flow rate, the potential energy in the filament that is in the bowden tube changes. When doing this too often, the material will degrade before entering the hot end, resulting in a poor print quality. Ultimaker is doing more research on material behaviour in the bowden tube, but this is out of the scope of this project.

4 Ideation

4.1 Ultimaker R&D

To gain more insight in what is required to get to a good slicing algorithm for variable layer height, several meetings were held with Ultimaker staff focussing on different disciplines. These range from user experience, relevant material processing settings (like flow rate), hot end thermodynamics and business cases.

4.1.1 Material processing

Several employees of the Materials & Processing team were interested in this research project. They had concerns about the behaviour of the material when subjected to many changes in flow rate during a short period (several layers) of time. This is usually not a problem as the same layer height is used throughout the printing process, and the flow rate only changes when changing the print speed. With the flow rate changes as a result of layer height variation added, the material can lose the required characteristics for a successful print. A visual example of this is color. The following images show the variation of color when changing the layer height during one of the early test prints.

[INSERT IMAGE]

Next to color, also layer bonding is affected. This is a more serious problem as reduced layer binding results in reduced part strength. In the end a model could break where two layers with a too big difference in layer height are touching.

4.1.2 Thermodynamics

A team member of the hardware R&D team is working on a thermic model of the print core (the Ultimaker 3's hot end). The behaviour of the material inside the print core is not well known and creating a mathematical model can improve the quality of the default settings used for slicing as it is easier to predict how the material will exit the print core onto the 3D model. Unfortunately further details of this research cannot be shared.

4.1.3 User experience

The Cura team has one UI/UX focussed team member and together with him we discussed how the feature can be controlled by the end user. Given that the layer height of the selected preset profile should become the center of the layer height range we defined three parameters that can be controlled by the user: maximum variation (in either direction), step size (layer height change per layer change) and threshold. This threshold is an arbitrary number that can be used to influence how quickly the layer height will change when the shape of the model changes. A lower threshold value will result in a slower adoption rate, and overall thicker layers. Visuals of differences between several threshold values can be found in section 6.1.

4.1.4 Business cases

Lastly the product owner of Cura was interested in the user value that this feature would add. At Ultimaker we always try to only add features that have a direct and positive impact on our end users. We defined several use cases that this feature could add benefit to:

- Mini figures; Mini figures (like game models) have many curves and are usually printed at small layer heights because the most important aspect is visual quality. The overall printing time could be reduced by only printing the curved sections at small layer heights. This result in similar visual quality but lower printing times.
- Functional parts with screw holes; Function parts with screw holes can greatly benefit from adaptive layer heights. The sections around the holes can be printed with smaller layers to achieve a better fit of the screws, while everything else can be printed at a larger layer height to decrease the printing time.
- Large objects with occasional detailed sections; Printing time can be significantly reduced for large objects that only require a small layer height at a few specific sections, for example custom bottle designs.

4.2 Optimizing the algorithm

While dissecting the VariSlice™ algorithm, it was quite obvious that it could be optimized a lot. Several calculations were done multiple times, for example the slope of a triangle in a potential layer. The following changes were made to the original algorithm when porting it to Python (NumPy) for the proof of concept and then C++ for the final implementation. The following figure can be used as reference to what is meant by the bottom and top of a layer, and its intersection with a triangle.

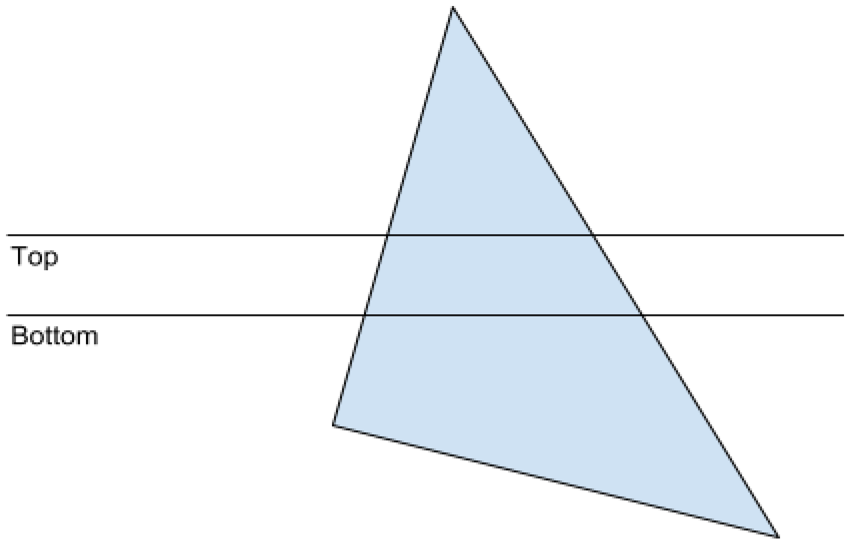


Figure [x]: Intersection of a triangle with the bottom and top of a layer.

- All triangle slopes are pre-calculated in a separate loop. We already know that each triangle will be used at least once, but most will be used multiple times (for each intersecting layer). By pre-calculating all the slopes and storing those in an array with the same indexing as the vertex index of the 3D model we only have to calculate the slopes once.
- Deciding whether a triangle is of interest for a potential layer was done every time while looping of the allowed layer heights, even though it's already known that triangles that don't fall in the thickest layer will for sure not fall in smaller layers in subsequent tries. By reducing the set of triangles of interest after the first iteration to only the triangles found for the thickest layer, calculating the triangles of interest for smaller heights of that potential layer goes much faster.

- Detecting whether a triangle intersects a layer can be done much simpler. The original algorithm split these out into four cases: a triangle lies entirely above the layer, entirely below the layer, within the layer, or is taller than the layer. These four cases can be reduced to only two states: the top most vertex of the triangle lies above or on the bottom of the potential layer, and the bottom most triangle lies below or on the top of the potential layer. This greatly improves the calculation of finding triangles of interest.
- The Z value of the top and bottom vertex of each triangle is stored in an indexed array when calculating the triangle slope. This array can be referenced when looping of the potential layer heights to find triangles of interest, as opposed to calculating the Z value every time it's needed in that loop.

The final algorithm now works like this:

1) The allowed layer heights are calculated from the user input. These are defined by the default layer height from the profile, the maximum variation in each direction and the step size. This results in an array of layer heights between the minimum and maximum variation with steps of the given step size. For example with an input of 150 microns as default layer height, a maximum variation of 50 microns and a step size of 10 microns, the resulting array would be [100, 110, 120, ..., 200].
2) For all the triangles in the 3D model the slopes are calculated and stored in an array with the same indexing as the 3D model faces. This is done by taking the normal in the Z direction from the 3 vertices of a triangle and converting that to the inverse cosine of its absolute value. We also skip any input that is not a printable 3D model (e.g. support structures).
3) We manually add the first layer at all times, as the layer height of that layer needs to be a fixed value to ensure proper print bed adhesion.
4) Now we loop as long as there are triangles of interest for the layer we're trying to determine. This starts with layer 2, the first layer above the fixed first layer.
5) We try the tallest layer height. We compare the layer height divided by the tangent of the steepest slope to an arbitrary threshold value. If the threshold is not exceeded, we can add this layer height to the list of layer heights to use. If it exceeds the threshold, we try again with the next allowed layer height. This is the maximum layer height

minus the step size. This process continues until a layer height is found that meets the threshold criteria, or until the smallest allowed layer height is reached.

6) We now go to the next layer, where the bottom position of the layer is the top of the previous layer. This time we start with the layer height that is one step above the layer height used in the previous layer. This is needed to ensure a gradual change of layer height between layers, otherwise the print quality would suffer due to sudden material flow changes.
7) This process continues until the top of the model is reached (meaning there are no more triangles that intersect with the layer we're trying to calculate). Now all the calculated layer heights are send to the next step in the slicing engine, which will use these instead of a fixed layer height to generate polygons per layer by intersecting the 3D model with a 2D plane at the correct height.

5 Realisation

In this section details about the prototyping, final implementation and development process are given. There are also screenshots of objects printed during development.

5.1 Criteria

Based on conversations with Ultimaker team members, community members and industry needs, the following criteria are needed to make adaptive layers a feature that is significant enough to be released in an upcoming version of Cura.

- It should automatically detect where to use which layer height, user interaction should only be needed when tweaking the auto detection settings in the experimental phase.
- The layer height cannot change too drastically from one layer to the next without affecting print quality. To compensate for this, a gradual change is needed. A setting could be added to give the maximum layer height change from one layer to the next (the step size).
- Every 3D model is different and requires different adaptive layer height settings. The user should be able to tweak the maximum variation from the base layer height and how quickly the layer height will adapt to the surface angle (the threshold).
- The user should be able to preview the layer height in the 3D viewer before starting the print. A new color scheme should be added to the 3D viewer that shows which sections have which layer height.

5.2 Development

At Ultimaker, all development is done using agile sprints. This means that a feature has to be scoped that it can be (partially) finished in a single print. Adaptive layers would be too much to finish in a single sprint, so the work was split up into two sprints.

In the first sprint a prototype was built in Python to proof that we could indeed automatically detect which section of a model required a certain layer height. A Cura plugin[5] was developed that scanned the 3D model and returned a list of layer heights for that model, which was then manually checked for correctness. Since Python is an interpreted language and can be quite slow, the implementation was done using NumPy, a Python library that executes mathematical functions in Fortran and C++ behind the scenes. This made it fast enough to test larger models with the prototype.

During the second sprint, the final implementation was done and tests were conducted together with the materials team. The algorithm was ported to native C++ and implemented in the slicing engine of Cura: CuraEngine[6]. The required settings were added to the front-end application under the experimental section and passed to the CuraEngine when slicing. Whenever adaptive layers is enabled, the CuraEngine will first calculate the required layer heights instead of using a fixed layer height.

After this implementation was done and released in an internal test build of Cura 3.2, the materials team started experimenting with the feature in order to come to reasonable default values for the newly added settings. This is a process that is done for all settings in Cura and ensures that the end user does not have to tweak the settings themselves in most cases.

Another member of the Cura team worked on implementing the new color scheme in the 3D viewer that can be used to preview the layer heights before starting the print.

[5] https://github.com/ChrisTerBeke/CuraVariSlicePlugin

[6] https://github.com/Ultimaker/CuraEngine/blob/master/src/settings/AdaptiveLayerHeights.cpp

5.3 Screenshots

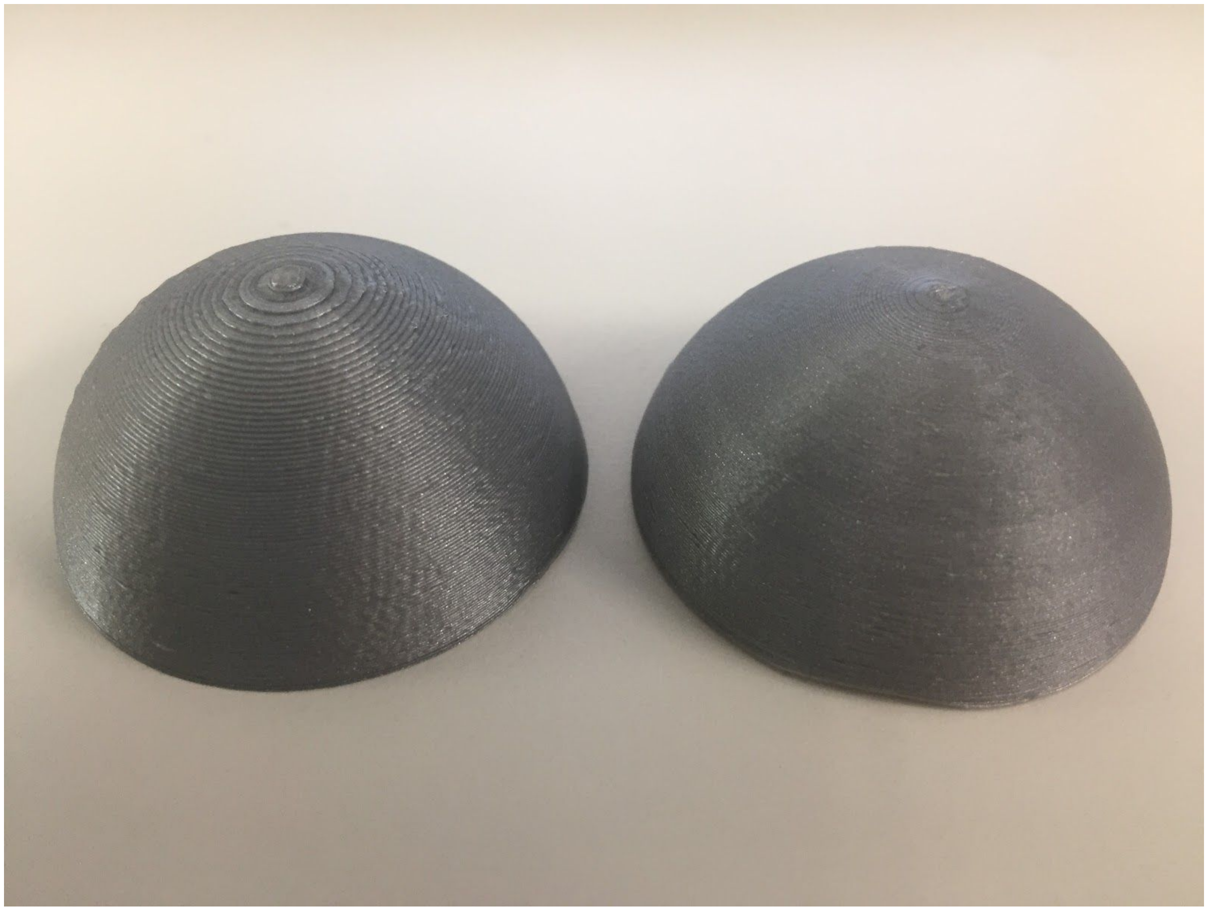


Figure [x]: First test prints from the prototype Python implementation.

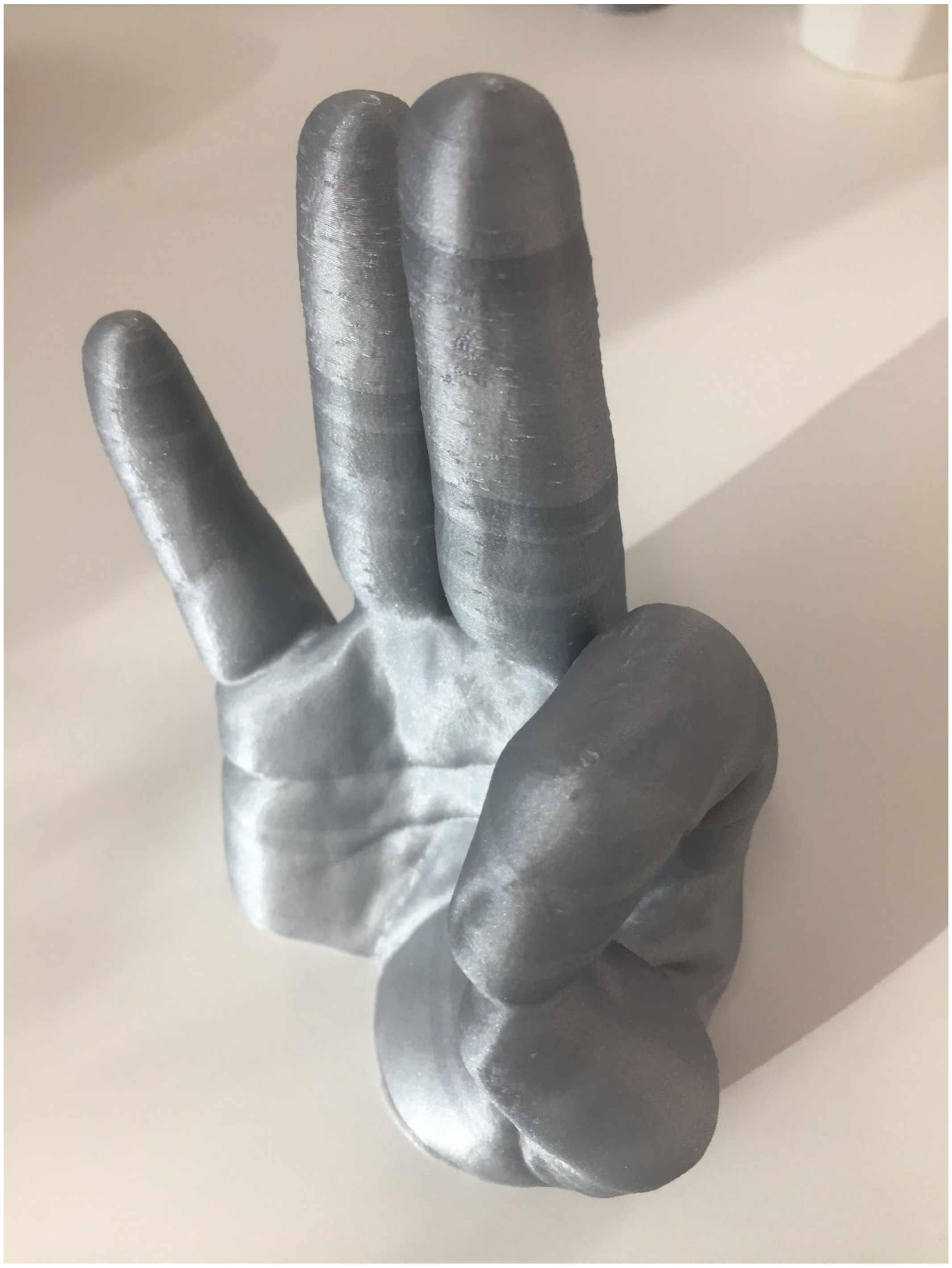


Figure [x]: Larger test print from the first final implementation in CuraEngine. Note the smoothed surfaces on top of the fingers.

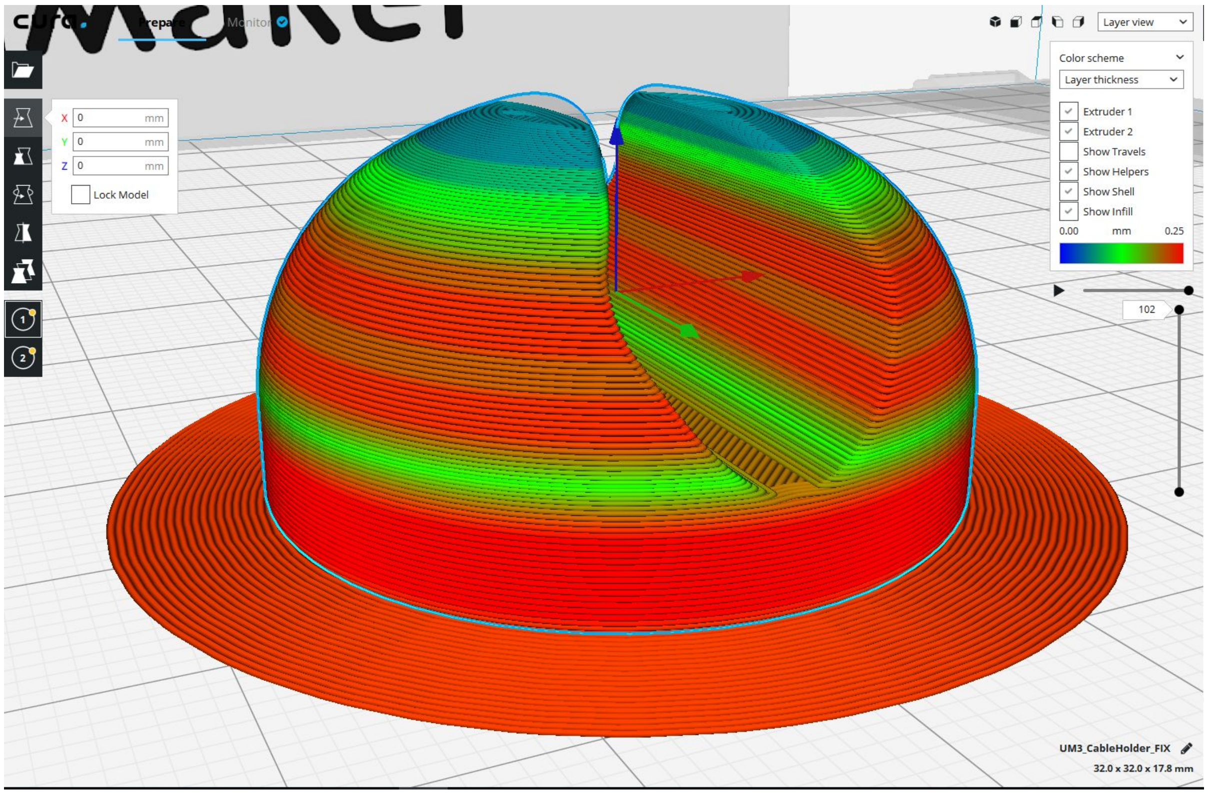
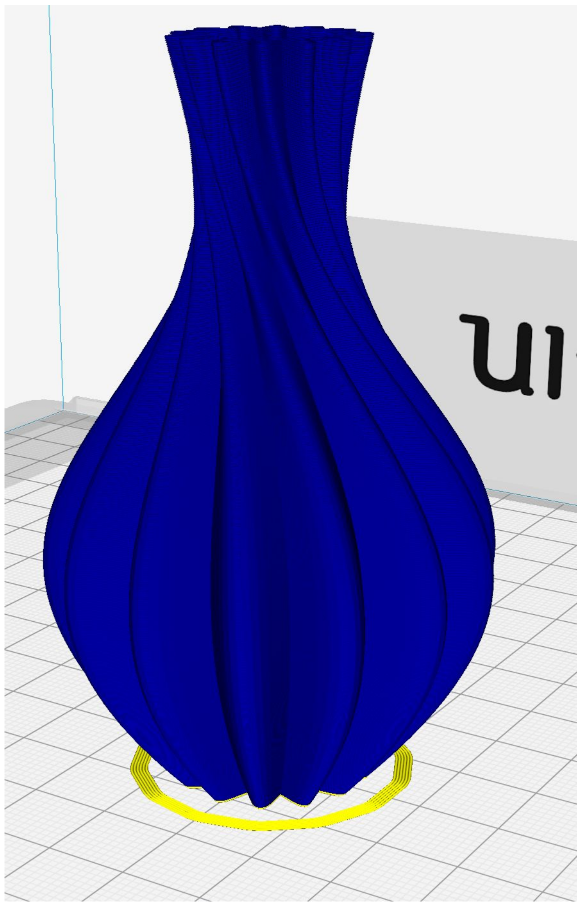
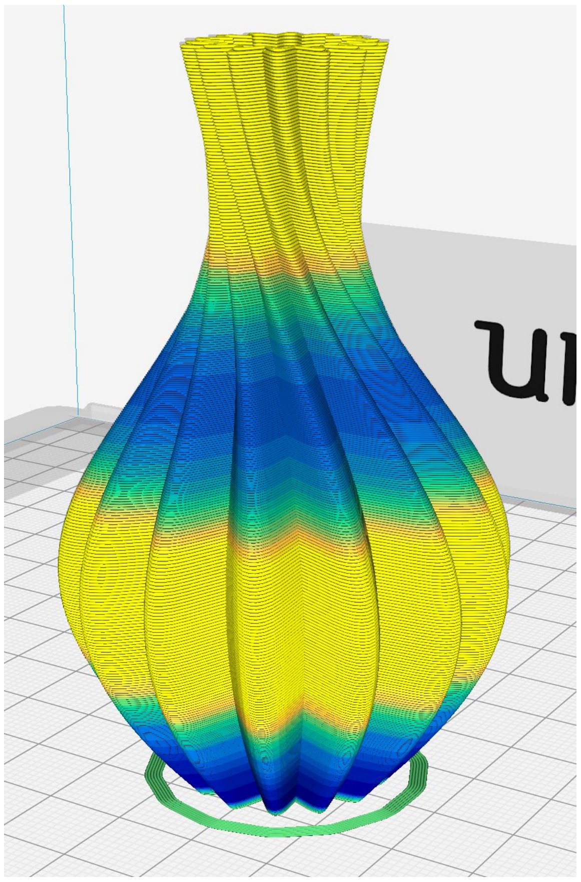


Figure [x]: Screen capture of the final CuraEngine implementation using the new color scheme. Red is the tallest layer height, blue is the thinnest.

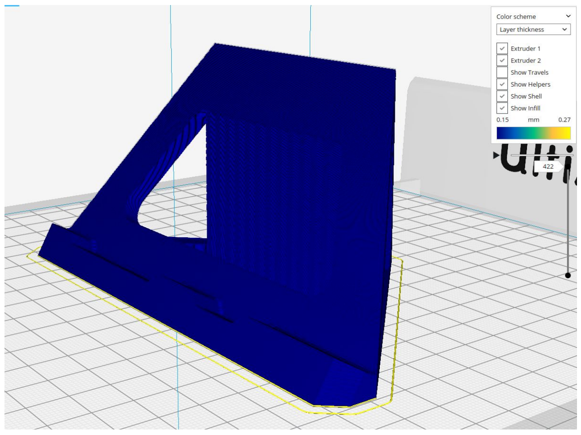
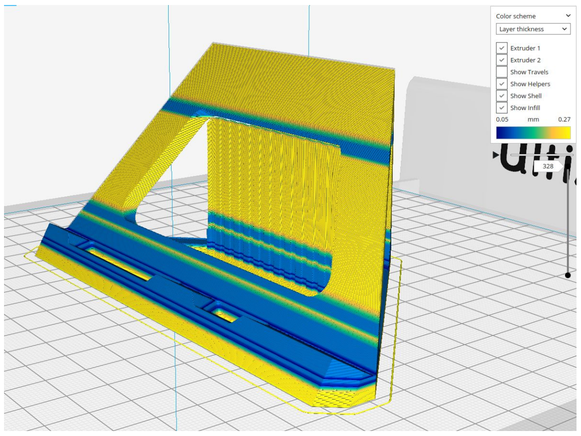
6 Evaluation

We will do three test prints using representative 3D models. Each test print will be done using a standard 100 micron layer height and a adaptive layer height from 50 to 350 microns. This means that the model with adaptive layer heights will not only have a shorter printing time but also improved visual quality on the parts that need it. For each test print the Cura 3D preview, final print result and print times are recorded.

Object 1: Vase

Normal mode	Adaptive mode
	
Time: 9 hours and 5 minutes	Time: 2 hours and 53 minutes
Layer height: 100 microns	Layer range: 50 microns to 350 microns
	68.3% decrease in printing time

Object 2: Smartphone stand

Normal mode	Adaptive mode
 Color scheme Layer thickness Extruder 1 Extruder 2 Show Travels Show Helpers Show Shell Show Infill 0.15 mm 0.27 422	 Color scheme Layer thickness Extruder 1 Extruder 2 Show Travels Show Helpers Show Shell Show Infill 0.05 mm 0.27 328
Time: 5 hours and 56 minutes	Time: 5 hours and 4 minutes
Layer height: 150 microns	Layer range: 50 microns to 250 microns
	10.4% decrease in printing time

6.1 Answer to research questions

Research question
“Can we use the shape of a 3D model to adapt the layer height in order to improve the surface quality of the printed object without increasing the printing time too much?”

Given the results above we can safely say that it is possible to use the surface geometry of a 3D object to determine the layer height for each part of the model. By doing that, we can adapt the layer height to the surface angle and get a better visual quality for that surface. At the same time, it is a huge timesaver for sections that don’t need a small layer height, sometimes even up to 70 per cent reduction of printing time. On average, the printing time reduction seems to lie between 10 and 20 per cent.

Secondary research question
“Does changing the layer height affect the print quality?”

Varying the layer height too much or too sudden will affect print quality. According to Ultimaker material researchers, the layer bonding will decrease. Also the material color might change slightly because the light bounces back differently depending on layer height. A possible way to fix this would be to also change the flow rate and print head temperature.

7 Conclusion and discussion

The feature has been released to Cura users in version 3.2 and was received very well. Several users have already shared their usage of the feature on the Ultimaker forum and Twitter. Although some users have trouble understanding the settings at first, some very nice prints were made using adaptive layers. Given the fact that the feature works, mostly automatically, and the print quality is not affected too much we think the research and implementation of adaptive layers in Cura is a success.

7.1 Recommendations

The following recommendations can be implemented by the Cura team in the future and would improve adaptive layers as well as print quality.

- Allow different step size and threshold values for surfaces angled outwards (overhangs) versus surfaces angled inwards (top surfaces). This allows for better overhang settings as those require layer layers to prevent falling over.
- Allow different step size and threshold values when increasing layer height versus increasing layer height.
- Change the settings so that a maximum and minimum layer height can be entered instead of a variation from the base layer height. This might be tricky however since the base layer height comes from the selected print profile and a too large deviation from that as the middle of the layer height spectrum might result in bad print quality.
- Automatically change the temperature and flow rate when the layer height changes. This requires deep modifications in CuraEngine, but will result in better print quality.

7.2 Related content

After the initial release of adaptive layers in Ultimaker Cura 3.2, the 3D printing community has been experimenting with the feature. The following is an overview of interesting community threads about using and improving the feature.

- Ultimaker community forums “Adaptive Layers”: https://community.ultimaker.com/topic/21520-adaptive-layers/
- Ultimaker community forums “A look at adaptive layers”: https://community.ultimaker.com/topic/21706-a-look-at-adaptive-layers/
- Ultimaker blog post “Discover Ultimaker Cura 3.2”: https://ultimaker.com/en/blog/52484-discover-ultimaker-cura-32
- firstlayer.co “Adaptive layers will save your time at 3D printing”: http://firstlayer.co/adaptive-layers-will-save-time-3d-printing/

8 Acknowledgements

I would like to thank Ultimaker for allowing me to spend time on this research. I would also like to thank Mark Suurmond in particular for allowing me to finish the feature in the sprint following the research sprint.
Next I would like to thank Erik Faber for coming up with this internship as a way to finish my studies of Creative Technology.
Finally I would like to thank Edwin Dertien from the University of Twente as supervisor of this internship.

9 Bibliography

Wikipedia - Fused Filament Fabrication, https://en.wikipedia.org/wiki/Fused_filament_fabrication (February 2018)

Intructables - Variable slicing for 3D printing on AutoDesk Ember, http://www.instructables.com/id/Variable-Slicing-for-3D-Printing-on-Autodesk-Ember/ (February 2018)

ISO - Additive Manufacturing, https://www.iso.org/standard/69669.html (February 2018)

Bowyer, A., Jones, R, RepRap - The Replicating Rapid Prototyper, *Cambridge University Press - Robotica*, http://reprap.org/mediawiki/images/d/da/Jones-et-al-paper.pdf (Nov 2009)

10 Appendices

Appendix I: Terminology

Term	**Description**
FDM/FFF	Fused Deposition Modeling / Fused Filament Fabrication, the process of building a 3D object by extruding melted plastic layer by layer.
Extruder	The part of an FDM 3D printer that pushes the filament through the hot end
Hot end	The part of an FDM 3D printer that heats up the filament so it can extruded through the nozzle.
Nozzle	The part of an FDM 3D printer that compresses the filament to a smaller diameter to it can be laid down precisely.
Filament	The material used by an FDM 3D printer to build a 3D object. Usually in the form of a spool of 0.5 to 1 Kg.
Slicer	Software that converts a 3D model into machine readable G-code.
G-code	Machine code that 3D printers and CNC machines use to execute instructions needed to replicate the original design.

Appendix II: VariSlice algorithm

This is an excerpt from the VariSlice source code. The full code can be downloaded at http://www.instructables.com/id/Variable-Slicing-for-3D-Printing-on-Autodesk-Ember/.

```
void buildLayers() {
    // Knowing the height of the STL and the smallest the layer sizes can be, we can
find out the maximum number of slices possible.
    // We'll create our arrays this size so they shouldn't go out of bounds, but will
have excess empty slots
    float minLayerSize = min(variableLayers);
    stlHeight = calcSTLheight();
    int maxLayers = int(stlHeight/minLayerSize);
    maxLayers *=2;
    println("maxLayers: " + maxLayers);

    // Create an arraylist of triangles to temporarily hold triangles of interest
    ArrayList<Triangle> tempTriangles = new ArrayList<Triangle>();

    layerHeights = new float[maxLayers];
    absoluteHeights = new float[maxLayers];
    minSlopes = new float[maxLayers];
    triInLayers = new int[maxLayers];

    float zLevel;
    float zInspect;

    for (int k = 0; k < maxLayers; k++){
        if (k==0) {
            zLevel =0;
        } else {
            zLevel = absoluteHeights[k-1];
        }

        // j iterates through the list of layer thicknesses that we'll consider
        for(int j = 0; j< variableLayers.length; j++){

            zInspect = variableLayers[j];

            // i iterates through every triangle to see if it is within the window/range
of interest
            tempTriangles.clear();
            for(int i = 0; i < triangles.length; i++){
                if(isTriangleInRangeOfInterest(triangles[i], zLevel, zLevel+zInspect) ==
true){
                    tempTriangles.add(triangles[i]);
                }
            }

            float layerSlope = findMinSlope(tempTriangles);

            // Is this layer thickness small enough? If so build it! Or if we're on the
thinnest layer thickness possible, build it.
            // Assumes variable layers go from thick to thin.
            if(smallEnough(layerSlope, variableLayers[j], stepoverThreshold) == true ||
j == variableLayers.length-1){
                layerHeights[k] = variableLayers[j];
                absoluteHeights[k] = zLevel + variableLayers[j];
                minSlopes[k] = layerSlope;
                triInLayers[k]=tempTriangles.size();
                break;
            }
            tempTriangles.clear();
        }
        if(tempTriangles.size()==0){
            break;
        }
        tempTriangles.clear();
    }
} // end Build layers
```

Appendix III: CuraEngine adaptive layers algorithm

The following is an excerpt from the final algorithm used in CuraEngine to determine adaptive layer heights. The full source code can be found at https://github.com/Ultimaker/CuraEngine/blob/master/src/settings/AdaptiveLayerHeights.cpp.

```
void AdaptiveLayerHeights::calculateLayers()
{
    const int minimum_layer_height =
*std::min_element(this->allowed_layer_heights.begin(),
this->allowed_layer_heights.end());
    SlicingTolerance slicing_tolerance =
this->mesh_group->getSettingAsSlicingTolerance("slicing_tolerance");
    std::vector<int> triangles_of_interest;
    int z_level = 0;
    int previous_layer_height = 0;

    // the first layer has it's own independent height set, so we always add that
    z_level += this->initial_layer_height;

    auto * adaptive_layer = new AdaptiveLayer(this->initial_layer_height);
    adaptive_layer->z_position = z_level;
    previous_layer_height = adaptive_layer->layer_height;
    this->layers.push_back(*adaptive_layer);

    while (!triangles_of_interest.empty() || this->layers.size() < 2)
    {
        double global_min_slope = std::numeric_limits<double>::max();
        int layer_height_for_global_min_slope = 0;
        // loop over all allowed layer heights starting with the largest
        bool has_added_layer = false;
        for (auto & layer_height : this->allowed_layer_heights)
        {
            // use lower and upper bounds to filter on triangles that are interesting
for this potential layer
            const int lower_bound = z_level;
            // if slicing tolerance "middle" is used, a layer is interpreted as the
middle of the upper and lower bounds.
            const int upper_bound = z_level + ((slicing_tolerance ==
SlicingTolerance::MIDDLE) ? (layer_height / 2) : layer_height);

            if (layer_height == this->allowed_layer_heights[0])
            {
                // this is the max layer thickness, search through all of the triangles
in the mesh to find those
                // that intersect with a layer this thick
                triangles_of_interest.clear();

                for (unsigned int i = 0; i < this->face_min_z_values.size(); ++i)
                {
                    if (this->face_min_z_values[i] <= upper_bound &&
this->face_max_z_values[i] >= lower_bound)
                    {
                        triangles_of_interest.push_back(i);
                    }
                }
            }
            else
            {
                // this is a reduced thickness layer, just search those triangles that
intersected with the layer
                // in the previous iteration
                std::vector<int> last_triangles_of_interest = triangles_of_interest;

                triangles_of_interest.clear();

                for (int i : last_triangles_of_interest)
                {
                    if (this->face_min_z_values[i] <= upper_bound)
                    {
                        triangles_of_interest.push_back(i);
                    }
                }
            }

            if (triangles_of_interest.empty())
            {
                break;
            }

            double minimum_slope = std::numeric_limits<double>::max();
            for (auto & triangle_index : triangles_of_interest)
            {
                double slope = this->face_slopes.at(triangle_index);
                if (minimum_slope > slope)
                {
                    minimum_slope = slope;
                }
            }
            double minimum_slope_tan = std::tan(minimum_slope);
            if (global_min_slope > minimum_slope)
            {
                global_min_slope = minimum_slope;
                layer_height_for_global_min_slope = layer_height;
            }
            bool has_exceeded_step_size = false;
            if (previous_layer_height > layer_height && previous_layer_height -
layer_height > this->step_size)
            {
                has_exceeded_step_size = true;
            }
            else if (layer_height - previous_layer_height > this->step_size &&
layer_height > minimum_layer_height)
            {
                continue;
            }

            // we add the layer in the following cases:
            // 1) the layer angle is below the threshold and the layer height difference
with the previous layer is the maximum allowed step size
            // 2) the layer height is the smallest it is allowed
            // 3) the layer is a flat surface (we can't divide by 0)
            if (minimum_slope_tan == 0.0
                || (layer_height / minimum_slope_tan) <= this->threshold
                || layer_height == minimum_layer_height
                || has_exceeded_step_size)
            {
                z_level += layer_height;
                auto * adaptive_layer = new AdaptiveLayer(layer_height);
                adaptive_layer->z_position = z_level;
                previous_layer_height = adaptive_layer->layer_height;
                this->layers.push_back(*adaptive_layer);
                has_added_layer = true;
                break;
            }
        }

        // stop calculating when we're out of triangles (e.g. above the mesh)
        if (triangles_of_interest.empty())
        {
            break;
        }
        // we cannot find a layer height that has an angle lower than threshold.
        if (!has_added_layer)
        {
            z_level += layer_height_for_global_min_slope;
            auto * adaptive_layer = new
AdaptiveLayer(layer_height_for_global_min_slope);
            adaptive_layer->z_position = z_level;
            previous_layer_height = adaptive_layer->layer_height;
            this->layers.push_back(*adaptive_layer);
        }
    }
}
```