



MODELING AND TOOLPATH GENERATION FOR CONSUMER-LEVEL 3D PRINTING



H. Quynh Dinh
Filipp Gelman

Sylvain Lefebvre
Frédéric Claux





COURSE OVERVIEW

Goal

Learn about what's involved in developing a slicer - issues, solutions, and open questions in printer toolpath generation

Schedule

Session 1: 3D Printing With Plastic Filament

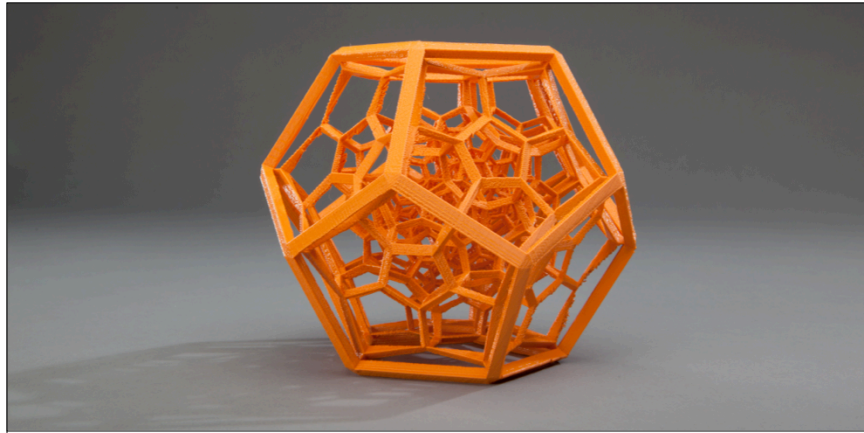
Session 2: Modeling for Filament-based 3D Printing



COURSE OVERVIEW

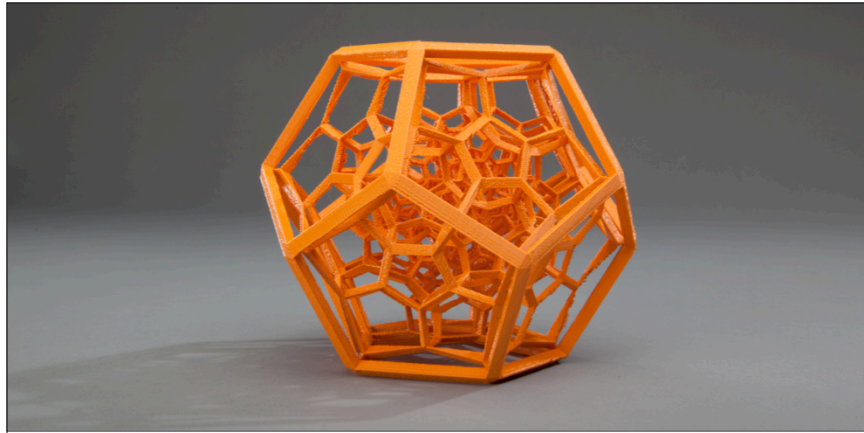
For updates, please visit:

<http://webloria.loria.fr/~slefebvr/sig15fdm/>



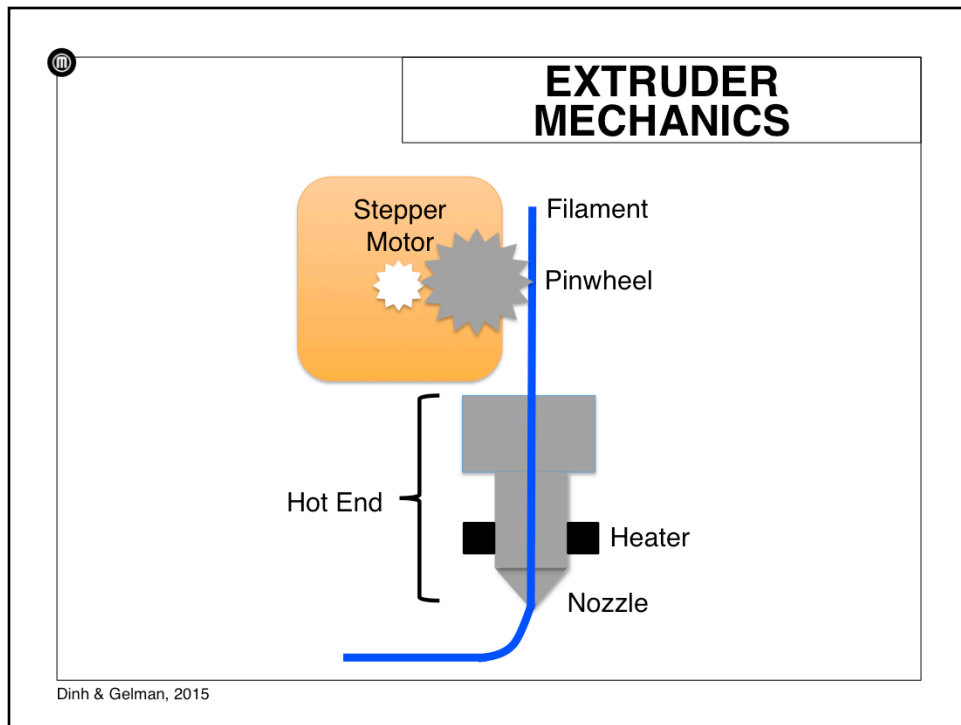
3D PRINTING WITH PLASTIC FILAMENT

8:35am – 10:15am



MATERIAL PROPERTIES AND MECHANICAL PROCESS

3D Printing with Plastic Filament




Stepper motor drives a pinwheel which forces plastic filament down into the hot end where a heater melts the plastic.

Melted plastic is extruded through a nozzle of some diameter (Makerbot's is 0.4mm).

References:


http://reprap.org/wiki/File:Extruder_lemio.svg

http://en.wikipedia.org/wiki/Plastics_extrusion



AMOUNT OF PLASTIC EXTRUDED

- Modeling volume of plastic through nozzle
 - Filament and nozzle diameters
 - Speed of extrusion
- Pinwheel slip results in under extrusion
- Modeling extruded filament profile



Dinh & Gelman, 2015

Print quality is directly tied to the amount of plastic being extruded – Too little, and you have under-extrusion effects (brittle, thin infill). Over-extrude, and you get blobs. So, we try very hard to estimate the amount of plastic that is going through the nozzle.

Pinwheel slip: This is when the pinwheel that is extruding the filament slips on the filament. This typically occurs at higher speeds.

The filament coming in typically has a circular profile, but as it is melted and laid down next to an adjacent line of extruded filament it takes on a different profile – e.g., compressed cylinder, rectangle, square. How you model this profile can affect your estimation of the volume of plastic being extruded.



PRINTER MECHANICS

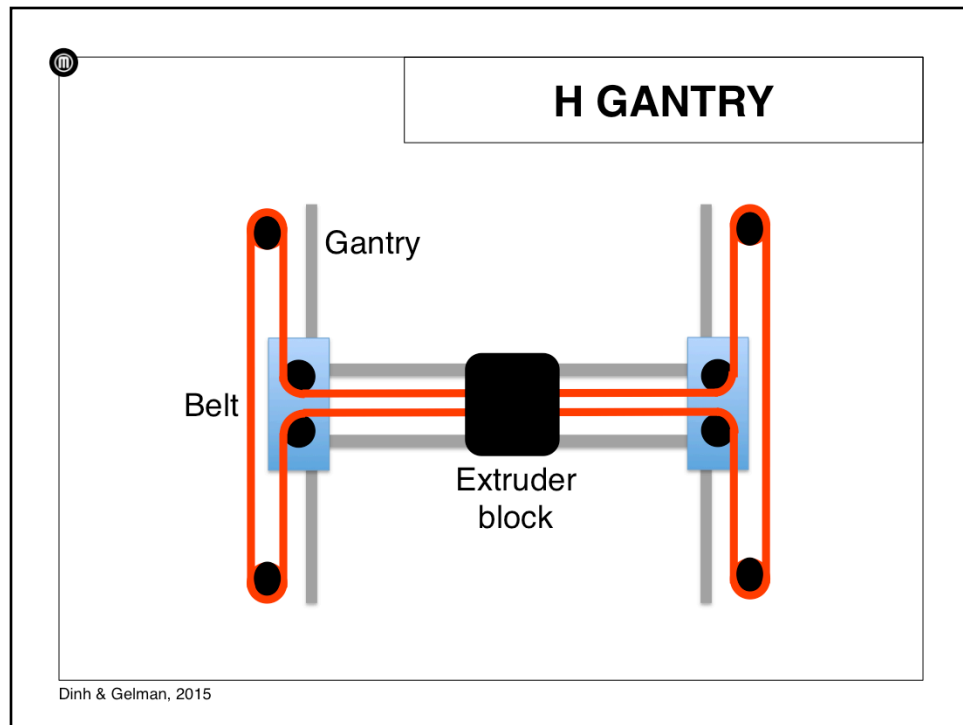
- Gantry type
 - H gantry
 - X-Y gantry
- Backlash: loss of mechanical motion due to slack in the belt
- Gantry racking

Dinh & Gelman, 2015

A 3D printer is composed of a gantry on which the extruder block moves, and a belt that enables motion in X and Y. Examples of gantry types include H and X-Y.

Backlash : Loss of mechanical motion due to slack in the belt. This has a dampening effect on motion control resulting in a lemon-like profile if we were to print a perfect circle.

Gantry racking is the divergence in Y as the toolhead is moved left to right and back in an H gantry. The divergence is due to a net torque about the x-axis center.



Makerbot printers have an H gantry similar to this.

BACKLASH COMPENSATION

- Backlash occurs when there is a change in direction of printing
- Determine the amount of lost motion due to backlash
 - We recommend printing a 20 mm cube and measuring its sides to see if it is < 20 mm
- Add the motion back in gradually after the change in direction

Dinh & Gelman, 2015

As mentioned, backlash is the loss of mechanical motion due to slack in the belt, and has a dampening effect on motion control. The effect is seen when we change direction during extrusion, typically resulting smaller prints. To compensate for backlash, we add the lost motion back in gradually after the change in the direction.

In Makerbot Desktop, users can set the following:

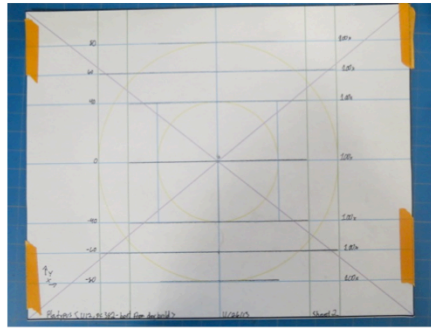
- backlashX" / "backlashY" – mm loss in each direction
- How quickly to compensate for this lost motion

References:

https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/10-MakerBot_Slicer_settings:_Backlash_Compensation

H GANTRY RACKING

Increased divergence in Y farther from the center



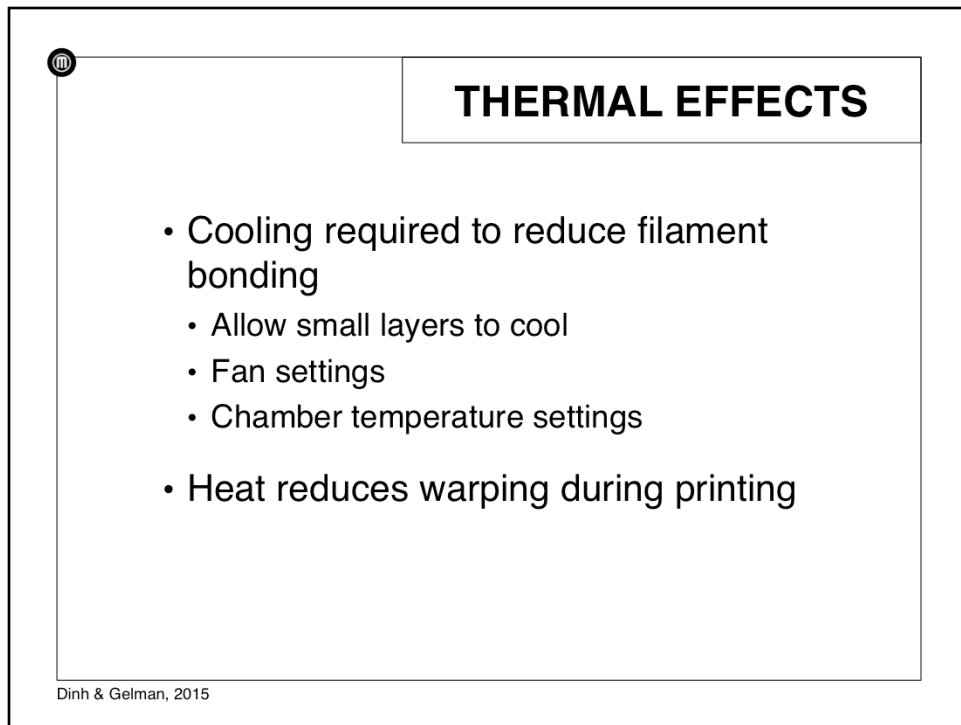
Pen Plot

Dinh & Gelman, 2015

Gantry racking is the divergence in Y as the toolhead is moved left to right and back. The divergence is due to a net torque about the x-axis center. What we get is a compressed figure-8 path where the Y-motion at the ends is on order of mm. This plot is just an example where a pen has been mounted to the toolhead to track the path.

Racking more exaggerated farther from the center. So, printers with smaller build platforms such as the Replicator Mini will experience less racking.

We can't really compensate for racking, but as long as it's small enough, plastic adhesion can overcome adverse affects.



THERMAL EFFECTS

- Cooling required to reduce filament bonding
 - Allow small layers to cool
 - Fan settings
 - Chamber temperature settings
- Heat reduces warping during printing

Dinh & Gelman, 2015

Next are thermal effects.

We enforce cooling in a number of ways.

Tall, thin prints would collapse if we don't allow its small layers to cool sufficiently. So we require each layer to take some minimum amount of time.

Fan usage: the fan is on most of the time, but off during travel moves, and runs at higher speeds during 1st layer.

Heated chamber only on the Z18 and a heated build plate on the Rep2x. Heated chamber is meant to reduce warping during printing. Heating is used sparingly, mostly adjusted settings for better rafts and reduce stringing.

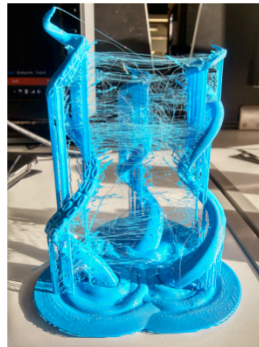
References:

https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/17-MakerBot_Slicer_settings:_Fan_controls



THERMAL EXPANSION

- Heat creeping up nozzle melting plastic
- Visual artifacts
 - Stringing
 - Blobbing



[remix of Octopus Arm Toothbrush Holder
([notcolinforreal](#)) / [CC BY-SA 3.0](#)]

Dinh & Gelman, 2015

Thermal expansion inside nozzle - heat creeps up plastic inside the nozzle. This can cause:

- Causing oozing plastic which can ooze out of the nozzle resulting in stringing
- Blobbiness as the volume of melted plastic is pushed out

So we have these conditions, what can we do to mitigate these problems?


Attribution:

<http://www.thingiverse.com/thing:155396>

<http://www.thingiverse.com/notcolinforreal/about>

Remix by <http://www.thingiverse.com/seanocr/about>

<http://creativecommons.org/licenses/by-sa/3.0/>



THERMAL EXPANSION

- Heat creeping up nozzle melting plastic
- Visual artifacts - Solutions
 - Stringing
 - Blobbing

- **Retraction**
- Chamber temperature
- Turn off fan during non-extrusion moves
- Account for oozing plastic
- Wiping off trailing plastic

Dinh & Gelman, 2015

Stringing can be reduced by tuning temperature for Z18s, for turning off the fan during travel moves, and most importantly **RETRACTION**.

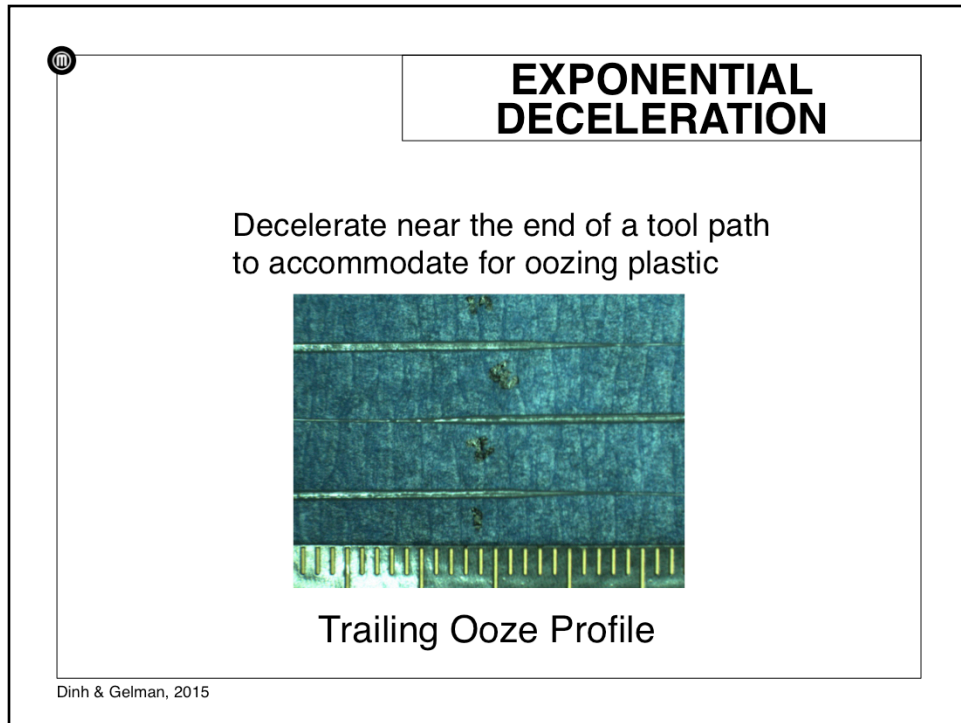
If temperature is too high, will lead to stringing.

Turning the fan off during long travel moves reduces stringing. Fan modulation is a way to avoid turning the fan on and off for EVERY travel move.

Some Slicers reduce stringing and blobs by wiping off excess plastic. One way to do this at the end of a print path is to move the extruder back over a traveled path. Simplify 3D does this. We effectively wipe off excess plastic in dual extrusion by printing purge walls described later.

References:

https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/17-MakerBot_Slicer_settings:_Fan_controls
https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/MakerBot_Slicer_settings:_Exponential_deceleration



Basic idea of exponential deceleration is to shut off the flow early, and decelerate while continuing to ooze plastic at the end of a path, thereby reducing stringing.

From Makerbot Support pages:

“Melted plastic is a liquid, so when your extruder stops extruding, plastic will continue to ooze out of the nozzle for a short time at a decreasing rate. Exponential deceleration allows you to use that oozing plastic. First, it stops extrusion a little bit early, so that it can use the ooze to finish the line of extrusion. Then it slowly decelerates the extruder to correspond with the decreasing flow of plastic, so that extrusion width remains consistent.”

We profiled the trail of oozing plastic to determine the rate of deceleration.

References:

https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/MakerBot_Slicer_settings:_Exponential_deceleration



THERMAL EXPANSION

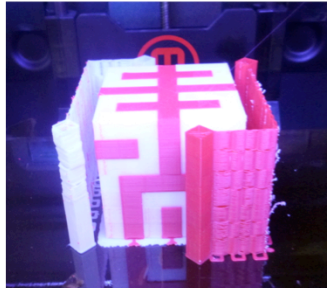
- Heat creeping up nozzle melting plastic
- Visual artifacts - Solutions
 - Stringing
 - Blobbing
 - For dual material, use Extrusion Guards
 - For single, don't stop printing!



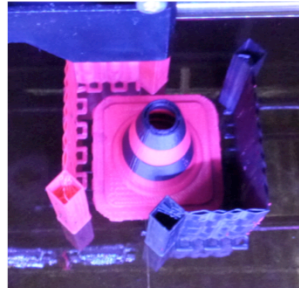
EXTRUSION GUARDS

Problem: idle extruders continue to ooze plastic causing “dirty” prints and clogged nozzles

Solution: print Extrusion Guard wall before printing object with idle extruder



[Hilbert Cube ([tbuser](https://www.thingiverse.com/tbuser)) / [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/)]



[Traffic Cone (Dual Extrusion) ([CocoNut](https://www.thingiverse.com/CocoNut)) / [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/)]

Patent pending

Dinh & Gelman, 2015

A problem with Dual extrusion is that when one of the extruders is idle, it will continue to melt and ooze plastic.

The solution to this is what we call Extrusion Guards. Before an idle nozzle is used to print the object, we use it to print a layer of the purge wall. Doing so expunges any blobs that have formed within the idle extruder.

Extrusion Guards are printed in zigzag pattern to prevent collapse. The walls are located at the bounding box of the meshes, as opposed to following the surface, for faster print times.

Attributions:

<http://www.thingiverse.com/thing:16343>

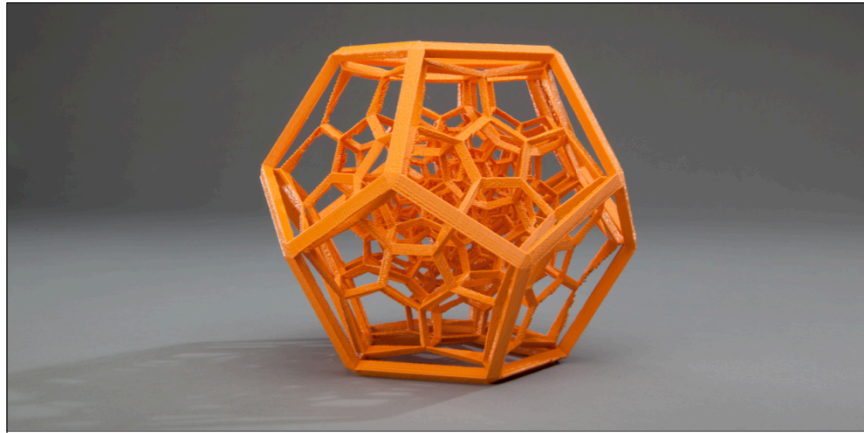
<http://www.thingiverse.com/tbuser/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

<http://www.thingiverse.com/thing:21773>

<http://www.thingiverse.com/CocoNut/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

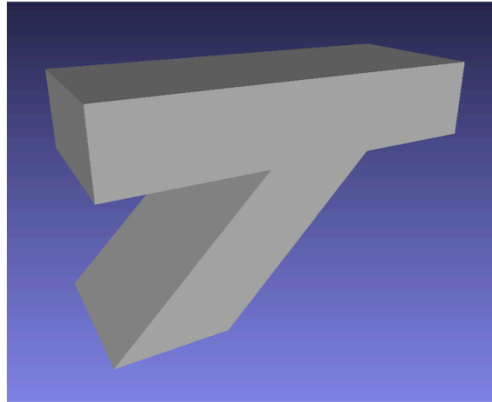


FACTORS AFFECTING PRINT TIME

3D Printing with Plastic Filament



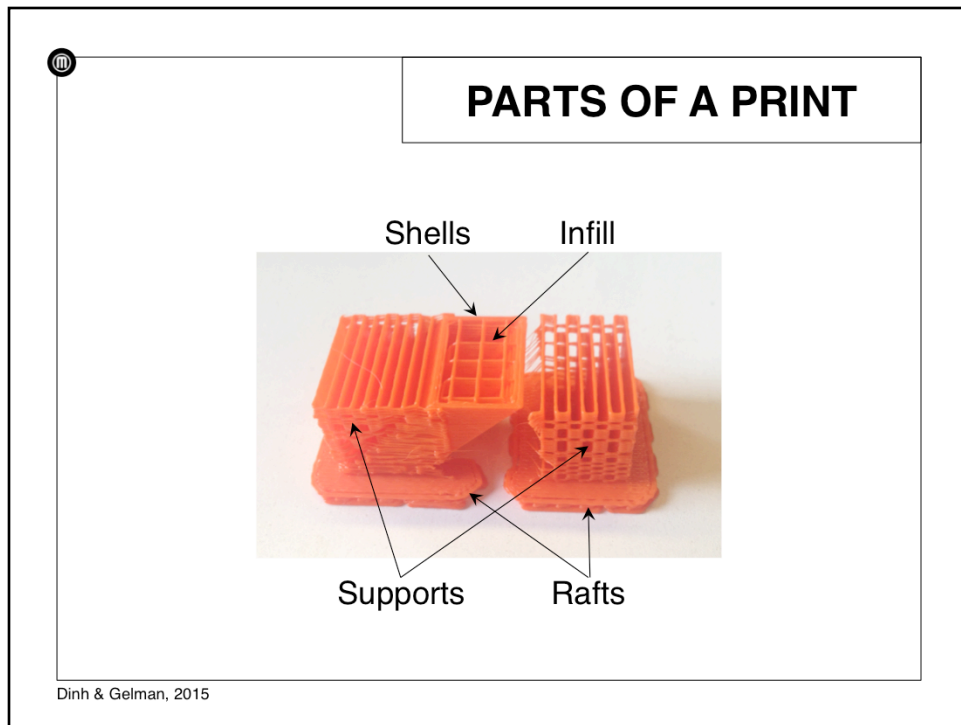
PARTS OF A PRINT



[Support test model, MakerBot]

Dinh & Gelman, 2015

Before describing the steps and factors in toolpath generation, let's cover the different parts of a printed model. Let's take this Support test model as an example.



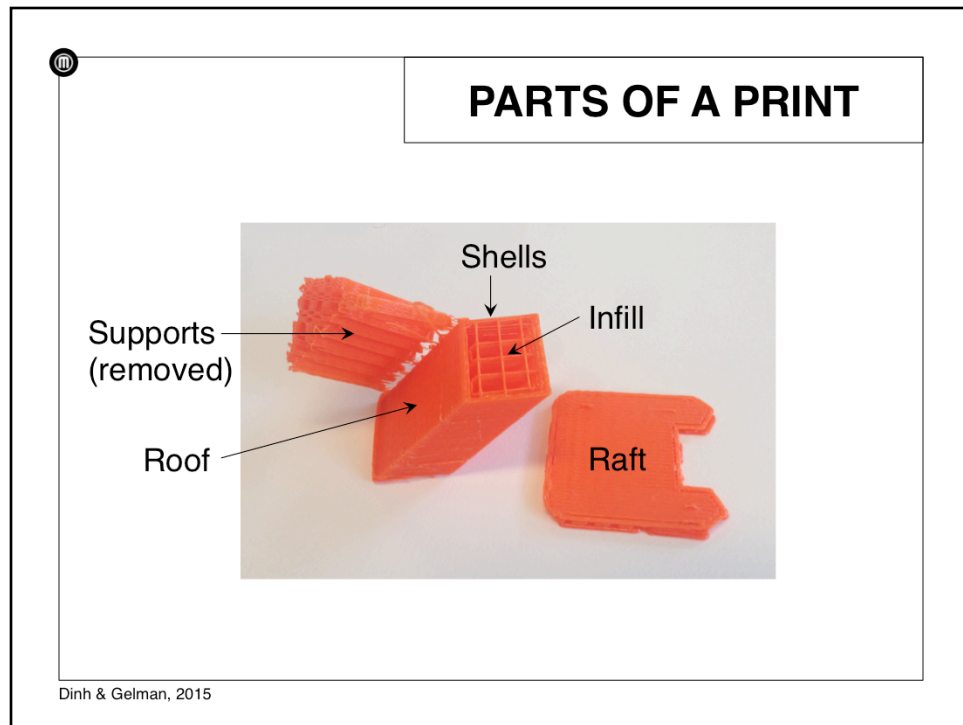
Shells are generated from the model contours.

Infill is the fill inside of shells to give the printed object strength and to hold up the tops (a.k.a. roofs) of the printed object.

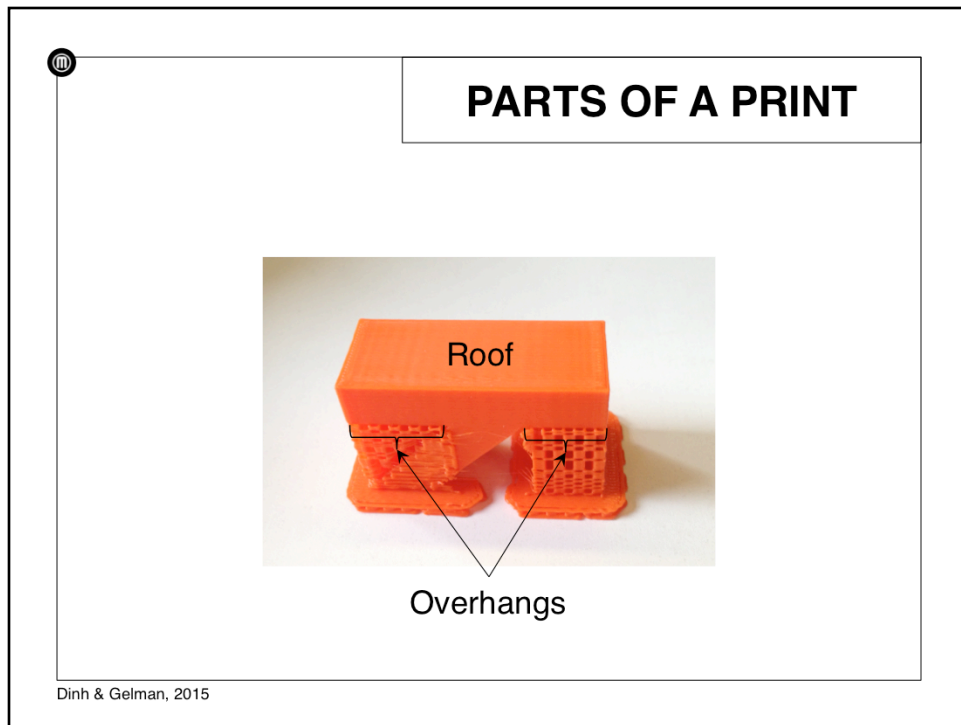
Supports hold up overhanging regions of the object. Overhangs are parts of the model where the layer above extends well beyond the layers below.

Rafts are not part of the object, but are printed to make it easy to remove the object from the build plate and reduce warping in the first few layers of the object. Rafts will be covered in more detail in a later session.

Regions to be filled in are often called "islands".

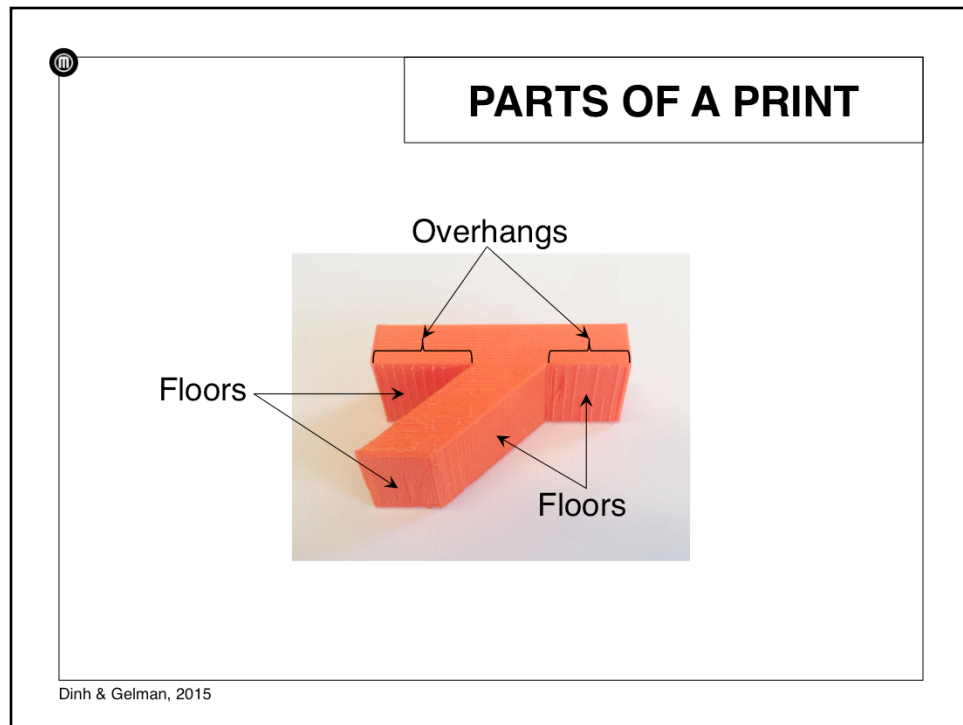


Roofs are the tops of the object. Supports may be printed on top of roofs as seen here to support parts of the object that overhang farther up in Z-height.



Typical roof that form the top of the object.

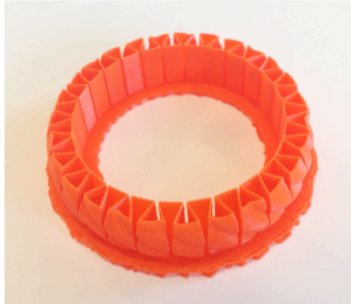
Overhangs are parts of the model where the layer above extends well beyond the layers below, and gravity will cause the printed plastic to droop down, like printing over air if they are not supported. Supports will be covered in more detail in a later session.



Floors are the bottom surface of an object. Floors may be supported (some shown here) if they are overhangs (printed supports removed).

SPECIAL FEATURES

- Single-filament walls
- Bridges



[Stretchy Bracelet ([emmett](#)) / [CC BY-SA 3.0](#)]

Dinh & Gelman, 2015

Single-filament objects may exhibit “elastic” behavior.

Attribution:

<http://www.thingiverse.com/thing:13505>

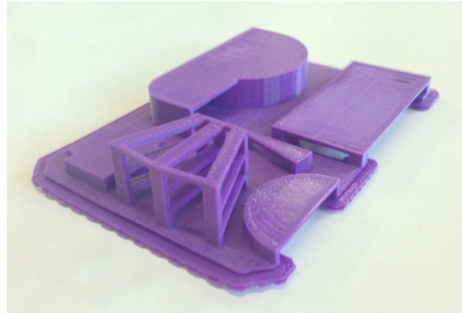
<http://www.thingiverse.com/emmett/about>

<http://creativecommons.org/licenses/by-sa/3.0/>



SPECIAL FEATURES

- Single-filament walls
- Bridges



[Bridges test model, MakerBot]

Dinh & Gelman, 2015

Bridges will be covered in more detail in a later session.



PRINT SPEED HEURISTICS

- Shells, floors, and roofs are printed slowly for high quality
- Infill and supports can be printed at higher speeds
- **Travel** moves are typically at faster speeds than extrusion moves
- **Leaky** moves avoid retraction

Dinh & Gelman, 2015

Travel moves are non-extrusion moves that include a retraction. Retractions require a full stop.

Leaky moves are travel moves w/o retraction. The extruder will continue to ooze plastic, causing stringiness during the travel, but avoids a retraction which is time consuming due to the required full-stop and the time for retraction.


Reference:

https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/03-MakerBot_Slicer_settings:_Travel_Movement
https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/MakerBot_Slicer_settings:_Leaky_connections

PRINT TIMES	
(Estimated hrs:mins:secs)	
20mm Cube	00:13:29
Full Replicator Mini Build Volume	21:18:22
Full Replicator Build Volume	42:17:59
Full Z18 Build Volume	199:19:12

Dinh & Gelman, 2015

Estimated print times with default settings.



REDUCING PRINT TIME

- Extrude at high speed for as much of the print as possible
- Tool path optimization
 - Start a layer near the last
 - Reduce retractions – allow leaky moves
 - Speed up travel moves
- Avoid sharp turns (e.g., in printing fill)
 - “Optimization of tool-path generation for material extrusion-based additive manufacturing technology”, Jin et al., Additive Manufacturing, 2014
- Thick infill and supports

Dinh & Gelman, 2015


Toolpath optimization – e.g., by starting a layer near where the last layer finished, and avoid retractions (which also require a full-stop).

We can reduce retractions by, for example, not retracting when traveling over infill since we don't care that we have extra plastic in the infill.

Avoiding sharp turns in printing reduces print time because the gantry and extruder does not have to slow down to make the turn. The sharper the turn, the more deceleration is required – e.g., 360 degree turn is the worst. Because we don't really want to alter the model contours, the only place we can avoid sharp turns is in the fill pattern. So, for example, linear infill is much faster than hexagonal infill. In Jin et al.'s paper, they optimize the orientation of the fill pattern to reduce sharp turns.

Reference:

“Optimization of tool-path generation for material extrusion-based additive manufacturing technology”, Jin et al., Additive Manufacturing 1-4 (2014), pp32-47



100 MICRON PRINTS

- Slow – dense tool path
- Poor quality
 - 100μ solid fill is brittle
 - 100μ supports fall down
 - 100μ bridges fail
- Without rafts, build plate needs to be leveled within 100μ
- **Solution:** Thick infill and supports

Dinh & Gelman, 2015

So with the advent of 100 micron layers, came a series of problems related to both print speed and quality.


Solid fill at 100 microns is brittle, and takes many layers to seal over.

Support at 100 microns doesn't stand up.

100 micron bridges don't work well.

W/o rafts, 100 micron prints require build plate to be leveled to within 100 microns.

Our slicer architecture enabled us to break the uniform layer constraint. Uniform layer constraint is where all layers must be the same height, and in particular, all elements within a layer must be the same height. By breaking the uniform layer constraint, we can more quickly print infill and supports by making them thicker than the high resolution shells. Simplify3D also does this.



THICK FILL

- Print infill, supports, and rafts at standard layer height (reduces print time)
- More complex tool path optimization
- Path dependencies:
 - Z height
 - Move platform down, not up
 - Extrusion guards
 - Minimum cooling time required between layers

Dinh & Gelman, 2015

Infill, supports, and rafts do not need to be high resolution, so print them at standard layer height. But this leads to a more complex toolpath because now within a layer, you may have components that are off different heights.

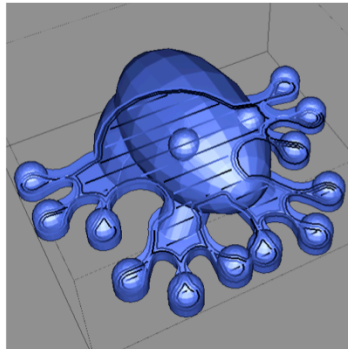
Z height dependency: obviously, we want to print layers below before printing those that sit on top of them.

But more specifically, we want to always move the nozzle up and not down – we don't want nozzle to hit any printed material

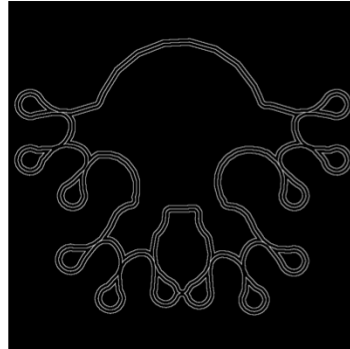
For dual material, need to account for extrusion guards

Minimum cooling time – minimum time spent on printing a layer. W/o this, very narrow tall things will not have time to cool between layers and they become unstable and collapse

These constraints become more complicated to implement when we have regions of variable layer height.



[Frog ([owenscenic](#)) / [CC BY-NC-SA 3.0](#)]



SLICES

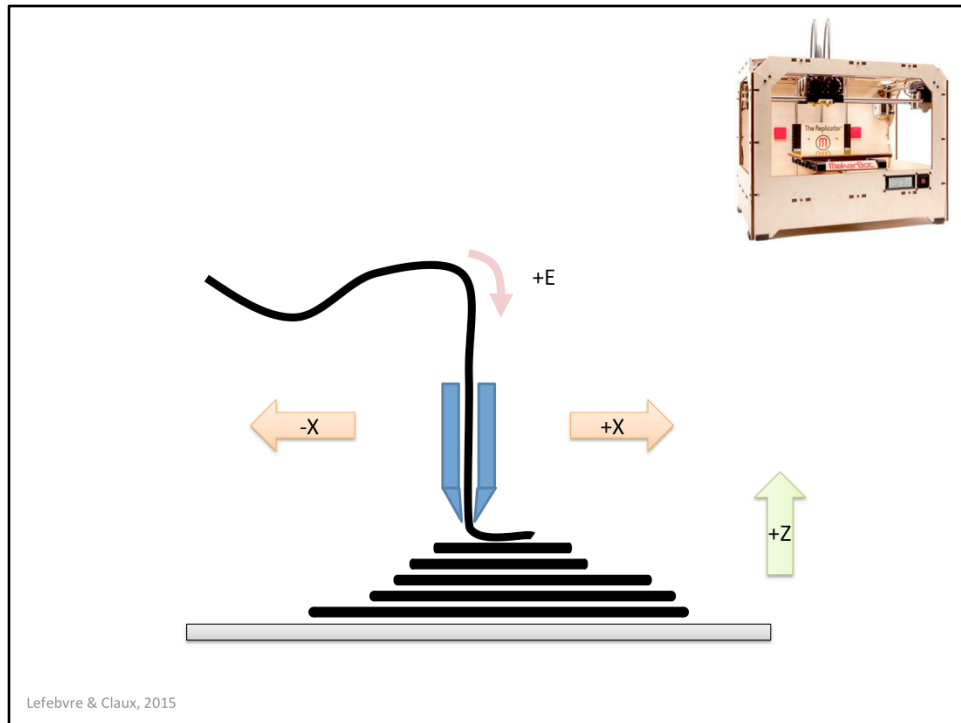
3D Printing with Plastic Filament

Attribution:

<http://www.Thingiverse.com/thing:3284>

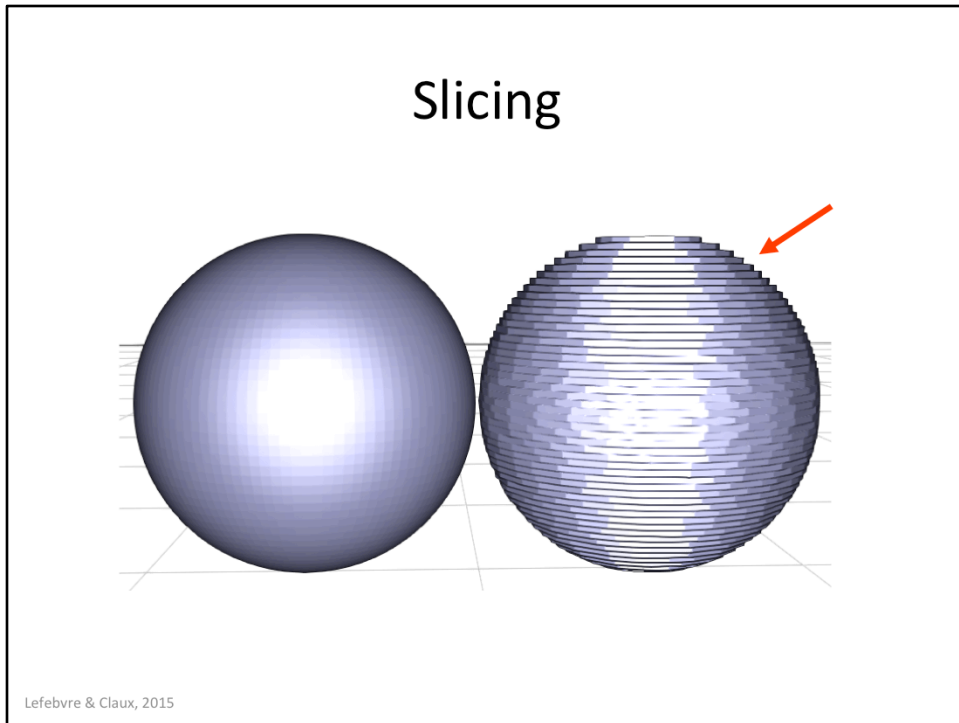
<http://www.thingiverse.com/owenscenic/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>



Filament printers, like most additive manufacturing processes, create the object layer after layer.
Therefore, a crucial operation is how to split, or *slice*, the object into a set of layers.

Slicing

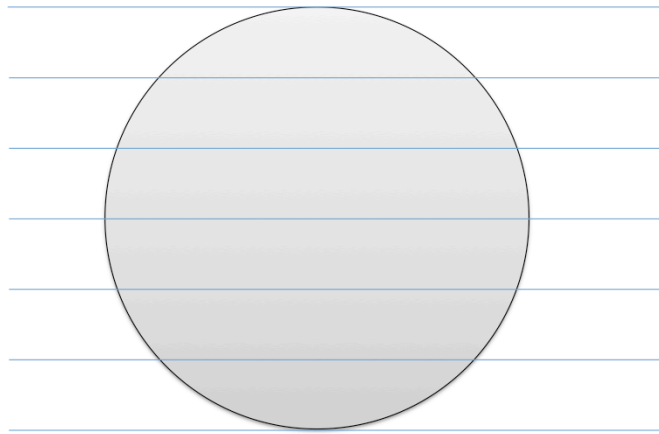


Here you can see on the left the real object (a finely tessellated sphere) and on the right what will actually get printed. It is obvious that the object we print is different from the initial one. In particular note the stair-stepping effect along the silhouette.

Using thinner slices will reduce this effect, but will require significantly more time. In fact, for the same object you can expect 0,1mm slices to take three times the time it takes to print with 0,3mm slices.

Uniform slicing

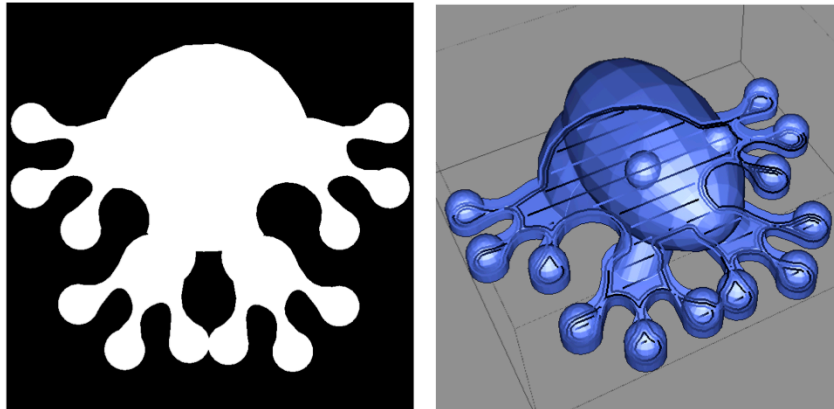
- Fixed thickness (typically from 0,1 to 0,3 mm)



Lefebvre & Claux, 2015

The simplest way to divide a 3D model into slices is to do a uniform slicing, that is all slices have the same thickness and the object is regularly subdivided.

Slice plane



[Frog ([owenscenic](https://www.thingiverse.com/thing/3284)) / [CC BY-NC-SA 3.0](https://creativecommons.org/licenses/by-nc-sa/3.0/)]

Lefebvre & Claux, 2015

The left image shows what a single slice looks like. It corresponds to the intersection between the object and a plane. Here we show in white the inside and in black the outside. Most slicers will represent this as a polygon, and others will represent the slice as an image.

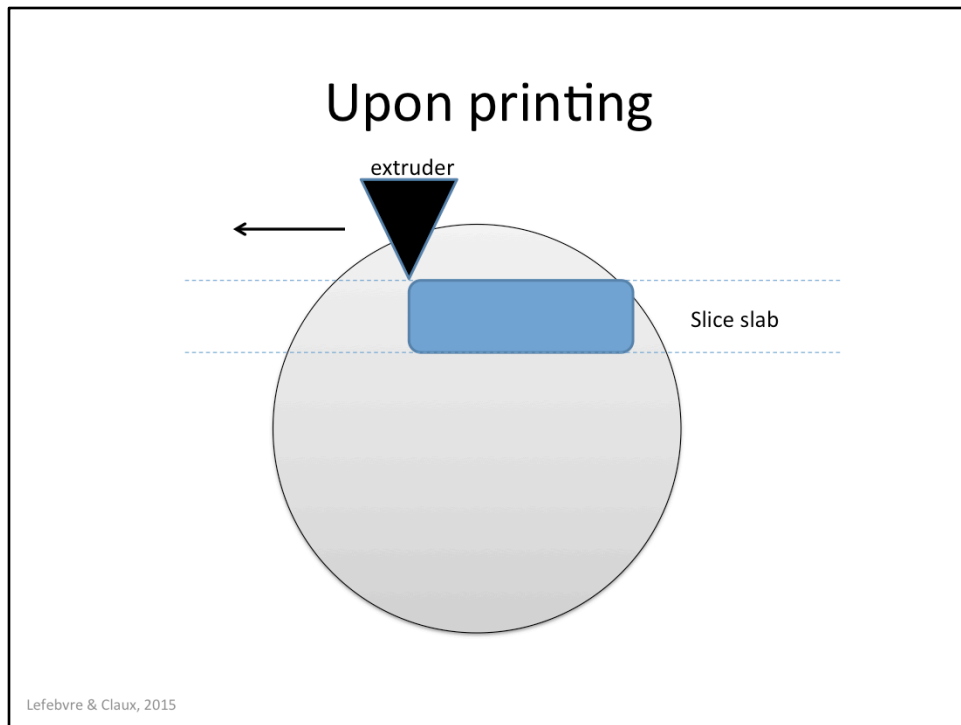
On the right side you can see the object and where the slice is located. You can also see the toolpaths, that is the paths along which plastic is deposited. We will detail toolpaths in the next part of the course.

Attribution:

<http://www.thingiverse.com/thing:3284>

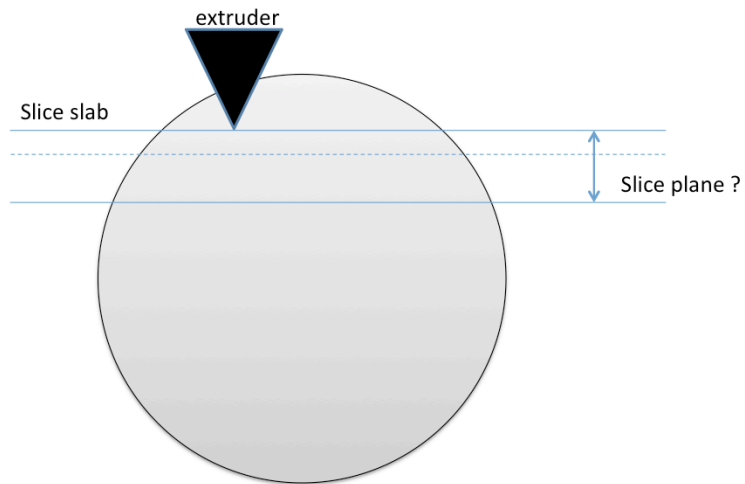
<http://www.thingiverse.com/owenscenic/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>



We have just seen that a slice is the intersection of a plane and the object. When printing, this plane will turn into a slab of plastic.

Where to sample the slice plane?

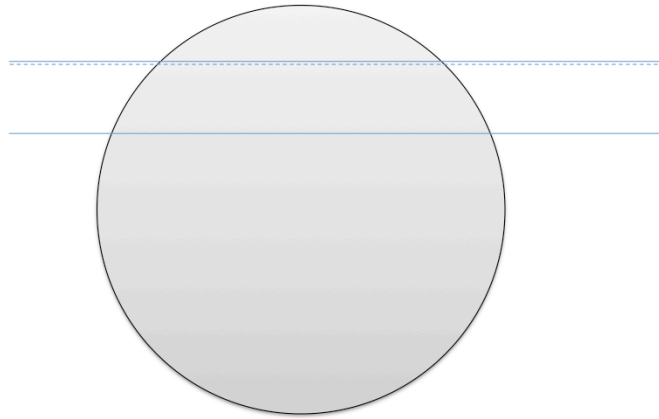


Lefebvre & Claus, 2015

Let us consider the slab of a slice, that is the interval in Z in between which plastic will get deposited.

An intriguing question is where should we sample the slice plane within the slab?

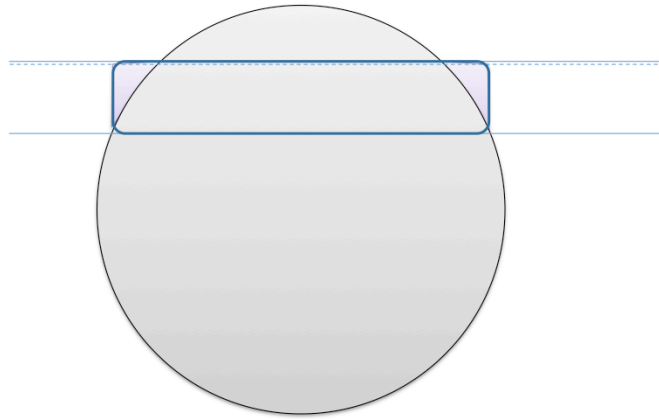
Where to sample the slice plane?



Lefebvre & Claux, 2015

For instance, if we sample at the top ...

Where to sample the slice plane?

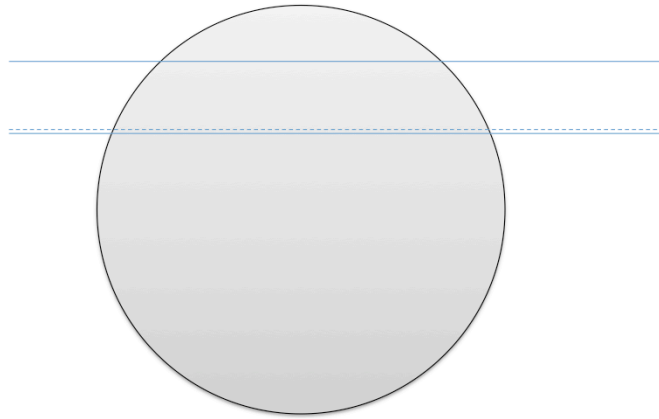


Lefebvre & Claus, 2015

... here is what will be printed.

You can see in purple the plastic that is in the slab but not in the object. In this case the error is quite large: we print an object bigger than the original.

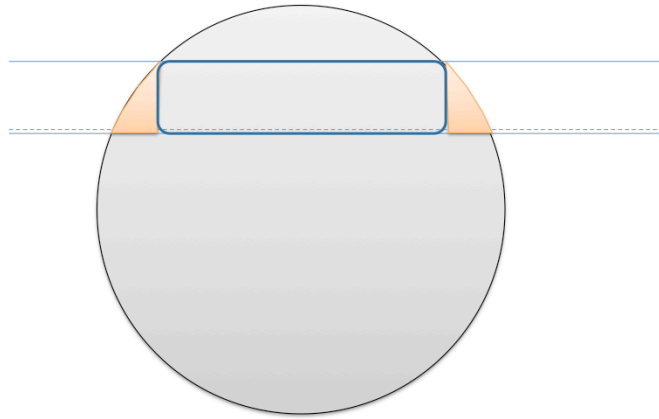
Where to sample the slice plane?



Lefebvre & Claux, 2015

Now let us sample at the bottom.

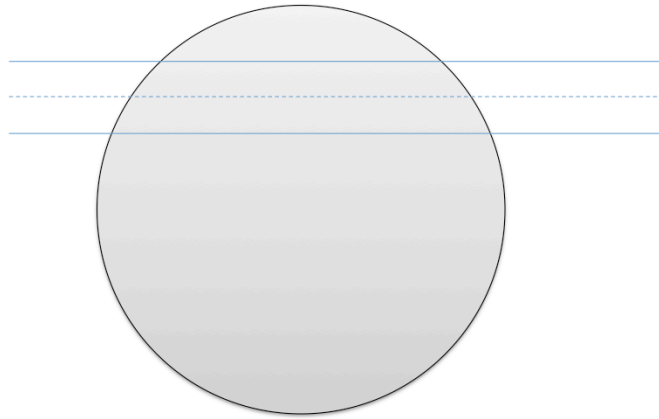
Where to sample the slice plane?



Lefebvre & Claux, 2015

Here is what gets printed. This time we have missing plastic: we print an object smaller than the original,
Again, this is a significant error.

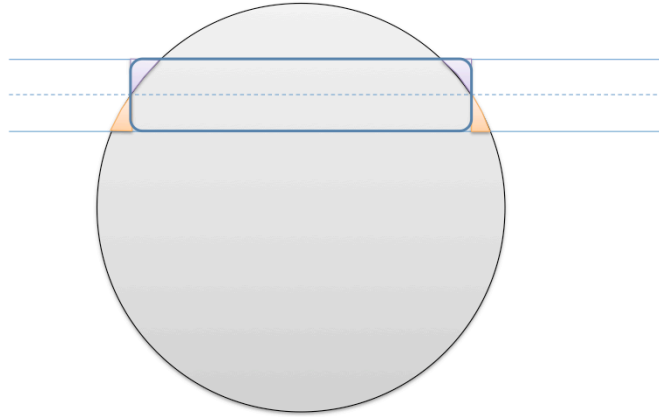
Where to sample the slice plane?



Lefebvre & Claux, 2015

Let's try the middle.

Where to sample the slice plane?



Lefebvre & Claux, 2015

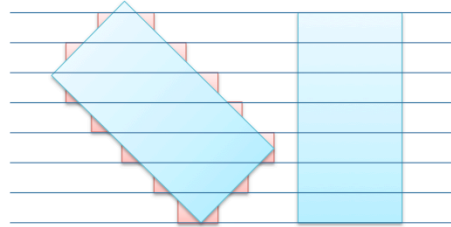
Here is what is printed. This seems a better compromise between both.

However, the 'best' sampling position really depends on the geometry of the object and the type of error the user is ready to have.

Many slicers simply sample at the middle of the slice though, which avoids complications and produces reasonable results.

Minimizing volume error

- Optimization
 - Minimize volume error?
- Variables:
 - Object orientation!
 - Slice thicknesses
 - Slice plane locations



Lefebvre & Claus, 2015

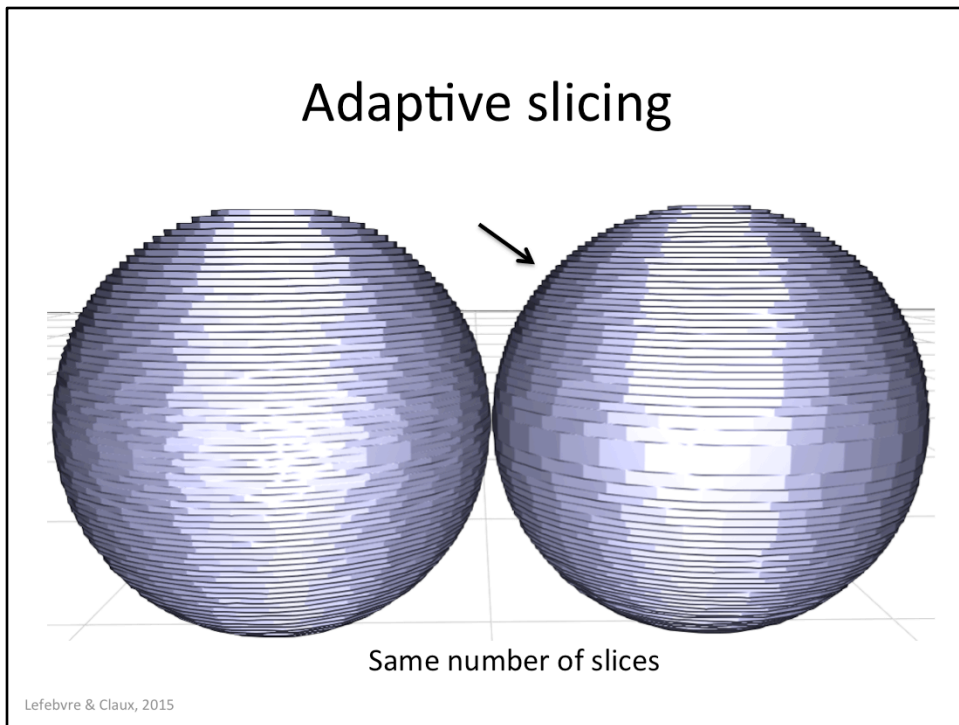
Of course this entire process can be seen as an optimization problem.

There are several variables. An important one is the orientation of the object! For a simple rectangle, the vertical orientation of course gives a perfect results, while any other will result in errors. Of course, on most complex objects this will be a tradeoff.

Other variables are slice thicknesses, and the location where the slice planes are sampled.

As we have already seen, making slices thinner improves the result – it reduces the error – however this requires significantly more print time.

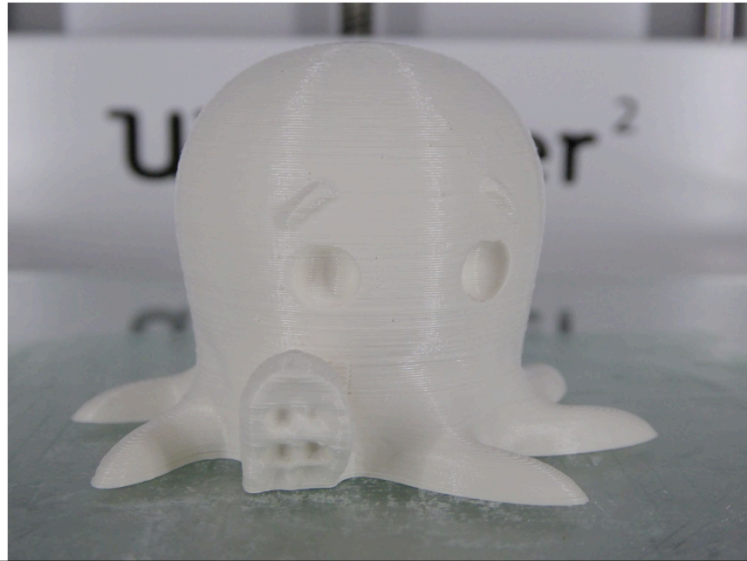
Adaptive slicing



Therefore a different approach has been investigated. The idea is that some parts of the object might produce less errors than others. In 'easy' regions thick slices can be used, whereas smaller slices will be used in more difficult regions.

Adaptive slicing

[Cute Octopus Says Hello ([MakerBot](#)) / [CC BY 3.0](#)]



Lefebvre & Claux, 2015

Here is a printer example using adaptive slices.

Attribution:

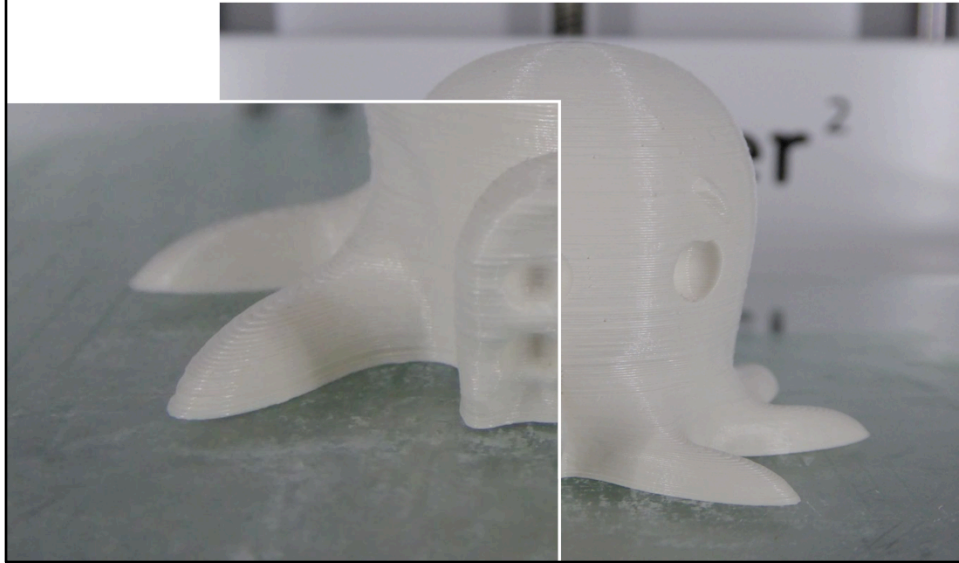
<http://www.thingiverse.com/thing:27053>

<http://www.thingiverse.com/MakerBot/about>

<http://creativecommons.org/licenses/by/3.0/>

Adaptive slicing

[Cute Octopus Says Hello ([MakerBot](#)) / [CC BY 3.0](#)]



A close up reveals that thinner slices are used on the tentacles, where the surface becomes almost flat.

Attribution:

<http://www.thingiverse.com/thing:27053>

<http://www.thingiverse.com/MakerBot/about>

<http://creativecommons.org/licenses/by/3.0/>

Adaptive slicing

- Faster for same precision
- Do not waste time in 'simple' regions
- Not so easy to determine best strategy
 - See survey by [\[P.M. Pandey et. al. 2003\]](#)
 - Recent work: [\[Wang et. al. 2015\]](#)

Lefebvre & Claux, 2015

Adaptive slicing has clear advantages. However, it is not easy to determine the best strategy and to optimize for slice thicknesses. There are several considerations such as the precision, the volume error, but also aesthetics of the final part.

There is a lot of work in this area, but I wanted to point this survey by Pandey and colleagues, as well as recent work by Wang and colleagues regarding adaptive slicing. These are good entry points into the domain if you are interested.

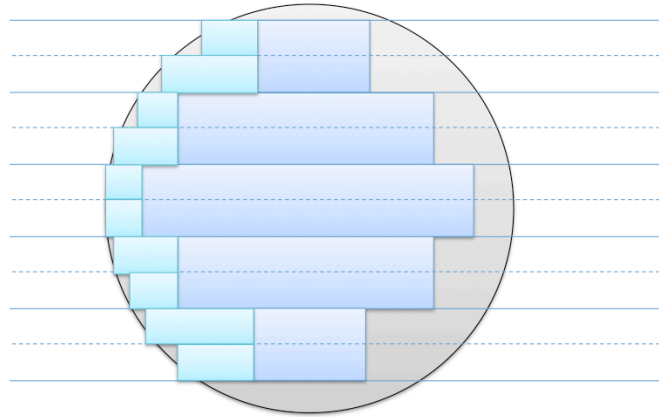
[P.M. Pandey et. al. 2003]

http://web.iitd.ac.in/~pmpandey/RP_html_pdf/slice_review.pdf

[\[Wang et. al. 2015\]](#)

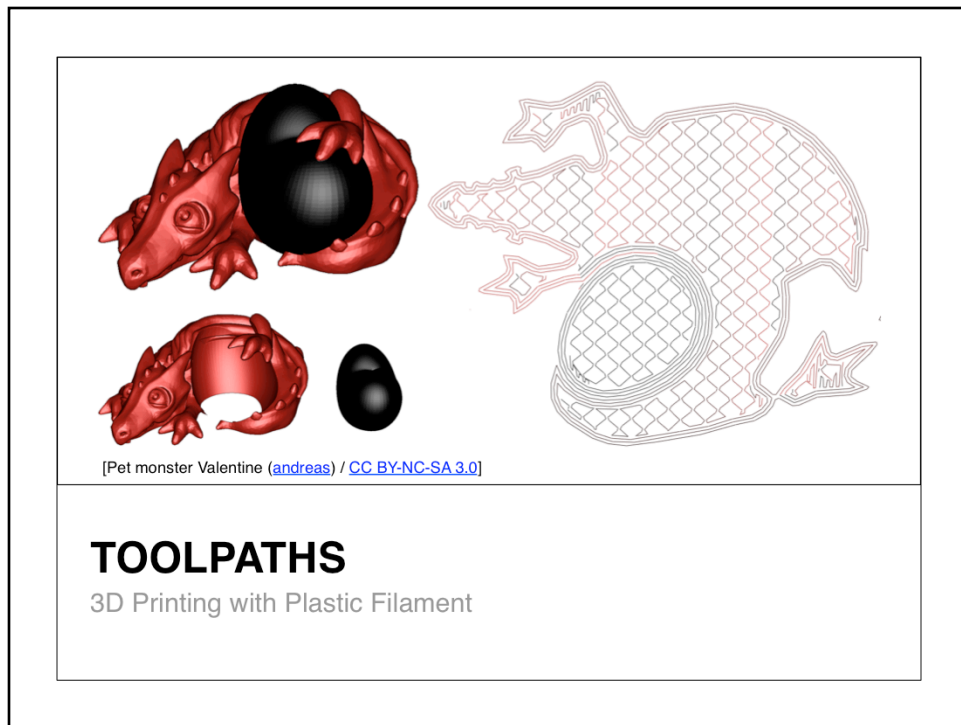
http://staff.ustc.edu.cn/~lgliu/Projects/2015_AdaptiveSlicing_3DPrinting/default.htm

Nested slices



Lefebvre & Claux, 2015

In IceSL we also looked into a different type of adaptive slicing. The idea is to continue using very thin slices on the outside, but use thicker slices inside. Indeed, most of the time is spent infilling the object!

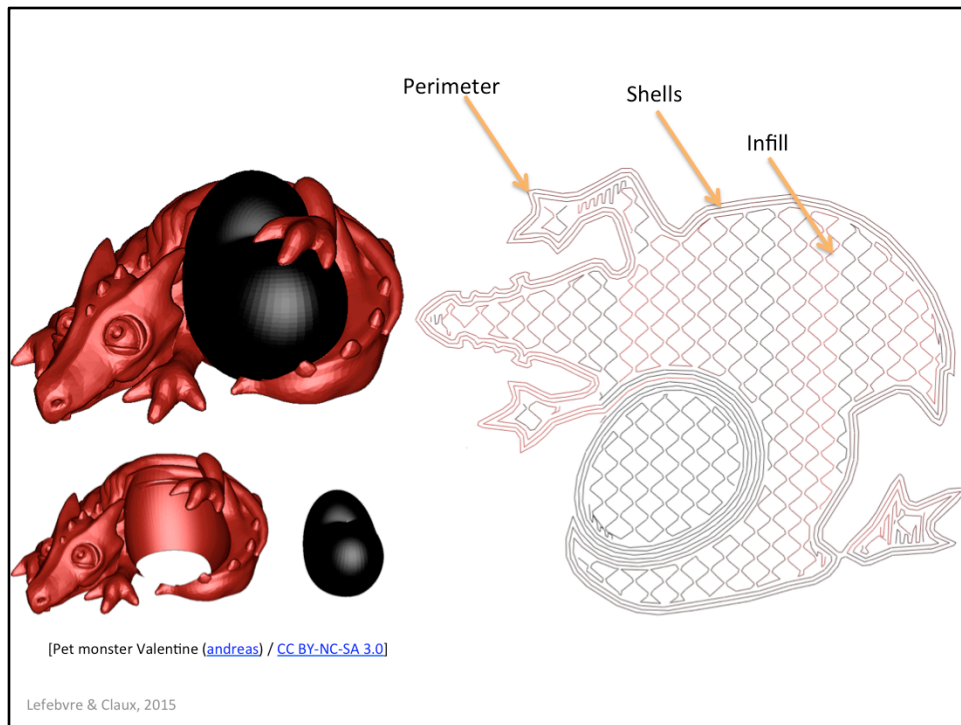


Attribution:

<http://www.thingiverse.com/thing:17204>

<http://www.thingiverse.com/andreas/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>



First, let's recall the naming conventions for the different paths. We have perimeters which are the visible part of the object, then shells which are contouring the object inside, and finally the infill paths which are filling the inside.

Attribution:

<http://www.thingiverse.com/thing:17204>

<http://www.thingiverse.com/andreas/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Erosion (morphological)



Structuring element
(nozzle exit hole)



Slice

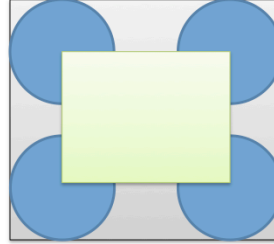
Lefebvre & Claux, 2015

The mathematical tool we will use the most for producing toolpaths is erosion. An erosion is done with respect to a structuring element. In our case this is a disk having the diameter of the exit hole of the nozzle.

Erosion (morphological)



Structuring element
(nozzle exit hole)



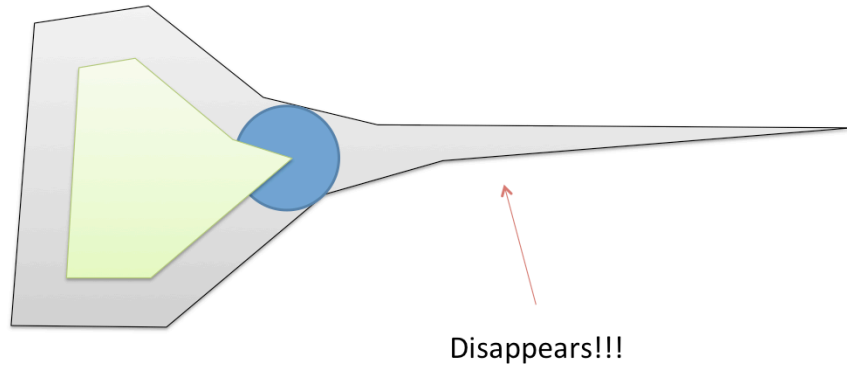
Erosion: All points where
structuring element is
entirely included

Lefebvre & Claux, 2015

Given a slice, shown on the right, the erosion of the slice by the element is defined as the points where the structuring element is fully included within the slice. Here, this is the green polygon.

There are good libraries to compute such operations on polygons, such as CGAL or Clipper (<http://www.angusj.com/delphi/clipper.php>).

'Thin features'

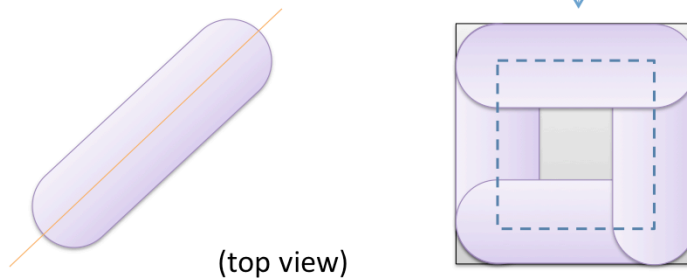


Lefebvre & Claux, 2015

Note that following this definition, thin features – part thinner than the nozzle – will disappear through erosion. We will come back to this later.

Perimeter

- Visible part of the filament
- Object contouring



Lefebvre & Claux, 2015

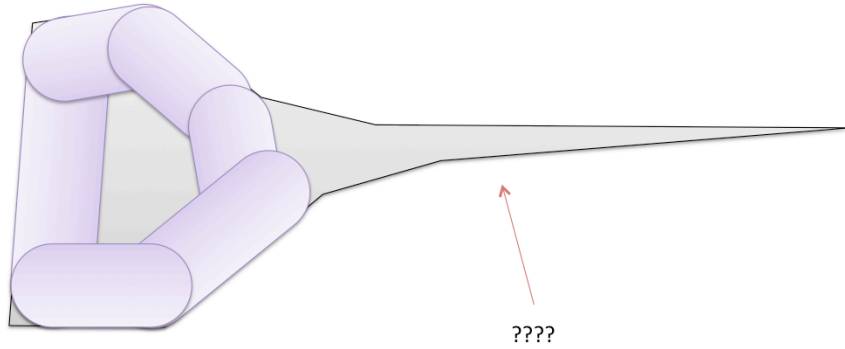
So, given a slice and a nozzle diameter, how do we extract the perimeter toolpath?

First, it is important to realize that when depositing plastic along a path, the track of plastic goes halfway on either sides of the path followed by the extruder, as shown left.

Therefore, we should not deposit plastic on the exact contour of the slice, otherwise the object will get too big by half the nozzle width!

Instead, we erode the slice by a disk having for diameter the nozzle width. The contour of the resulting polygon is the perimeter path.

'Thin features'

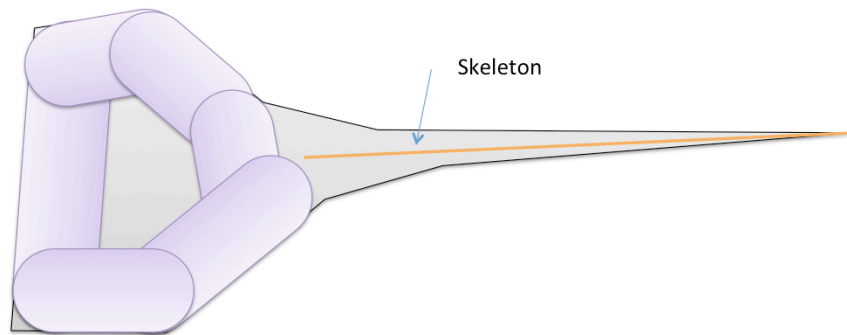


Lefebvre & Claux, 2015

But now, let's come back to thin features. When we do the erosion, features below the nozzle width simply disappear.

Sometimes this is what you want, because these features are arguably below the minimum thickness and can be removed.

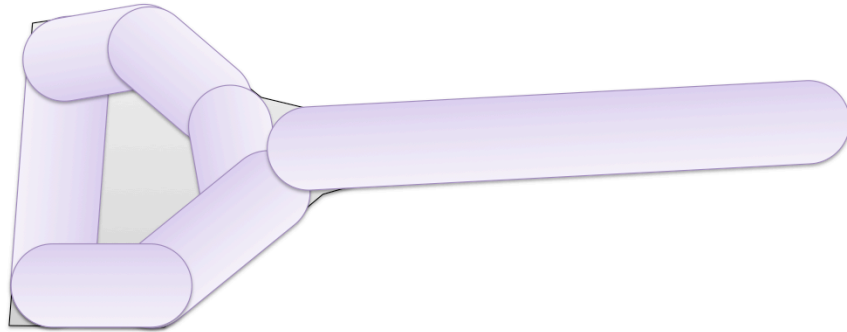
A possible approach



Lefebvre & Claux, 2015

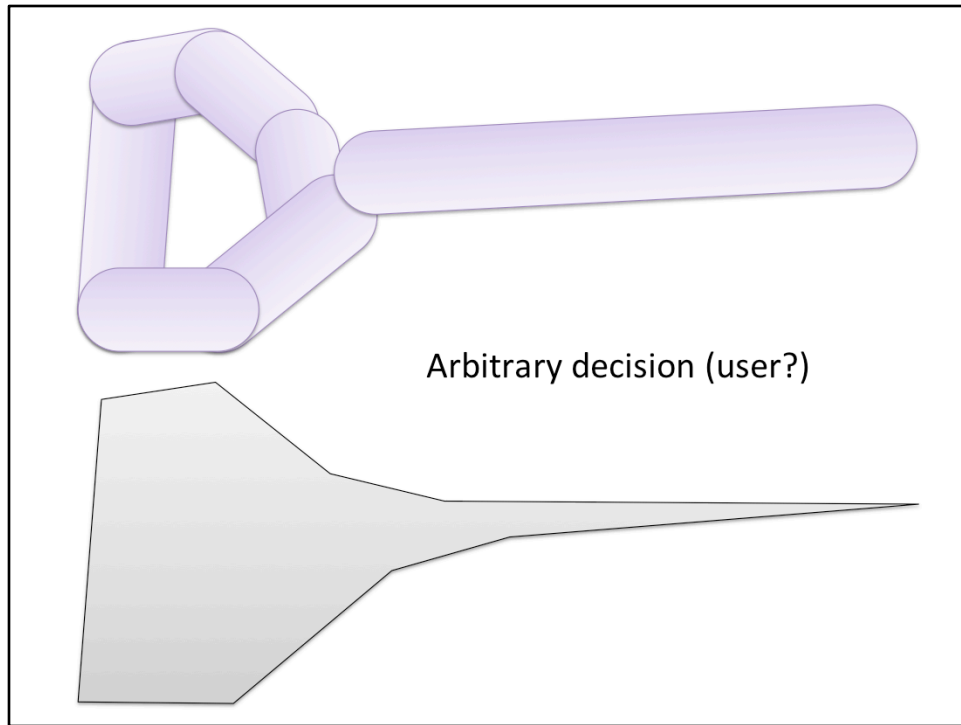
But what if you still want these features? Then a possible approach is to extract the skeleton of the thin parts. This sounds easy, but it is not as skeleton can be difficult to compute.

A possible approach



Lefebvre & Claux, 2015

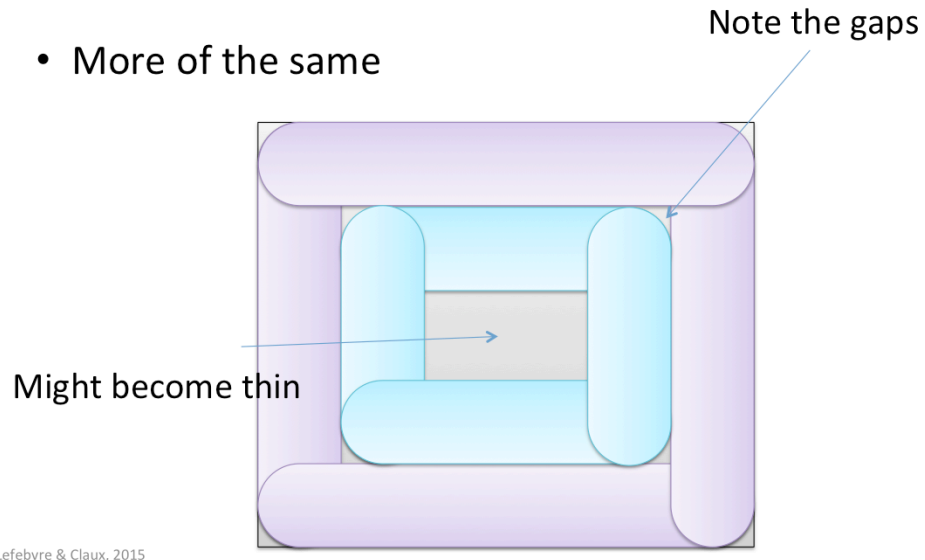
However, once you get the skeleton you can use it as an additional toolpath.



Whether this is a good idea or not is probably up to the user to decide!

Shells

- More of the same



Once we have the perimeter, we are ready to extract a number of shells. We use erosion again.

Note how the inner part might become thin. The thin feature problem will unfortunately appear very often when trying to fill the inside.

Also note that we are leaving small gaps in sharp corners. One possible way to fill these is to push slightly more plastic than exactly required.

Infills

- Top / bottom covers
- Inside
 - Full infill (very robust, slow, lots of plastic)
 - Sparse infill (save plastic and time, less strong)

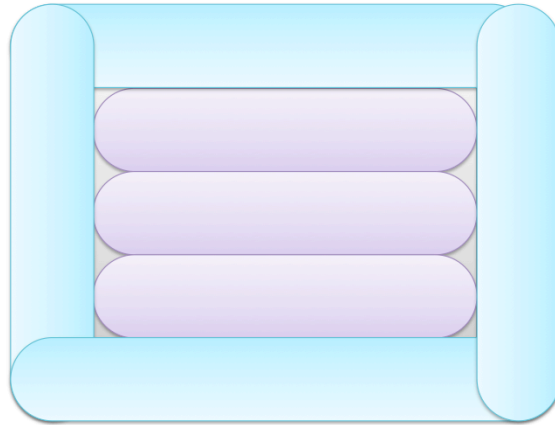
Lefebvre & Claux, 2015

Once we have perimeters and shells, we are ready for infilling.

There are two types of infill. The first is used on tops and bottoms. This infill produces covers such that the inside of the object is not visible through the 'top' of the slices.

The second type of infill is used inside the object. With opaque plastic this is completely hidden from view.

100% Infill

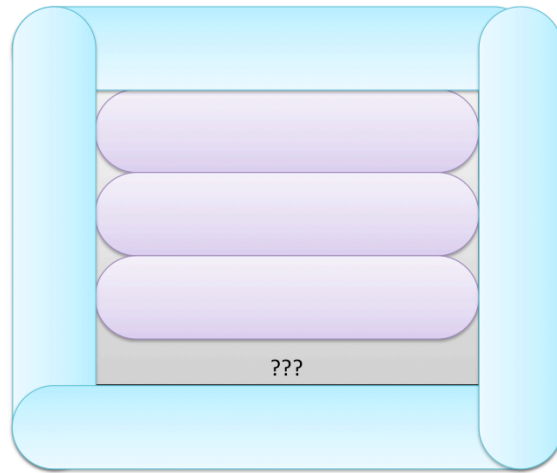


Lefebvre & Claus, 2015

In purple you can see a typical full infill. The idea is to produce a raster pattern covering the inside.

This seems easy enough, however ...

100% Infill

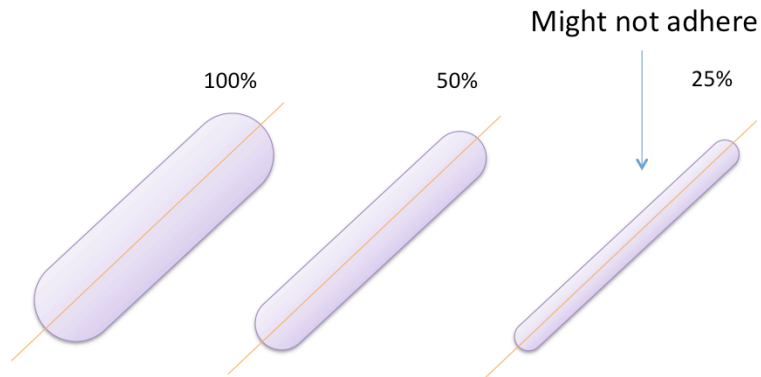


Lefebvre & Claus, 2015

... some difficulties appear again due to thin gaps.

Flow control

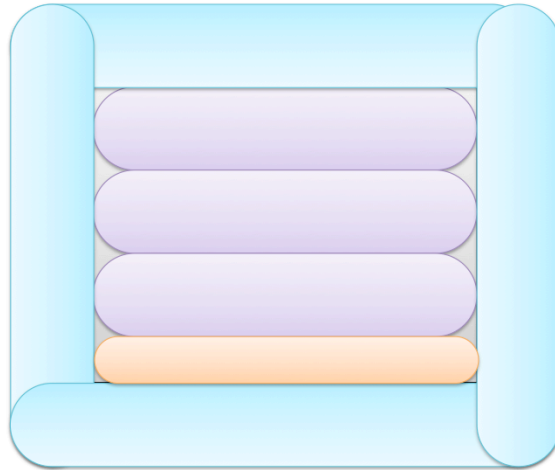
- Limited change of thickness



Lefebvre & Claus, 2015

One idea to fill-in these gaps is to play with plastic flow. By changing the flow the 'thickness' of the plastic track tends to become smaller. However, under some threshold plastic will stop exiting the nozzle continuously, so this idea should not be pushed too far.

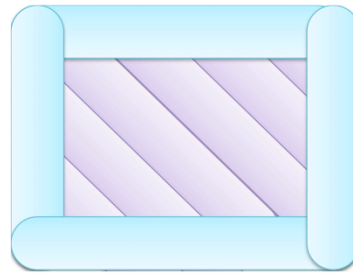
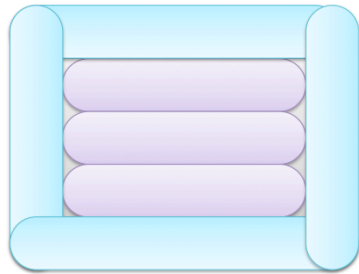
100% Infill



Lefebvre & Claus, 2015

Here is how a smaller plastic track can be used to fill-in the gap. Again this is just a straight segment, pushing less plastic than usual.

Speed



Faster!

Both axis work together: better acceleration

Lefebvre & Claux, 2015

As a side note, keep in mind that using angled segments can actually print faster since the acceleration of both axes is combined.

Sparse Infill

- Simple approach

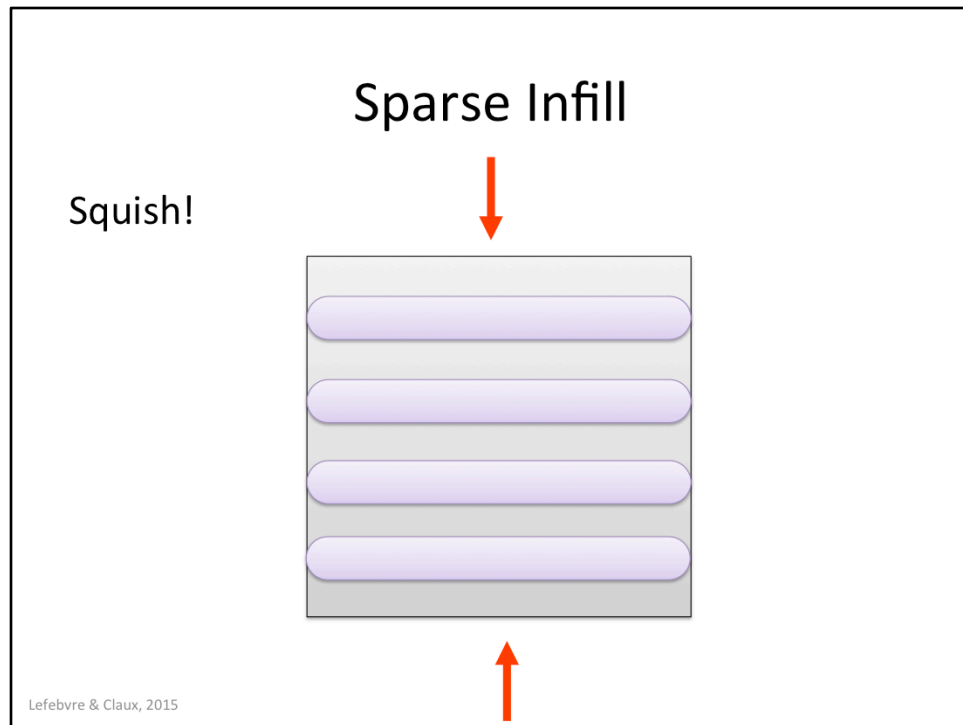


Lefebvre & Claus, 2015

Now let us discuss sparse infillings.

A simple idea is to skip one segment every N . Here, a 50% infill, keeping only half the infill segments.

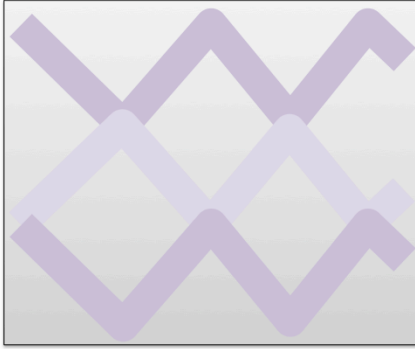
The direction is typically alternated between X/Y every other layer to make the part slightly stronger.



Still, this type of infill will not produce strong prints, and the prints might be easily squished.

Sparse Infill

Better:



Lefebvre & Claux, 2015

Here is a much better infill pattern.

There is a much larger variety of possible infill patterns, and a lot of slicer include exotic infillings.

Best sparse infill ?

Tradeoff between:

- Plastic
- Time
- Strength

Recent work:

- [Lu et. al. 2014] “build to last”

Lefebvre & Claus, 2015

Of course a key question is what is the best infill? This is a tradeoff between print time, plastic use and part strength.

For a recent work on the topic please refer to this paper:

[Lu et. al. 2014]

<http://vr.sdu.edu.cn/~lulin/3DP/build-to-last.html>

In this work the infill pattern is optimized to make the final print stronger.



PATH ORDERING

After generating toolpaths for the different parts of a print (shells, infill, supports, etc), a single, fully ordered toolpath must be generated

Path ordering heuristics are used that promote structural integrity and print quality, and reduce print time

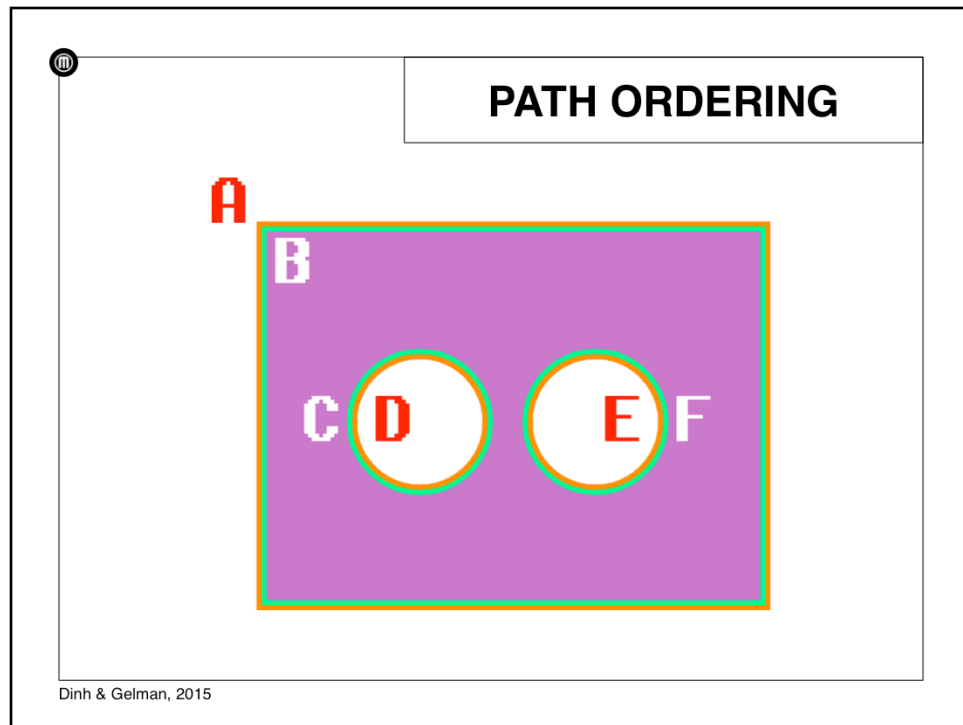
Dinh & Gelman, 2015



PATH ORDERING HEURISTICS

- Reducing print time
 - Start at point closest to the last
 - Contiguous regions of a given type will be printed completely before moving on to other regions
- Print quality
 - Avoid crossing over exterior parts (roofs)
- Structural integrity
 - Shells are printed first before infill
 - Adjacent shells are printed innermost to outermost
 - Holes are printed before outlines

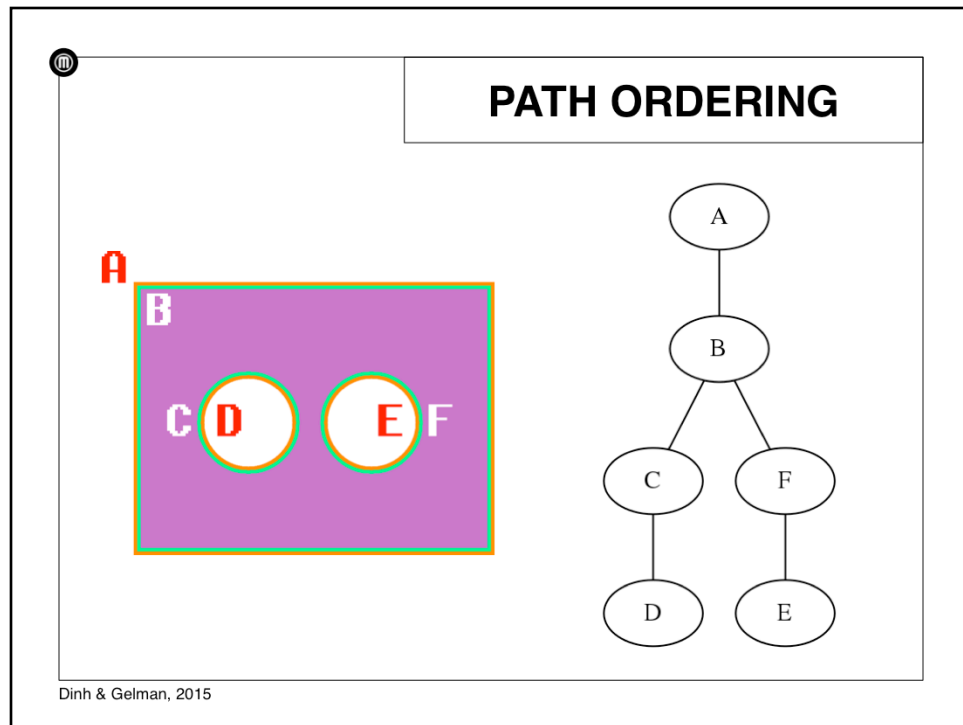
Dinh & Gelman, 2015



We print each island completely before moving onto the next. Within each island, we print all of the shells inner-most first, then all of the infill.

The shells are labeled A to F. Notice how A, D, and E are outer shells and B, C, F are inner shells. Shells are done inner to outer so that out-ward shells can attach the inward shells for more stability. This relative order matters only for shells that are in contact.

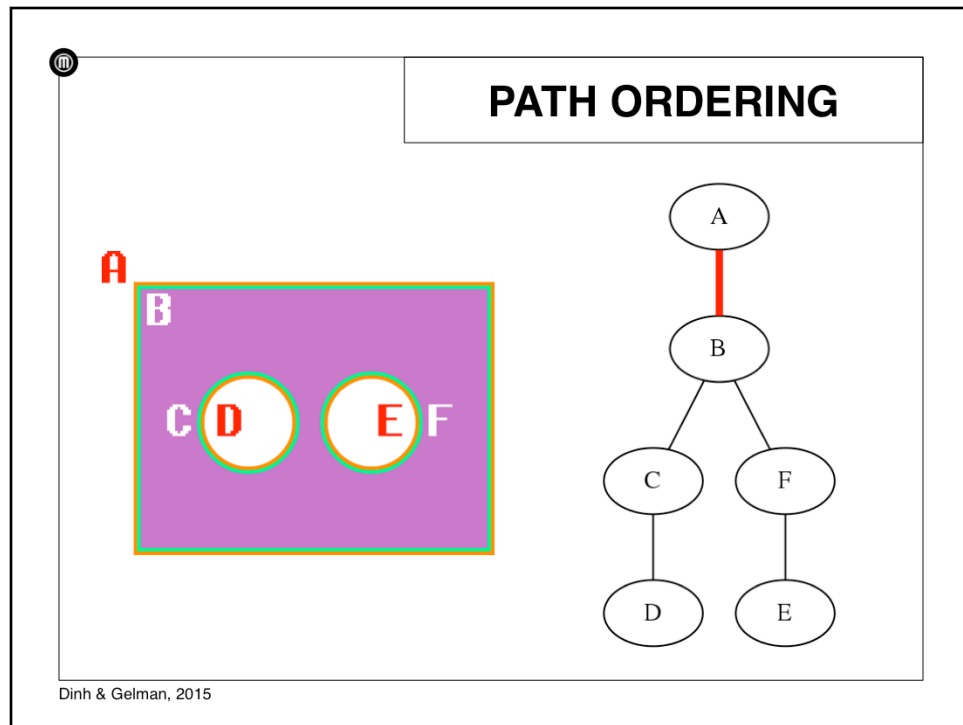
Here's how we satisfy this requirement.



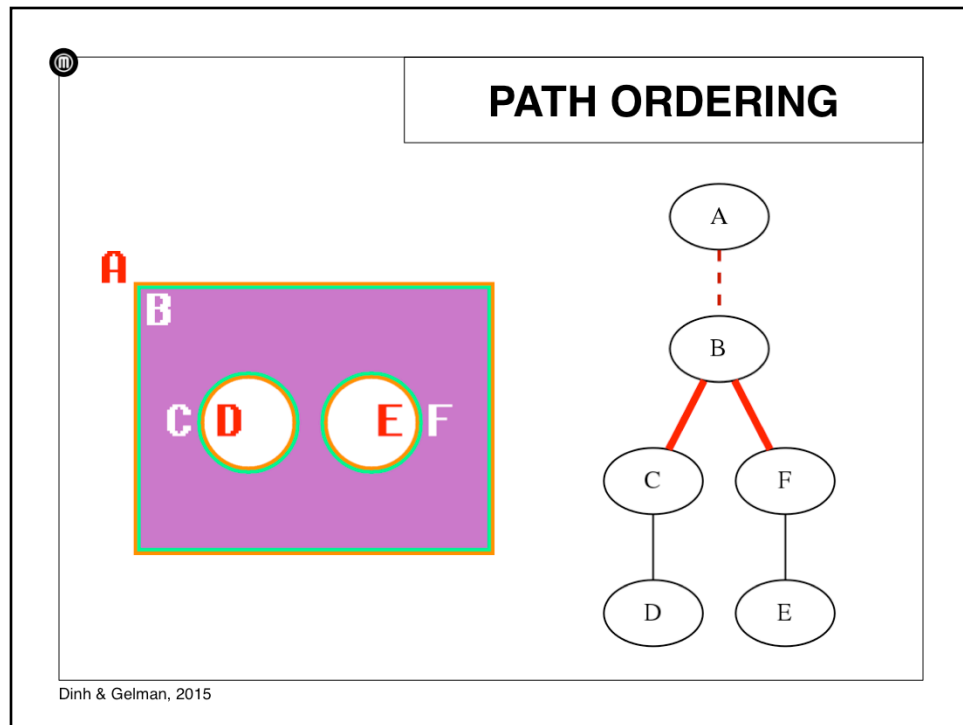
Build a tree out of the loops that make up the shells. Loop X is a child of loop Y if X is inside Y using the even-odd test. Traverse this tree depth-first.

The only choices to be made:

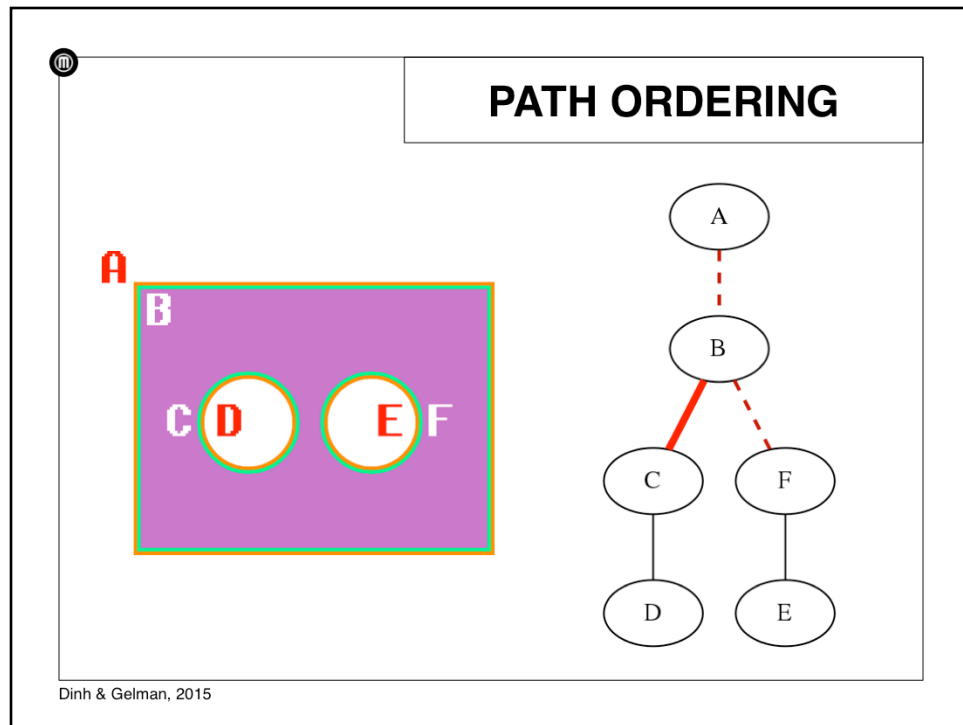
- To do a shell on the way down or on the way up?
- In which order to traverse siblings?



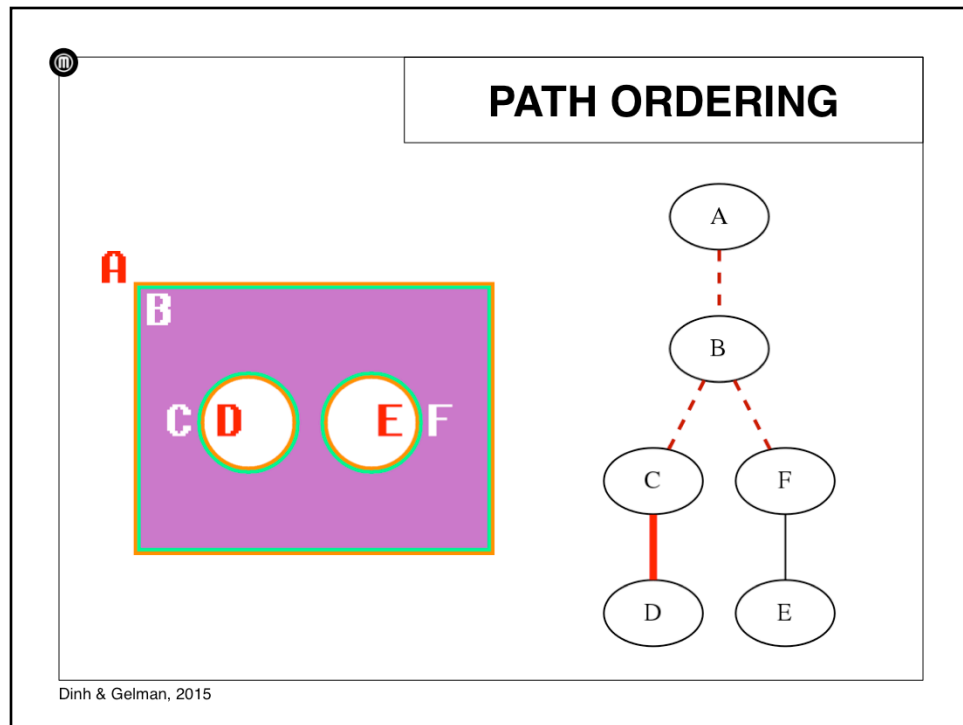
Start at the root. There is exactly one child. We want to do shells inner-most first. B is an inner shell and A is an outer shell. A has to be done in the way up.



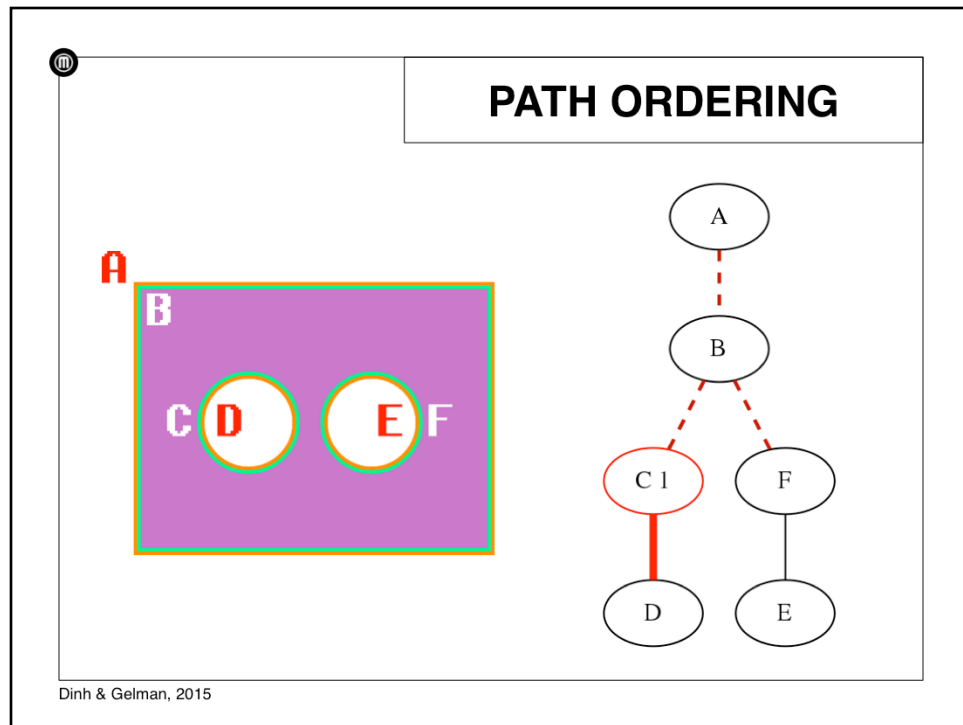
B has multiple children. How do we choose an order?



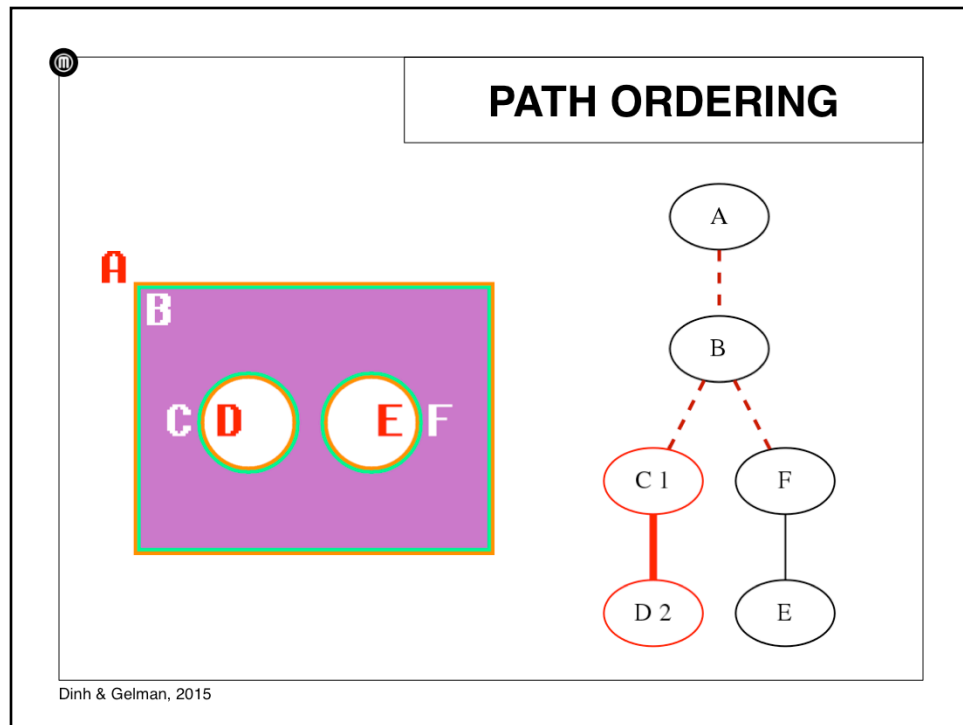
Which loop is closest to the previously printed point? This will be the last point of the previous layer or of another island. Assume for this example that C is closer.



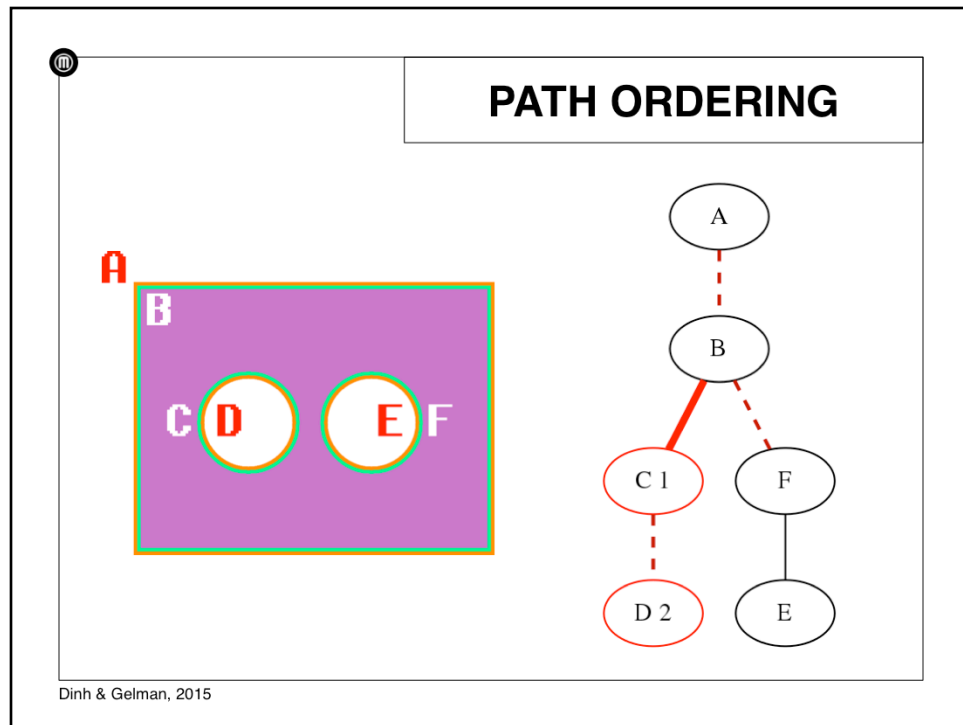
C is an inner shell and D is an outer shell. C has to be done on the way down.



C is the first shell to be printed.

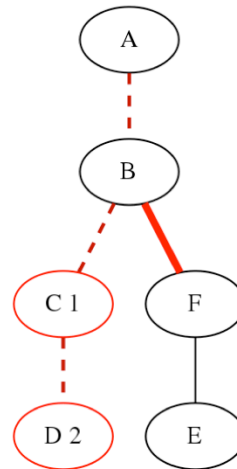
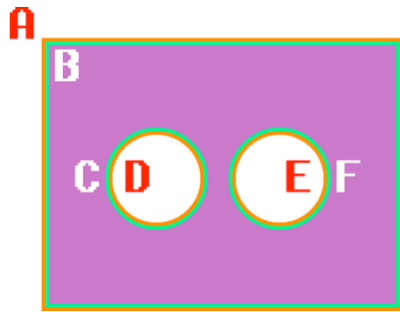


D has no children. Print it next.



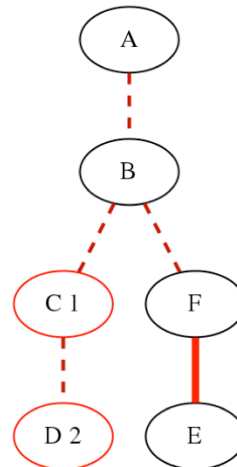
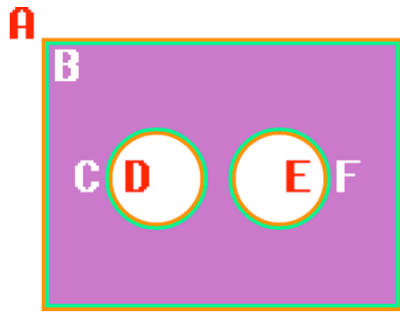
We continue the depth-first traversal...

PATH ORDERING



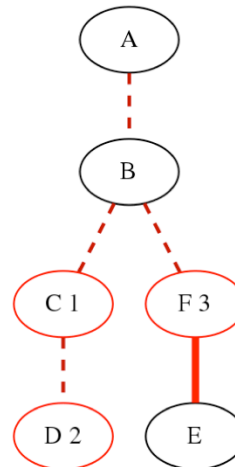
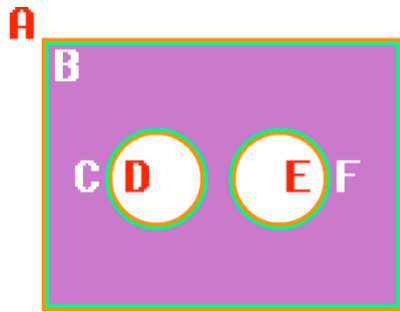
Dinh & Gelman, 2015

PATH ORDERING

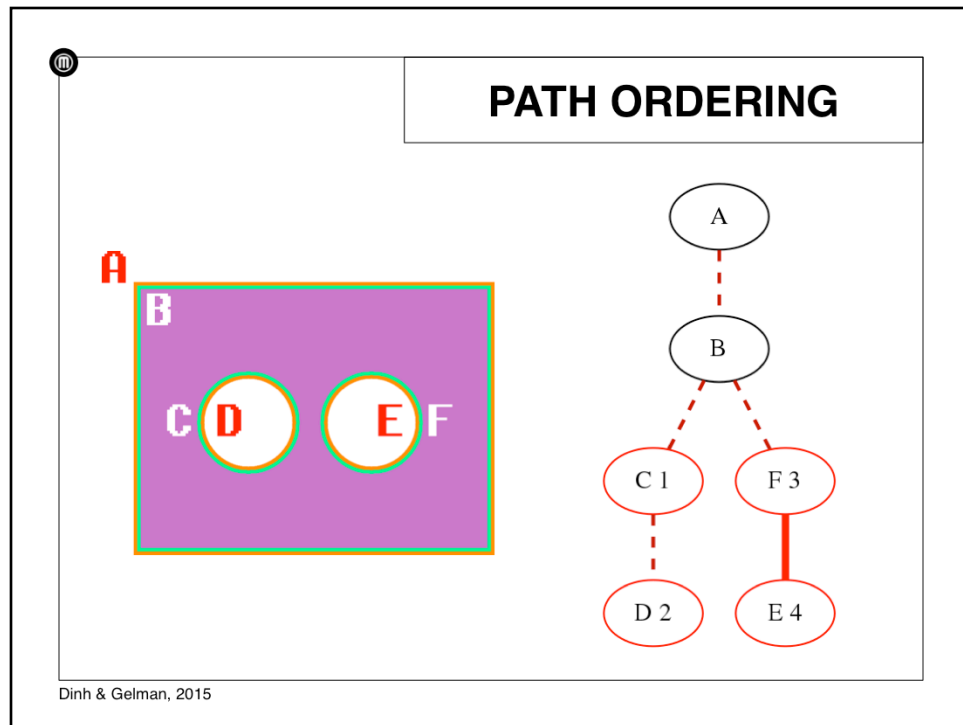


Dinh & Gelman, 2015

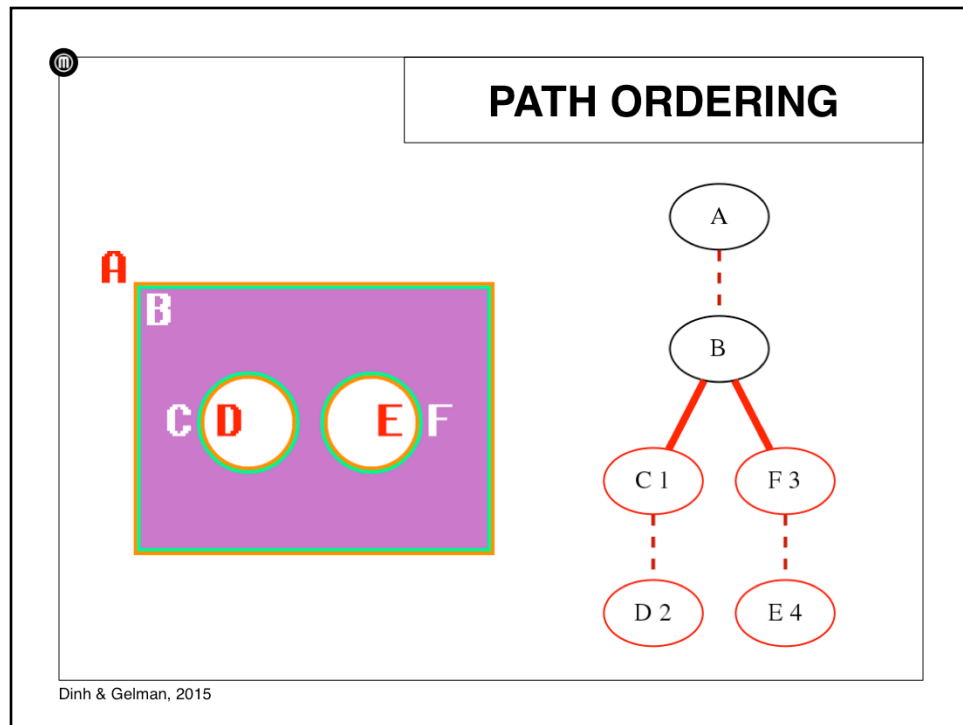
PATH ORDERING



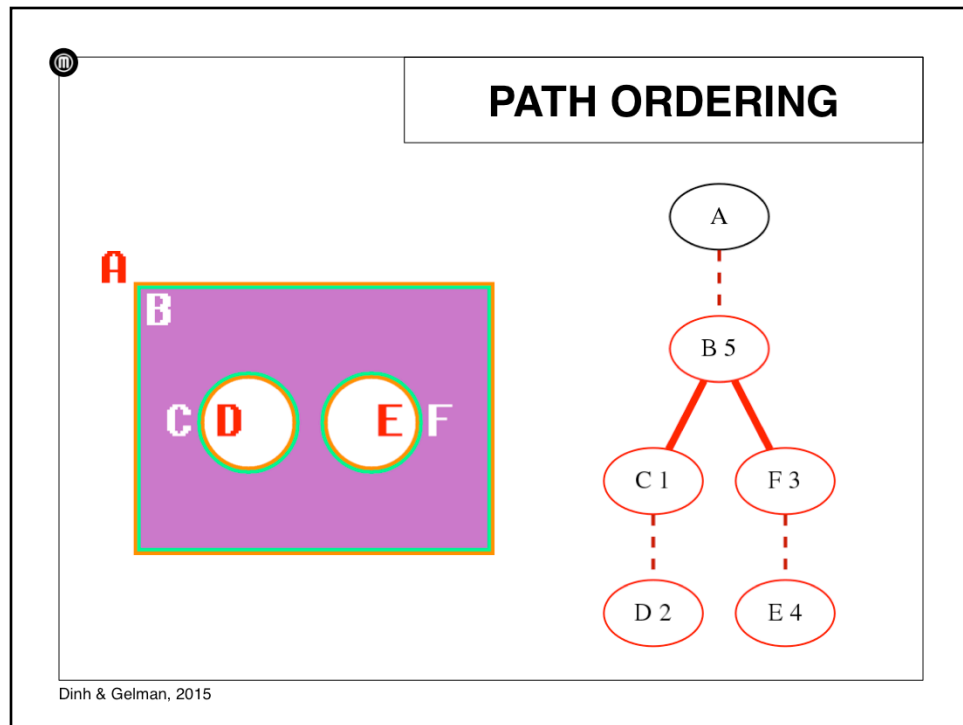
Dinh & Gelman, 2015



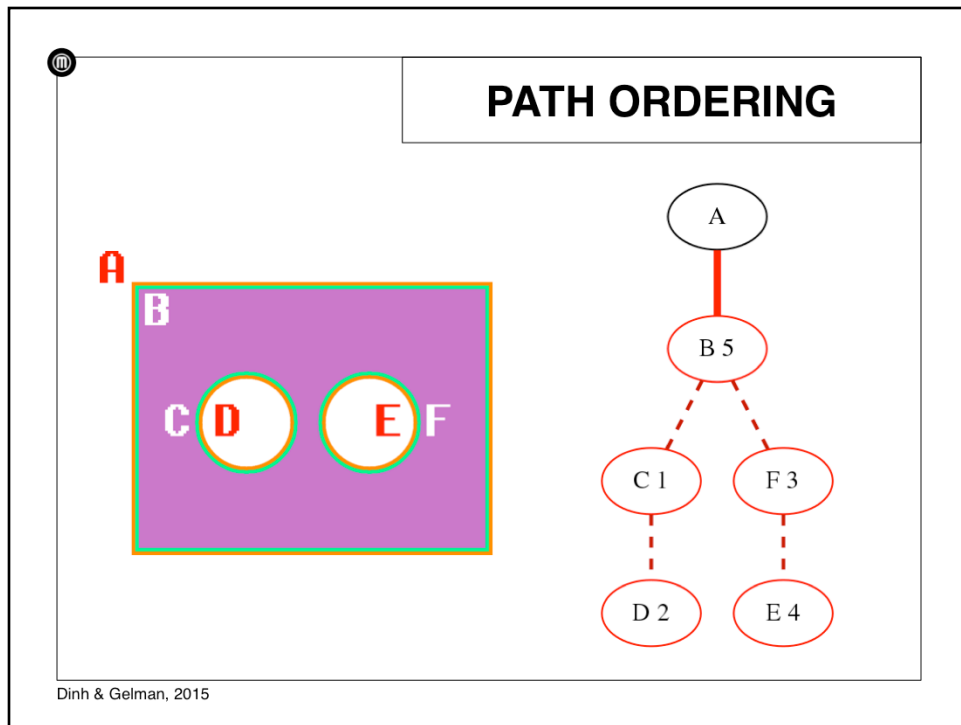
The same procedure is done for the other sub-tree.



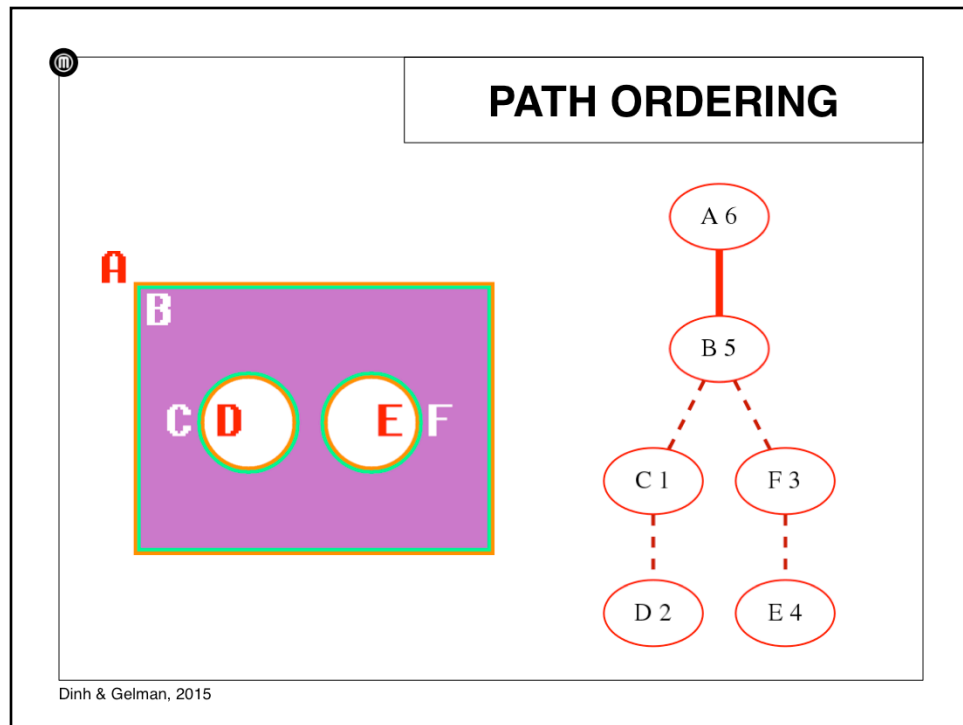
For any node, if we did not chose to print it on the way down, we print it on the way up, after all of its children have been traversed.



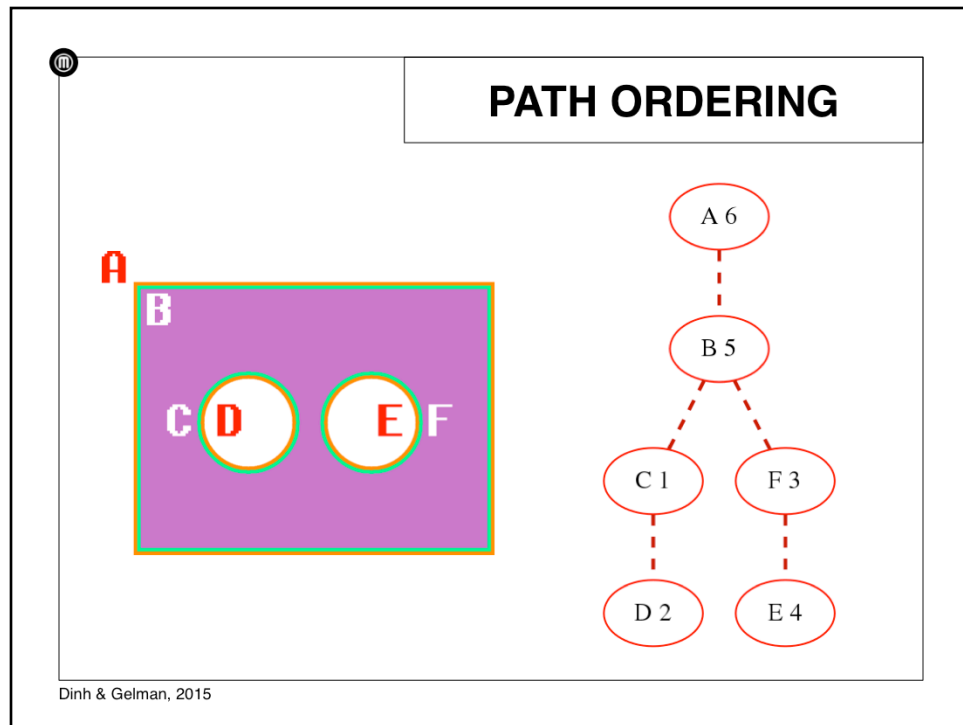
This means B is printed 5th, after C, D, F, and E.



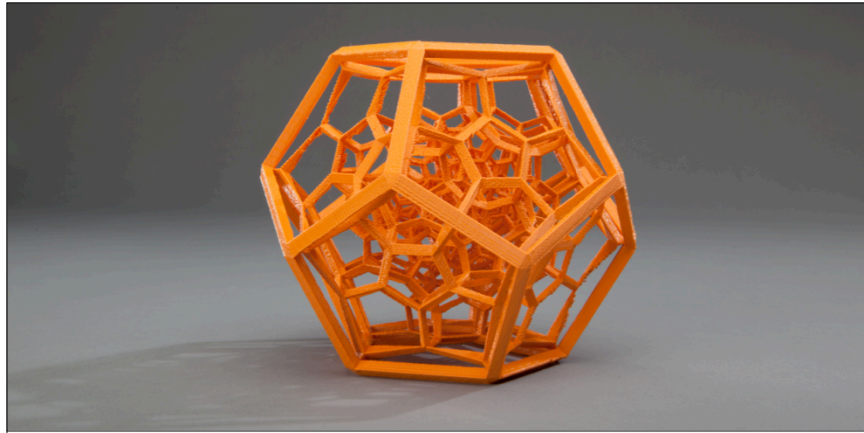
Finally...



The outermost shell, A, is printed last.

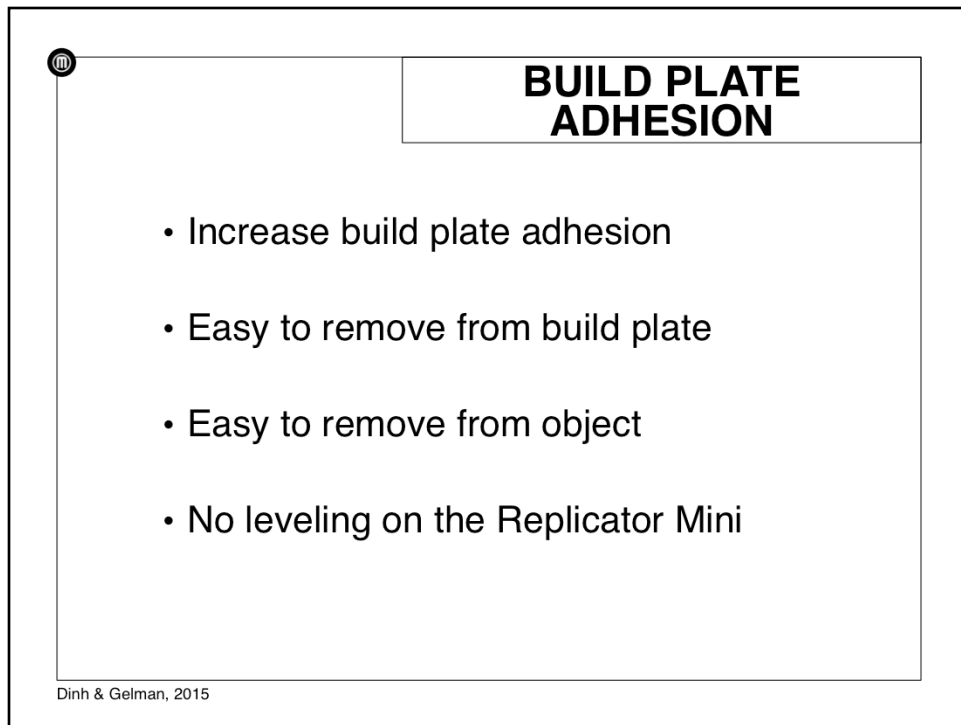


Final order for printing:
C, D, F, E, B, A



BUILD PLATE ADHESION

3D Printing with Plastic Filament



BUILD PLATE ADHESION

- Increase build plate adhesion
- Easy to remove from build plate
- Easy to remove from object
- No leveling on the Replicator Mini

Dinh & Gelman, 2015

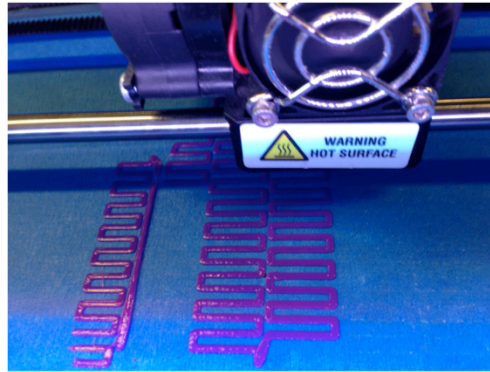
Once we've loaded our 3D model, we can start slicing. And the first layer is most often a raft.

Rafts were developed to solve warping on Rep2. Warping occurs as the plastic cools and contracts, and it's most apparent with objects that have a large footprint. Contraction typically occurs at the ends.

The goals of rafts were to increase build plate adhesion to prevent warping. But it had to be easy to remove from the build plate and easy to remove from the object, so that you don't get scarring on the object surface.

As an added benefit, it helped with the Mini which has no leveling. It allowed us to have non-parallelism of the plate to within 1.6mm.

REMOVABLE RAFTS



First raft layer comprised of thick filaments for easy removal, and widely spaced with short runs to prevent warping

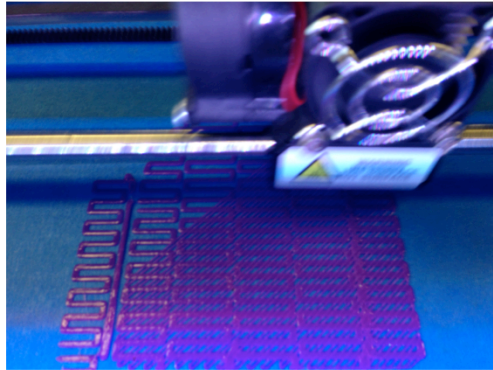
Dinh & Gelman, 2015

First, we have thick filaments with wide spacing for easy removal from build plate.

Reduce warping – no straight line longer than specified amount. This zigzag pattern are short runs.

Adhesion to object's first layer prevents object itself from warping.

REMOVABLE RAFTS

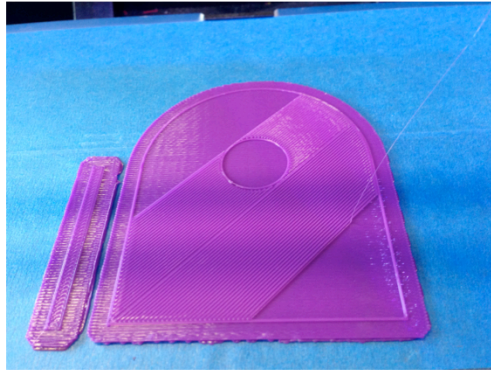


Two solid layers complete the raft

Dinh & Gelman, 2015

Next the solid layers of the rafts are printed – typically 2 layers.

REMOVABLE RAFTS




First layer of object is *drooped* onto raft to reduce bonding between raft and object

Dinh & Gelman, 2015

Finally, we start printing the object itself.

For an effective raft, the bonding between the raft and the object must be reduced such that there will be no scarring when the raft is removed from the object. But at the same time, there must be sufficient adhesion to ensure that the object itself does not warp. This bonding may be reduced by additional cooling time and by applying active cooling (fan) on the top surface of the raft.



CONFIGURABLE RAFTS

- Raft layers
 - Base (zigzag)
 - Interface
 - Surface
- Configurable settings for each
 - Number of layers
 - Density
 - Orientation
 - Thickness
 - Size and spacing of zigzag (base only)

Dinh & Gelman, 2015

The base layer is for maximum adhesion and to avoid warping.

Interface layers are to absorb any defects not absorbed by the base – e.g., ripped pieces of tape, or if homing was too close.

Surface layers should create an even surface on which object first layer is drooped.

Reference:

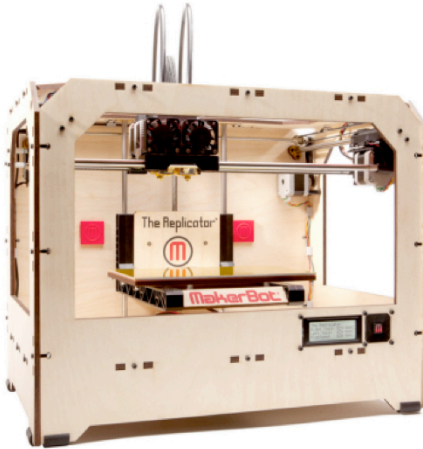
https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/13-MakerBot_Slicer_settings:_Rafts



MULTI-MATERIAL PRINTING

3D Printing with Plastic Filament

Dual extrusion for Multi-material



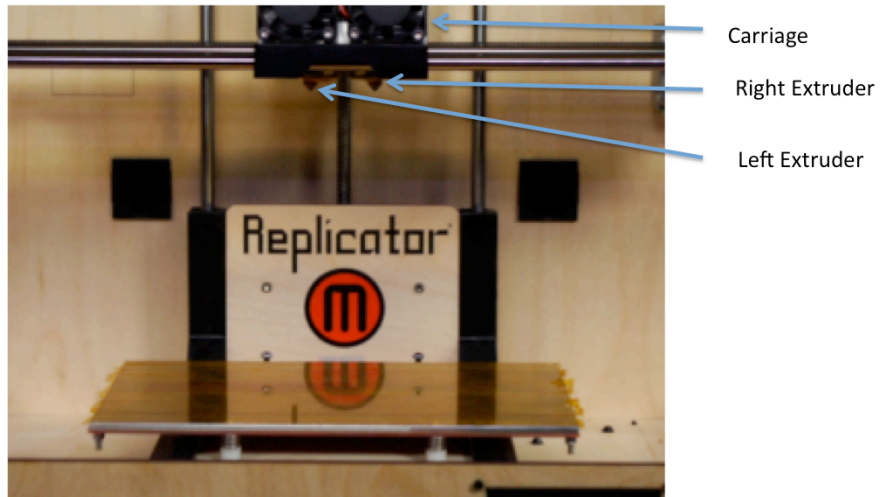
Lefebvre & Claux, 2015

1

In this part of the course we are going to focus on multi-material filament printers. For instance, this printer, the Makerbot Replicator 1, features two different extruders.

Most of the ideas I present here originate from our 2014 publication 'Clean Color'
<http://webloria.loria.fr/~jhergel/cleanColor.pdf>

Dual extrusion



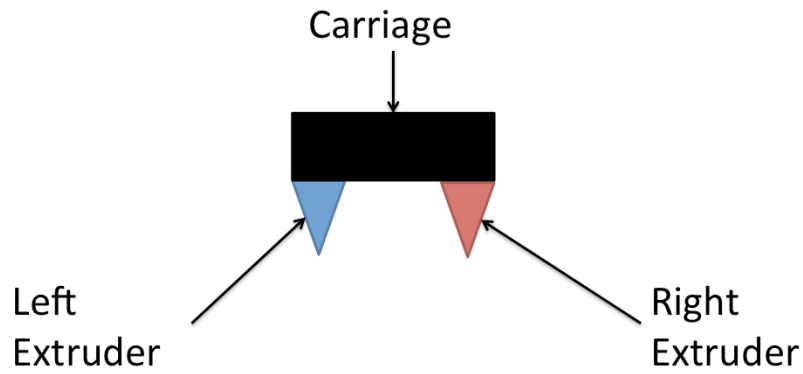
Lefebvre & Claux, 2015

2

The extruders are mounted on a single carriage, and two different plastic filaments can be used on the same print.

You can use different colors, or different plastics such as rigid / soft materials.

Dual Color Print, Symbols

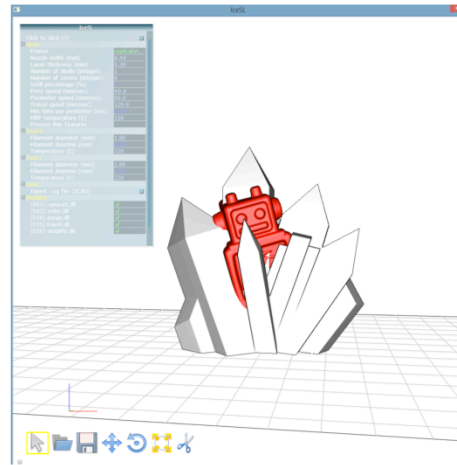


Lefebvre & Claux, 2015

In this presentation we will use these symbols to represent the carriage and the extruders

Slicer

- Generate toolpaths

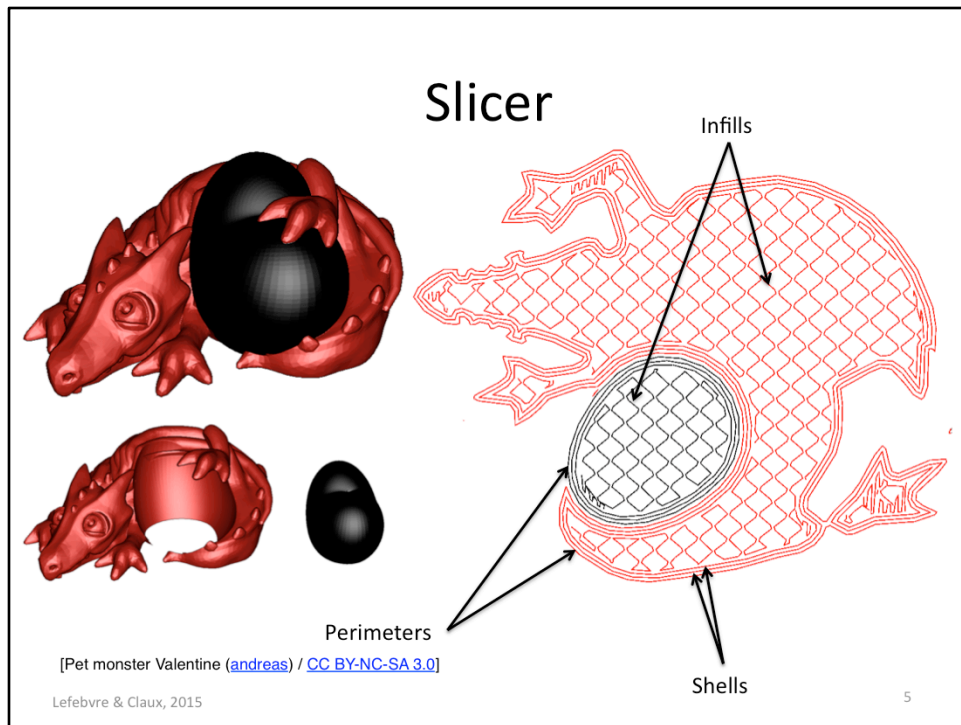


www.loria.fr/~slefebvr/icesl/

Lefebvre & Claux, 2015

4

Let's see how the slicer is going to deal with dual extrusion. We will be using IceSL in this case.



The slicer will generate two sets of toolpaths in each slice, one for each material.

Of course there are challenges! For instance, the ordering and the way the carriage travels will be even more important than when printing with a single material.

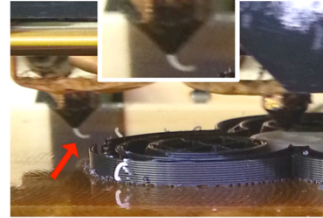
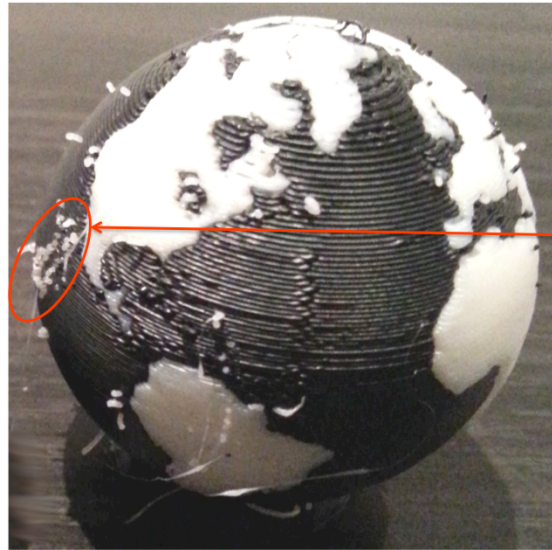
Attribution:

<http://www.thingiverse.com/thing:17204>

<http://www.thingiverse.com/andreas/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Defects



Oozing



Lefebvre & Claux, 2015

[Two Color World ([m6mafia](http://m6mafia.com)) / [GNU-GPL](http://gnu.org/licenses/gpl-2.0.html)]

6

Indeed, several defects can occur without special considerations.

The main issue is oozing, which we have already mentioned earlier. When an extruder stops printing, the melted plastic inside leaks out. In single material prints the effect is relatively limited. With dual printing, one extruder might be idle for several seconds, and oozing can be very significant.

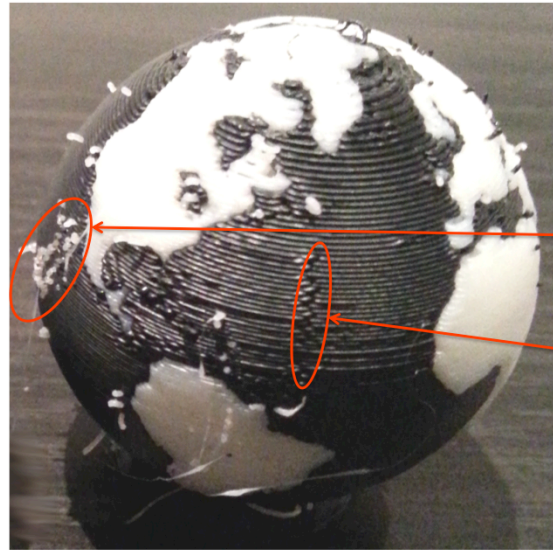
Attribution:

<http://www.thingiverse.com/thing:11660>

<http://www.thingiverse.com/m6mafia/about>

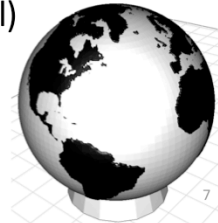
<http://www.gnu.org/licenses/gpl-2.0.html>

Defects



Oozing

Zippers (single and dual)



Lefebvre & Claux, 2015

[Two Color World ([m6mafia](http://m6mafia.com)) / [GNU-GPL](http://gnu.org/licenses/gpl-2.0.html)]

Besides oozing, another defect that appears on both single and dual prints are zippers. These occur when the extruder stops printing a perimeter. Very careful calibration can limit these, but they can be seen on most printed parts.

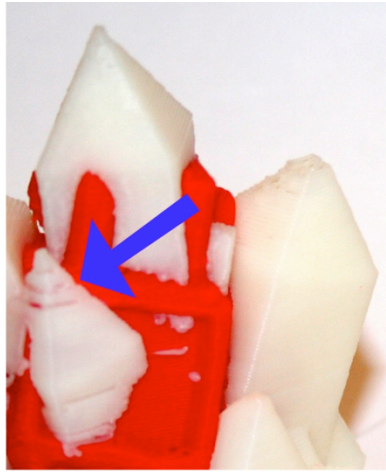
Attribution:

<http://www.thingiverse.com/thing:11660>

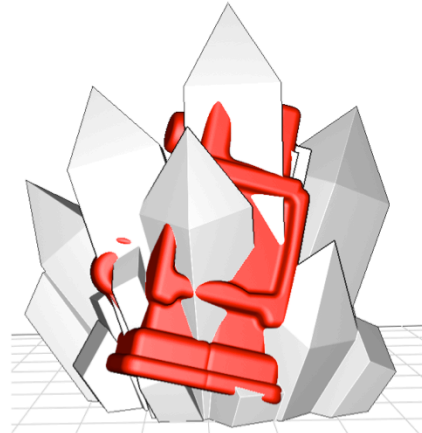
<http://www.thingiverse.com/m6mafia/about>

<http://www.gnu.org/licenses/gpl-2.0.html>

Defects: Holes



Lefebvre & Claux, 2015



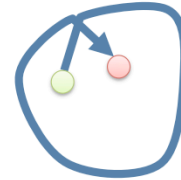
Robot-ice, YouMagine, Ultimaker

8

One last issue are holes: these appear because the plastic that oozed from the extruder is now missing! So when the extruder restarts, the plastic is not deposited as expected.

Typical approaches

- Zippers
 - Watertight zippers [\[Stratasys\]](#)
- Oozing
 - Cleaning station [\[Stratasys\]](#), tower [\[Kisslicer\]](#)
 - Cool idle extruder [\[Stratasys\]](#), [RepRapPro](#)
 - Drawback: time



An approach proposed by Stratasys to reduce zippers is to start and end print paths inside the print.

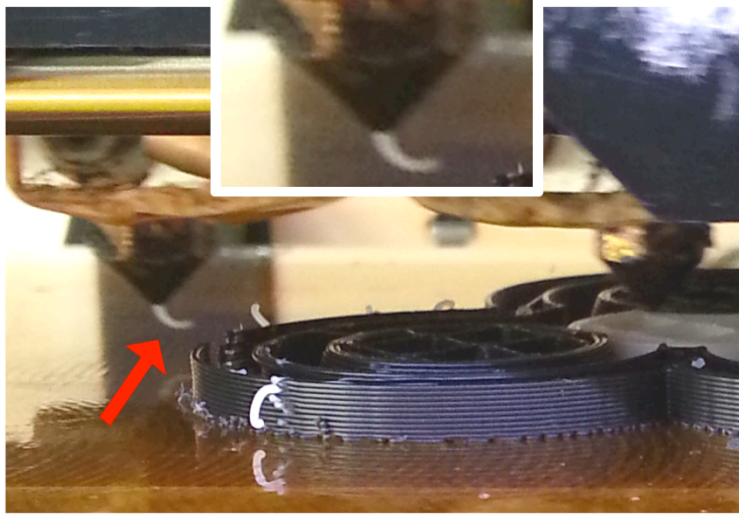
Regarding oozing, typically solutions are to use a cleaning station and reduce the temperature of the idle extruder. However this takes a long time, since the extruder first moves to the cleaning station, is shutdown, and then a delay is necessary to ensure all the plastic is gone. This is to be done once for each dual print layer. Other solutions, as mentioned before, involve a tower or walls to protect the print. These strongly reduce the defects but do not remove them entirely.

Outline

- Analysis
 - Oozing
 - Holes
 - Zippers
- Improvements
 - Azimuth Optimization
 - Rampart
 - Deal With Zippers
- Results

Let's now have a closer look at the defects, before we explain what can be done in more details

Analysis: Oozing

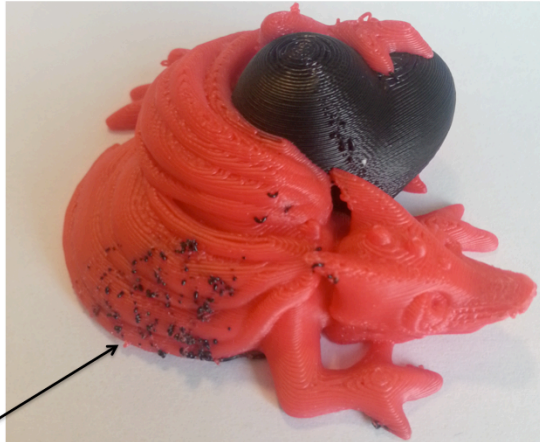


Lefebvre & Claux, 2015

11

Here is a picture taken during a dual printing session, just to show how significant oozing can be.
Several millimeters of plastic are dangling from the extruder after only a few seconds.

Analysis: Oozing



[Pet monster Valentine ([andreas](#)) / [CC BY-NC-SA 3.0](#)]

Lefebvre & Claux, 2015

12

The result are all these little blobs of plastic that are deposited on the other color. These cannot be trivially cleared since the materials mix when they are fused together.

Attribution:

<http://www.thingiverse.com/thing:17204>

<http://www.thingiverse.com/andreas/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Analysis: Oozing

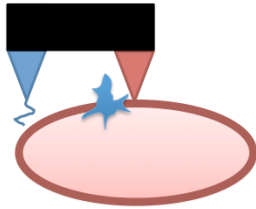
- Case A
- Case B

There are in fact two distinct cases where oozing is problematic.

Analysis: Oozing

- Case A

- Case B



Lefebvre & Claux, 2015

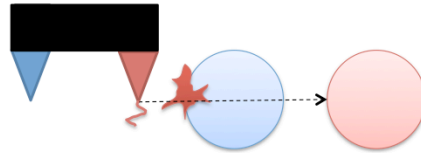
14

The first case is during print moves, when one extruder (red) prints and the other (blue) crosses the just printed path. This leaves a blob of plastic. It is a difficult case because we cannot really change the way the carriage moves: it has to deposit plastic along the red path.

Analysis: Oozing

- Case A

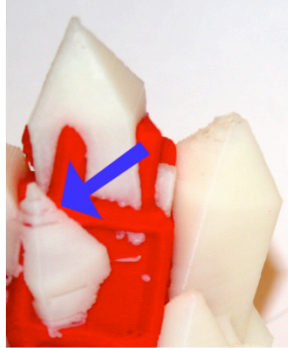
- Case B



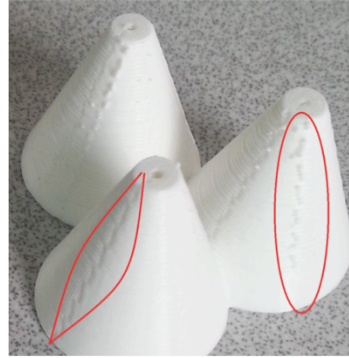
A second case is during travel moves: here we cross the blue part to reach the red part, but the red extruder is likely to deposit plastic along the blue perimeter. The case is slightly easier, since we can change the way the carriage travels.

Analysis: Holes And Zippers

- Holes
 - Consequence of Oozing
- Zippers
 - When extruder starts



Lefebvre & Claux, 2015



16

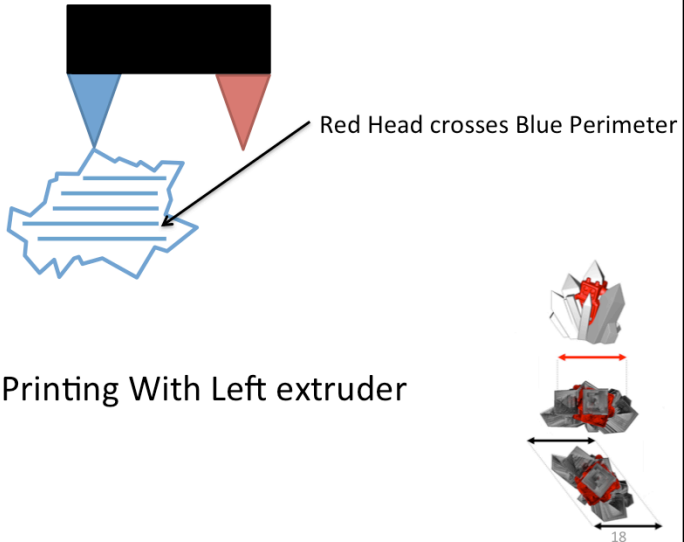
We will further consider holes and zippers, and how to reduce their effect.

Reducing these issues

- Azimuth Optimization
- Rampart
- Deal With Zippers

Ok, let's have a closer look at what we can do to reduce such problems,

Azimuth Optimization

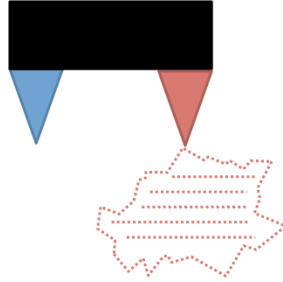


Lefebvre & Claux, 2015

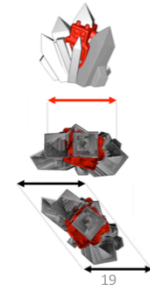
We observed that a simple change of orientation – azimuth angle -- can greatly improve quality.

When the blue extruder prints, the red extruder will move for some time just above the blue region. This is the effect we want to minimize.

Azimuth Optimization



Red head motion as blue head prints



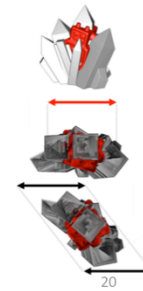
Lefebvre & Claux, 2015

We can easily compute the region spanned by the red extruder when the blue prints.

Azimuth Optimization



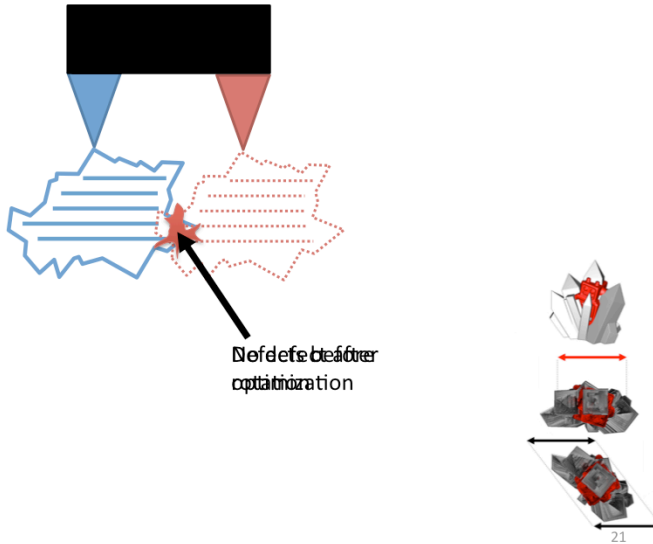
Overlap



Lefebvre & Claux, 2015

Our goal is to minimize the overlap region.

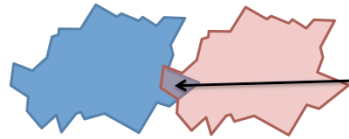
Azimuth Optimization



Lefebvre & Claux, 2015

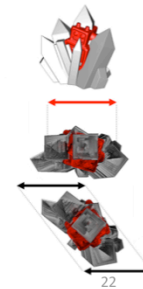
One simple way to do that is to change the orientation of the print!

Azimuth Optimization



Minimize the
volume of this
overlap

- Intersection computed by CSG for a fixed number of angles
- Fast GPU Implementation
(3 seconds on Robot-Ice Model)



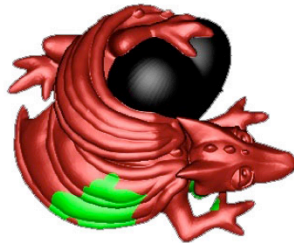
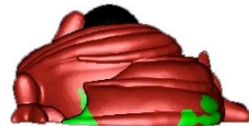
Lefebvre & Claux, 2015

Across all slices, this overlap region turns into a volume. We can compute this volume by fast boolean intersections between the object, and a translation of the object. As we will later see, our software is able to perform such operations very rapidly.

Azimuth Optimization

Without

With



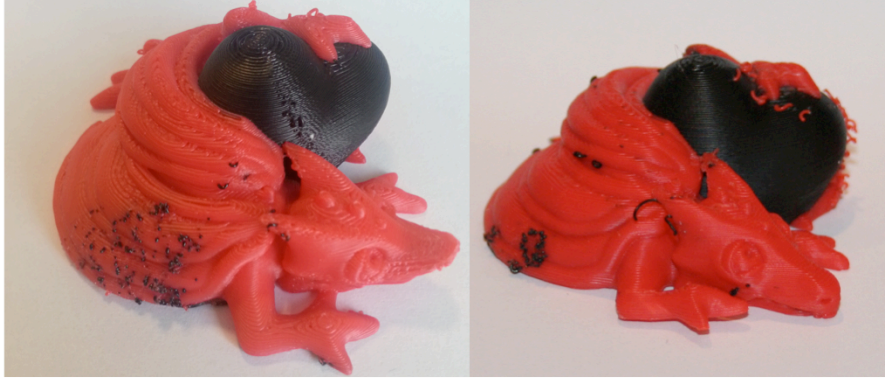
Overlap in green

Lefebvre & Claux, 2015

23

Here you can see on the left the overlap region in green before optimization, and after optimization. This alone greatly reduces the chance of oozing.

Azimuth Optimization



Without

With

Not always enough

Lefebvre & Claux, 2015

24

Here is a comparison on final prints.

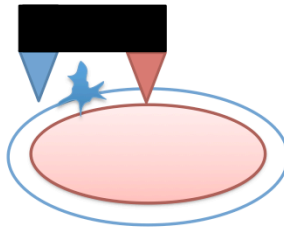
Reducing these issues

- Azimuth Optimization
- Rampart
- Zippers

We will now discuss a second ingredient, which is the rampart.

Rampart

- Catch ooze before extruder crosses the object
- As close as possible from the object
- Built layer by layer during the print

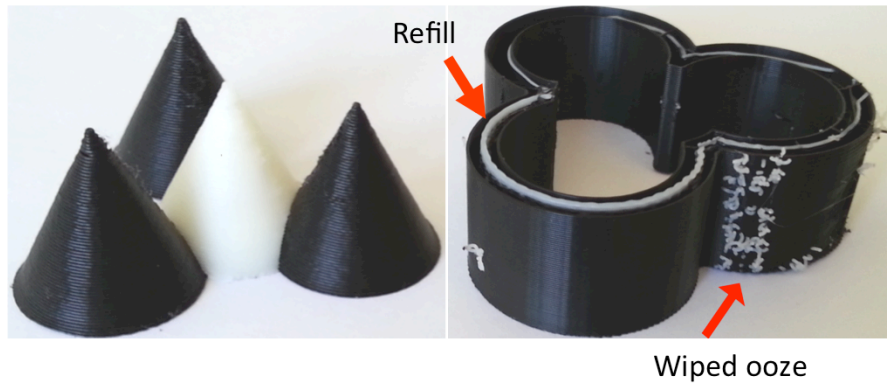


Lefebvre & Claux, 2015

26

The idea is similar to the oozing walls discussed earlier. We want to catch ooze before it reaches the surface. For this, we print a rampart surrounding the shape. This rampart will catch ooze just before the surface.

Rampart



Lefebvre & Claux, 2015

27

Here you can see the rampart after printing the cones.

Limitation

- The shape of the rampart



Lefebvre & Claux, 2015

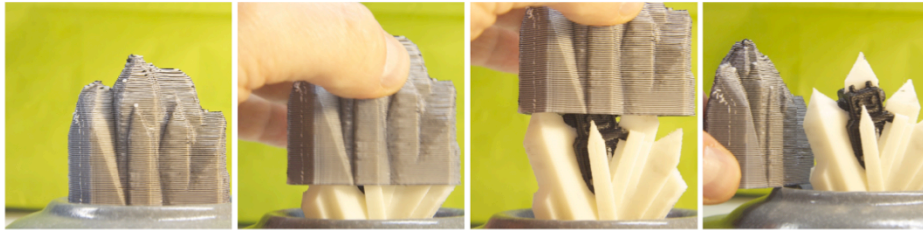
28

One limitation of the vertical rampart is that the shape of the rampart tends to be far away from the object in some cases.

Better!

Tight printable enclosures for additive manufacturing

S. Hornus, S. Lefebvre, J. Dumas, F. Claux



See also CURA ooze shields which are closely related.

Lefebvre & Claux, 2015

29

Here is a novel way to generate the rampart which makes it much closer to the surface, and faster to print.

<https://hal.inria.fr/hal-01141706/file/RR-8712.pdf>

CURA also feature ooze shields which use a similar approach to stay close to the surface.

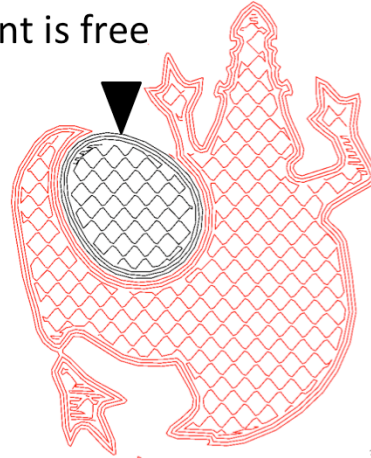
Reducing these issues

- Azimuth Optimization
- Rampart
- Zippers

Finally we need to find a way to deal with zippers.

Zippers

- Perimeters are cyclic
- Position of start/end point is free
- Change zipper locations
- They can be hidden



Lefebvre & Claux, 2015

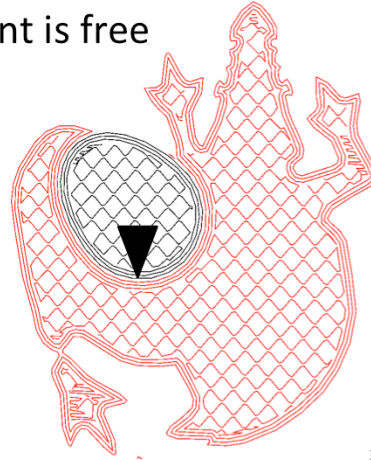
31

Zippers mostly occur along perimeters, which are – in our software – always cycles. Therefore we can freely choose the start/end point for printing a perimeter. We can thus hope to *hide* the zippers in less visible regions of the print.

Zippers

- Perimeters are cyclic
- Position of start/end point is free
- Change zipper locations
- They can be hidden

Ambient Occlusion



Lefebvre & Claux, 2015

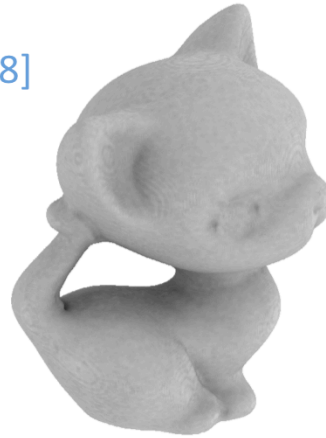
32

To determine what's visible or not we rely on a Computer Graphics technique called Ambient occlusion.

Zippers

- Ambient Occlusion [Zhukov*98]

- Used to enhance realism of a render
- Darker zones are less visible
- Hide defects in occluded zones



[Kitten (MBCook) / CC BY-SA 3.0]

Lefebvre & Claux, 2015

33

This computes a map along the surface which is darker when it is less visible and brighter otherwise. This correspond to how many viewpoints can actually see each surface points. We can compute this quickly on modern GPUs.

Attribution:

<http://www.thingiverse.com/thing:12694>

<http://www.thingiverse.com/MBCook/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Results

With azimuth only



With our technique



Lefebvre & Claux, 2015

34

Here are some results, note the significant improvement. These objects have not been cleaned in any way.

Results

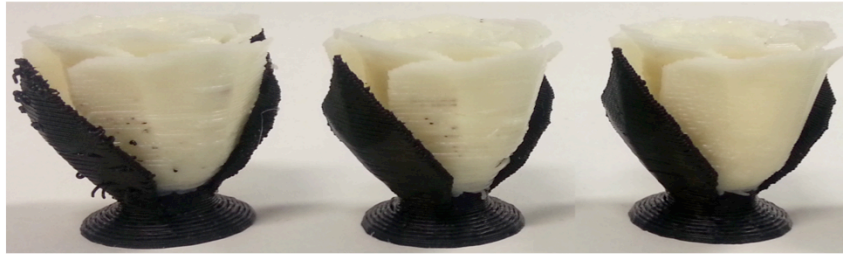


Lefebvre & Claux, 2015

35

Two other results in challenging cases that use very contrasted colors.

Results



Without our
technique

With rampart
only

With our
technique

[Jillian's Rose ([Bob_East](#)) / [CC BY-SA 3.0](#)]

Lefebvre & Claux, 2015

36

A comparison using the various ingredients.

Attribution:

<http://www.thingiverse.com/thing:26417>

http://www.thingiverse.com/Bob_East/about

<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Remember the Earth?



Lefebvre & Claux, 2015

[Two Color World ([m6mafia](#)) / [GNU-GPL](#)]

37

The earth model was quite bad.

Results: Earth



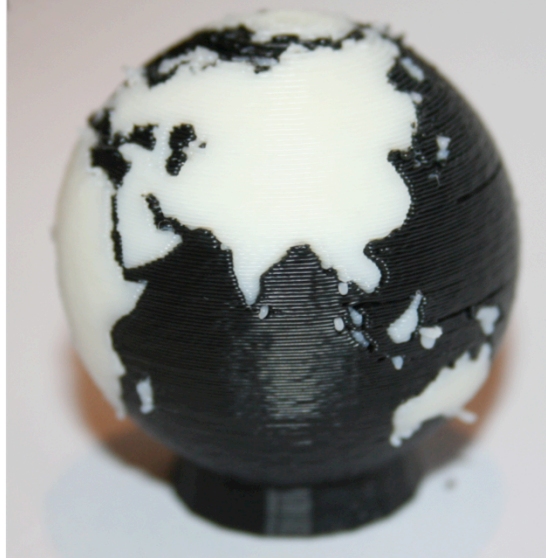
Lefebvre & Claux, 2015

[Two Color World ([m6mafia](#)) / [GNU-GPL](#)]

38

Here is our result.

Results: Earth



Lefebvre & Claux, 2015

[Two Color World ([m6mafia](#)) / [GNU-GPL](#)]

39

Another view.

Results: Earth



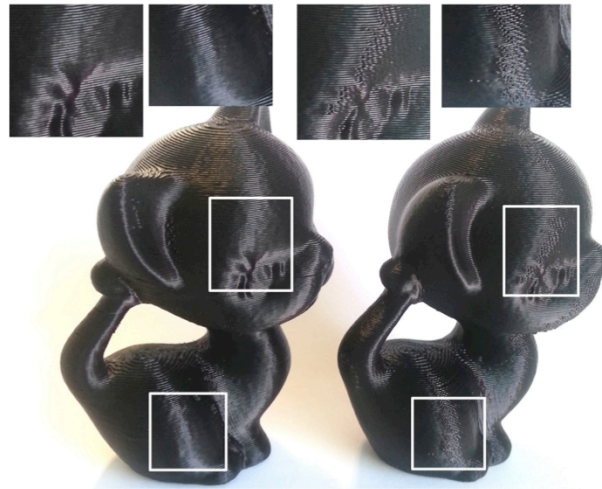
Lefebvre & Claux, 2015

[Two Color World ([m6mafia](#)) / [GNU-GPL](#)]

40

And another view.

Results



With Our Technique

Without

Lefebvre & Claux, 2015

[Kitten ([MBCook](http://www.thingiverse.com/thing:12694)) / [CC-BY-SA 3.0](http://creativecommons.org/licenses/by-nc-sa/3.0/)]

41

Our approach also removes most zippers – in fact they are not removed but hidden from view. This greatly improves the finish of shiny surfaces.

Attribution:

<http://www.thingiverse.com/thing:12694>

<http://www.thingiverse.com/MBCook/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>

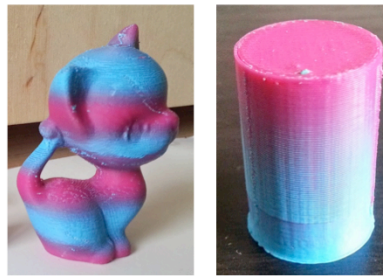
Dual versus mixing

- Mixing is possible!

- Dithering style

Dual-Color Mixing for Fused Deposition Modeling Printers
[Reiner et al, 2014]

- Layering style



Lefebvre & Claux, 2015

Print and picture by S. Lefebvre

42

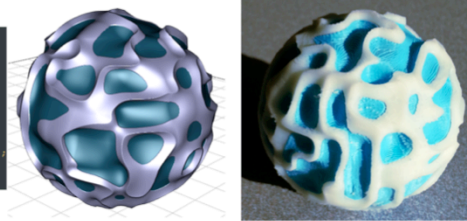
So far we only looked at printing one color or the other. There are ways to mix the two colors!

Reiner and colleagues presented a very interesting approach where dithering is used to bring one color visually in front of the other.

<https://cg.ivd.kit.edu/publications/2014/DCM/DualColorMixing.pdf>

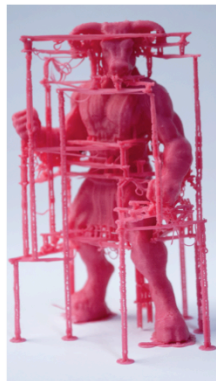
We also experimented with a mixed layering approach, where the plastic flow is modified to deposit each layer twice, in proportions such that the total amount of plastic matches what's normally required. By carefully calibrating the printer color gradients can be obtained, as shown here.

```
sphA = implicit(v(-30,-30,-30), v(30,30,30), [{  
  float minDistanceSphereTracing=0.001;  
  float perturb(vec3 p)  
  {  
    return 5.0*abs(noise(p/7.0+2.0));  
  }  
  float distanceEstimator(vec3 p)  
  {  
    return 0.3*(max(sphere(p,26),-sphere(p,23)) + perturb(p));  
  }  
}])
```



MODELING FOR FILAMENT-BASED 3D PRINTING

10:30am – 11:45am



[Minotaur ([ajolivette](http://www.thingiverse.com/thing:46646)) / [CC BY-SA 3.0](http://creativecommons.org/licenses/by-sa/3.0/)]

OVERHANGS AND SUPPORTS

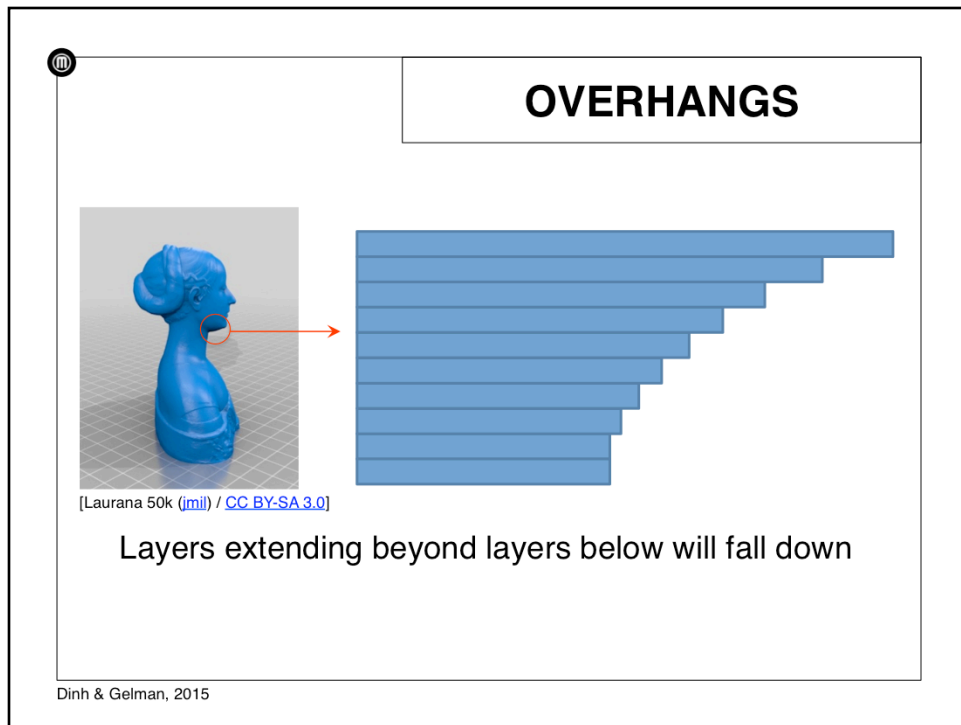
Modeling for Filament-Based 3D Printing

Attribution:

<http://www.thingiverse.com/thing:46646>

<http://www.thingiverse.com/ajolivette/about>

<http://creativecommons.org/licenses/by-sa/3.0/>



Overhangs are parts of the model where the layer above extends well beyond the layers below, and gravity will cause the printed plastic to droop down, like printing over air.

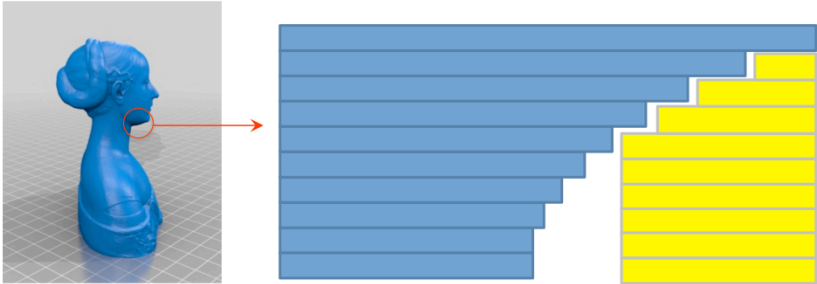
Attribution:

<http://www.thingiverse.com/thing:1978>

<http://www.thingiverse.com/jmil/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

SUPPORTS



[Laurana 50k ([jmil](#)) / [CC BY-SA 3.0](#)]

Layers extending beyond layers below will fall down
They need to be **Supported**

Dinh & Gelman, 2015

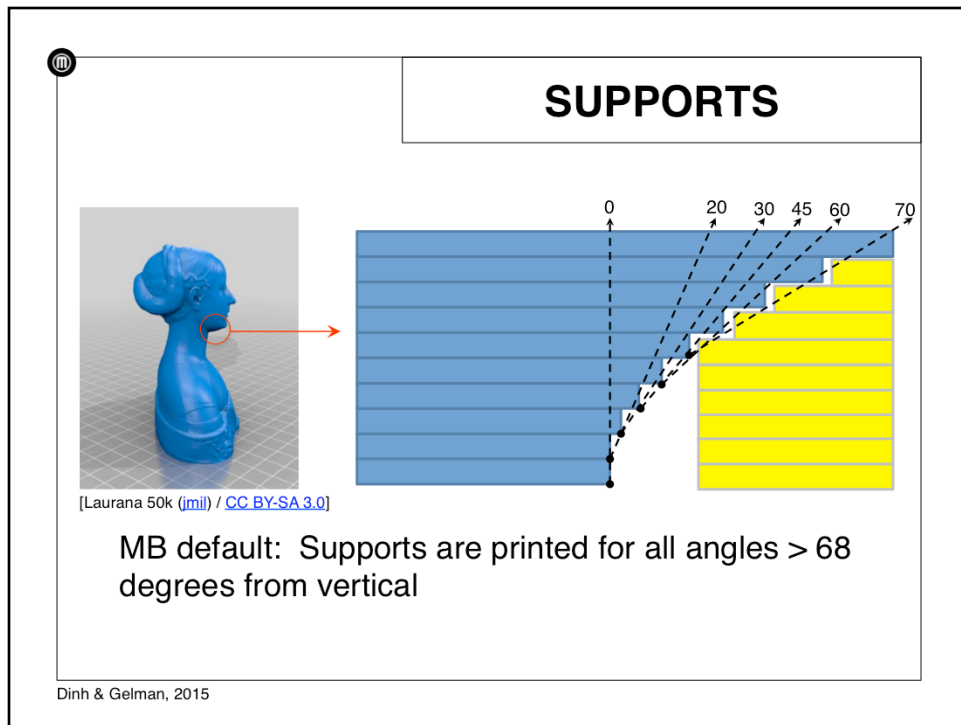
Overhangs need to be supported so that we are not printing over air.

Attribution:

<http://www.thingiverse.com/thing:1978>

<http://www.thingiverse.com/jmil/about>

<http://creativecommons.org/licenses/by-sa/3.0/>



However, if the overhang is small enough, then it will not droop down, and we do not need to support such overhangs.

This is typically determined by the angle formed between the overhang and the closest point in the layer below.

Our default is to leave all angles < 68 degrees from vertical unsupported.

References:

https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/15-MakerBot_Slicer_settings:_Support

Attribution:

<http://www.thingiverse.com/thing:1978>

<http://www.thingiverse.com/jmil/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

FILAMENT-BASED SUPPORT GENERATION

- Factors to consider
 - Angle at which supports begin
 - Support pattern
 - Density
 - Space between model and supports
- Models are printed from the bottom-up, but support generation needs to account for floors in slices above

Dinh & Gelman, 2015

Pattern often affects removability


Denser supports will support more of the floor, but uses more plastic, takes longer to print, and may be more difficult to remove.

Space between model and supports:


- More space makes it easier to remove supports, but will mean that less of the floors above will be supported which could result in a drooping floor.

References:

https://www.makerbot.com/support/new/04_Desktop/Knowledge_Base/Using_Custom_Slicing_Profiles/15-MakerBot_Slicer_settings:_Support



SUPPORTS



Filament-based supports result in simpler toolpaths

Dinh & Gelman, 2015

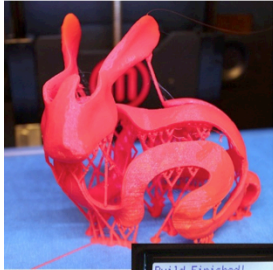
Unlike many other slicers, ours does not model supports as a mesh. Instead, supports are just filaments loosely laid down – this makes the toolpath simpler (no shells and fewer retractions).

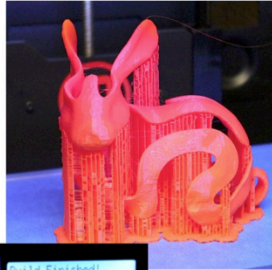
The object surface is also more uniformly supported, unlike cone-based supports where the contact is a point.

This example shows a change in version 2.2.0 last fall where we significantly reduced the amount of support we printed.

References:

<http://www.makerbot.com/blog/2013/06/12/makerware-2-2-0-preview/>





Build Finished!
 Build Time 03h31m

Build Finished!
 Build Time 04h33m

[Bunny Peel with meshmixer Support ([meshmixer](#)) / [CC BY-SA 3.0](#)]

Filament-based supports can result in longer print times and more plastic usage, but are less prone to falling support columns

Dinh & Gelman, 2015

Filament-based supports may use more plastic and take longer to print than alternative approaches, such as MeshMixer and INRIA's scaffolds.

One concern with these narrow column-based supports is that they will tend to fall over during printing.

References:

"Clever Support: Efficient Support Structure Generation for Digital Fabrication", Juraj Vanek, Jorge A. G. Galicia, Bedrich Benes, Computer Graphics Forum 33(5): 117-125 (2014)

Attribution:

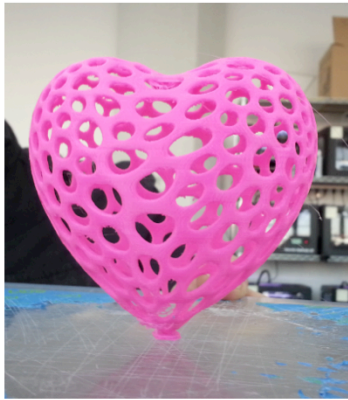
<http://www.thingiverse.com/thing:131054>

<http://www.thingiverse.com/meshmixer/about>

<http://creativecommons.org/licenses/by-sa/3.0/>



AVOIDING SUPPORTS



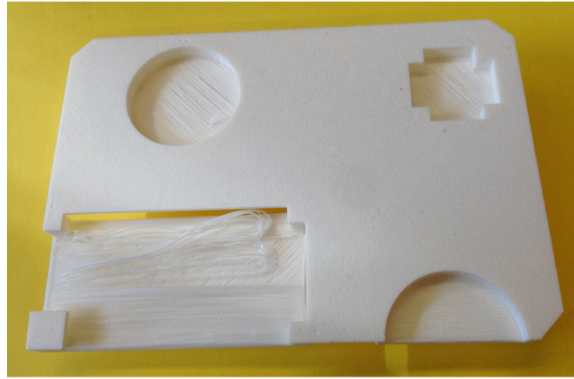
[Print by Stephen Detsch, MakerBot]

Optimize overhangs

Dinh & Gelman, 2015

It is possible to construct a model that prints completely unsupported, exploiting the angle at which gravity is overcome by adhesion and the contraction of plastic.

AVOIDING SUPPORTS: BRIDGING



Omit supports under bridges

Dinh & Gelman, 2015

Another way to reduce supports is to print bridges. These are areas of the surface that sit over open air, but are short enough and are surrounded by parts of the shell which can be used to anchor a bridge. The bottom of the object is shown in this example.

Sometimes fails – can be due to temperature and contraction properties of plastic.



BRIDGING FACTORS

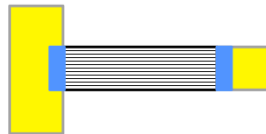
- Anchors: where bridges attach to islands of the layer below
- Span direction: given a bridgeable floor area, in which direction should bridges extend?
- Maximum length of bridges

Dinh & Gelman, 2015



BRIDGE ANCHORS

- Location: where should **anchors** intersect shells?
- Size of anchoring islands



Islands in yellow with grey shells
Anchors in blue with a bridge in between

Dinh & Gelman, 2015

The start and end of bridges are typically on the shell. The location of anchors and making room for them on the shells are required.

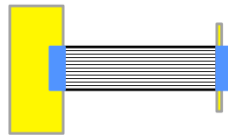
Anchoring island must be wide and deep enough to support a bridge

A successfully bridged extrusion is anchored at both ends. An anchor is the portion of the layer below that is adjacent to the region to be bridged.



BRIDGE ANCHORS

- Location: where should **anchors** intersect shells?
- Size of anchoring islands



Islands in yellow with grey shells
Anchors in blue with a bridge in between

Dinh & Gelman, 2015

Anchors must have sufficient depth for the extrusions of the bridge to attach



BRIDGE ANCHORS

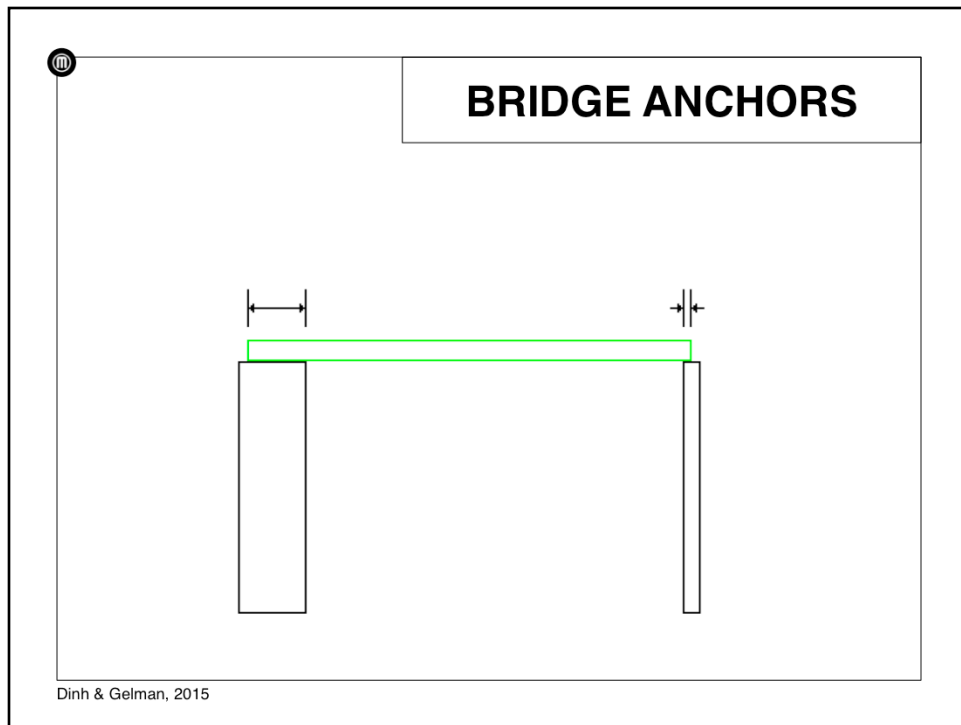
- Location: where should **anchors** intersect shells?
- Size of anchoring islands




Islands in yellow with grey shells
Anchors in blue with a bridge in between

Dinh & Gelman, 2015

Islands that are too small can't support bridges



This is a cross section view of a hypothetical model as seen from the side. The potential bridge is shown in green. The anchor on the left is deep enough for the bridge filaments to attach. The anchor on the right is shallow and would not result in a well attached bridge.



BRIDGE SPAN DIRECTION

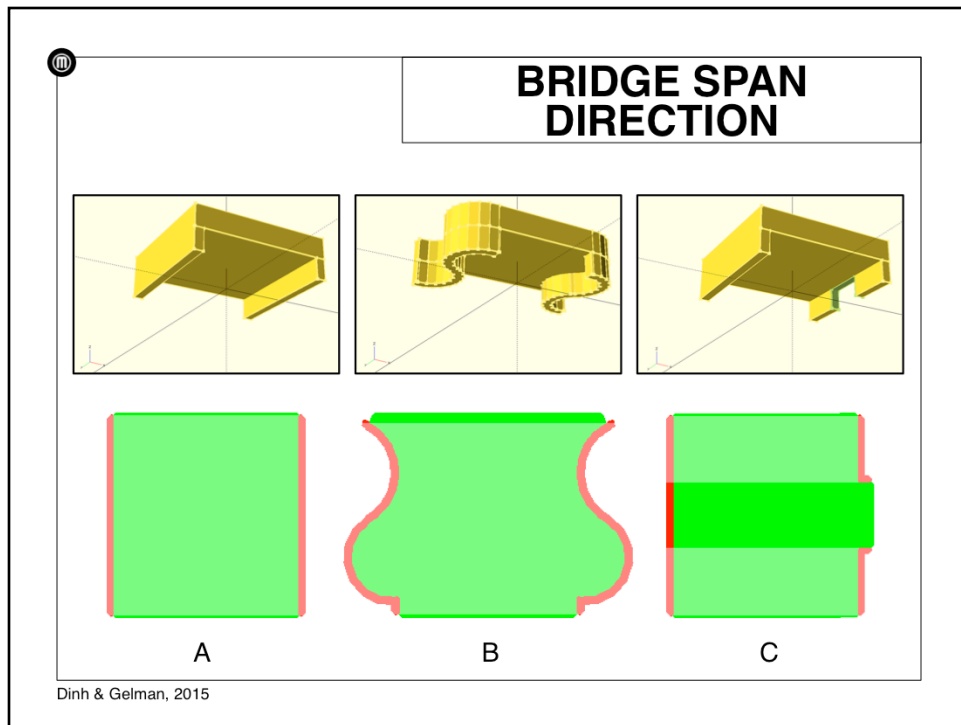
- Goal
 - Maximize anchored extrusions
- Affected By
 - Shape of the region to bridge
 - Size, shape, and arrangement of anchors

Dinh & Gelman, 2015

In other words, we want to maximize good bridges that don't collapse. For a given, potentially bridgeable, region, different directions of the bridge span will result in different areas being bridged, and we essentially want to maximize that area while ensuring that the bridge doesn't collapse.

The start and end of bridges are typically on the shell. Programmatically computing the location and making room for bridge anchors on the shell(s) is what is required to implement bridges.

Anchoring island must be wide and deep enough to support a bridge



Top:

A) A simple example of bridging – two straight aligned anchors

B) A slightly more complex case of bridging. Here, the anchors are S shaped

C) In this case, the bridge would be supported by three anchor regions.

Bottom:

These are top-down views of the model as seen by the bridging algorithm. Anchors are marked in red and the region to be bridged is marked in green.

A) In this case, bridge filaments would be aligned horizontally.

B) Like in the previous example, the bridge filaments would be aligned horizontally. Notice that the portion of the region that is considered anchored has a paler green color. The part of the region near the top of the image is not considered to be anchored sufficiently.

C) As in the previous examples, the filaments would be arranged horizontally. Notice

Typical Approach

- Makerware
 - ✓ Reliable
 - ✗ High material usage
 - ✗ Difficult to remove



Lefebvre & Claux, 2015

[Minotaur ([ajolivette](#)) / [CC BY-SA 3.0](#)]

2

The standard approaches work great, but tend to use a lot of plastic.

Attribution:

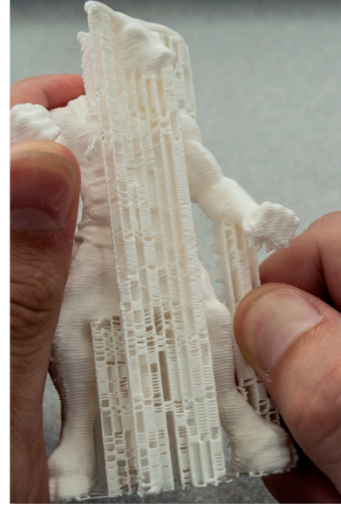
<http://www.thingiverse.com/thing:46646>

<http://www.thingiverse.com/ajolivette/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

Typical Approach

- Makerware
 - ✓ Reliable
 - ✗ High material usage
 - ✗ Difficult to remove



Lefebvre & Claux, 2015

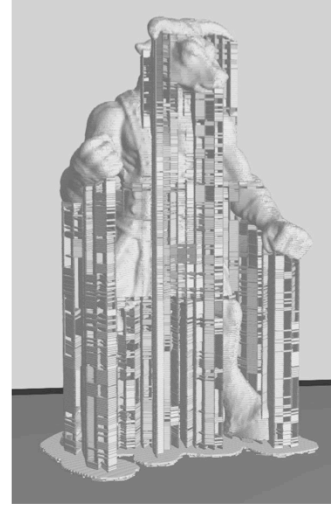
[Minotaur ([ajolivette](#)) / [CC BY-SA 3.0](#)]

3

Here is an example of typical support.

Typical Approach

- Makerware & Others
 - ✓ Reliable
 - ✗ High material usage
 - ✗ Difficult to remove



Lefebvre & Claux, 2015

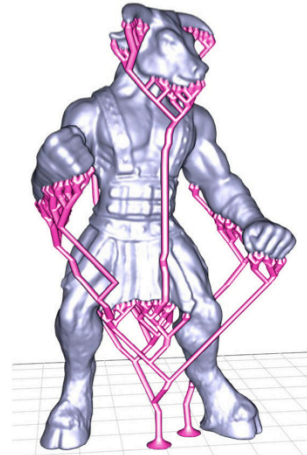
[Minotaur ([ajolivette](#)) / [CC BY-SA 3.0](#)]

4

They are however very reliable, which is of course very important for consumer-level products.

Recent Work

- MeshMixer (Autodesk), Vanek et al. [2014]
 - ✓ Low material usage
 - ✗ Imbalanced structure
 - ✗ Requires manual tuning
 - ✗ Subtle print parameters



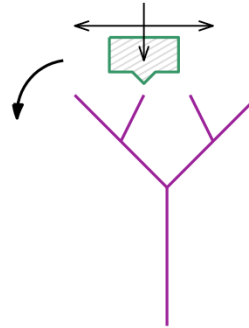
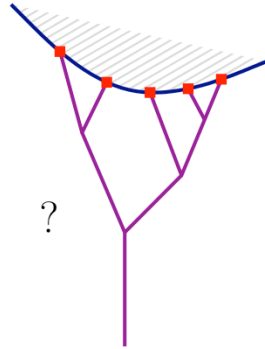
Lefebvre & Claux, 2015

5

Some recent work attempts to reduce significantly material usage. This is an example of MeshMixer supports. They are very interesting because they support large surfaces while being very small. Unfortunately, these structures are not balanced and can fail during printing, as we will see later.

Trees – Pros and Cons

✓ Support/length ratio ✗ Sensitive to **torque**



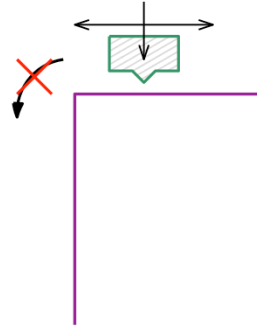
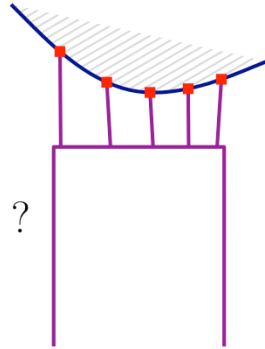
Lefebvre & Claux, 2015

6

We measure the effectiveness of a support structure by the ratio of its length to the number of supported points. Trees have a very good ratio. Unfortunately, as they grow they become very sensitive to torque, and start oscillating which can lead to rupture.

Think With Bridges

✓ Support/length ratio ✓ Resistant to torque



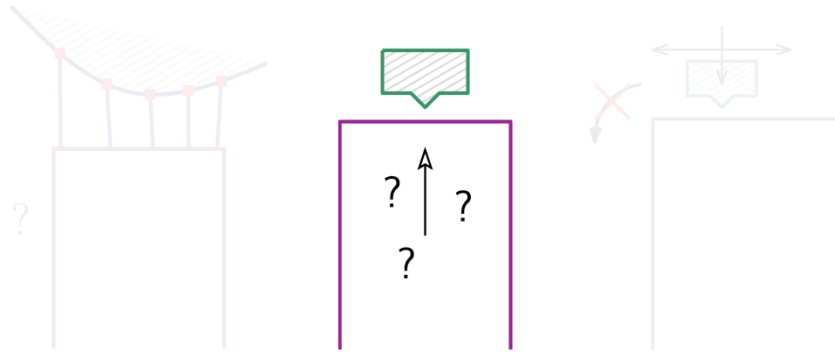
Lefebvre & Claux, 2015

7

Instead, we propose to use bridge structures. They also provide a good support ratio, but are much less sensitive to torque.

Think With Bridges

✓ Support/length ratio ✓ Resistant to torque

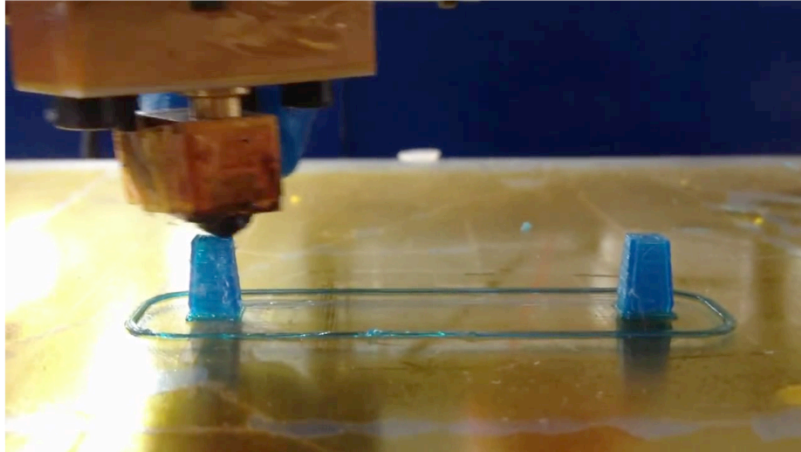


Lefebvre & Claux, 2015

8

It seems, however, counter intuitive to print bridges since they are themselves in overhang and not supported.

Printing Bridges



Source: <http://youtu.be/wK2APNwEoSsk>

Lefebvre & Claux, 2015

9

As we have discussed earlier in the course, this is actually a special case where support is not required, as illustrated here.

Torque



Lefebvre & Claux, 2015



10

The video on the left shows the behavior of the tree. Notice the large oscillating motions. On the right the equivalent bridge structure. The structure is much more stable. The same quantity of plastic is used in both, and they support the same area.

Material usage

✓ Competitive to trees



Lefebvre & Claux, 2015

11

An important consideration is to understand when each structure is at an advantage. At low heights, the bridge structures are actually smaller than the trees supporting a same area.

Material usage

✓ Competitive to
trees
x

$\|\cdot\| = 120\text{mm}$

Lefebvre & Claux, 2015

...Up to a certain
height

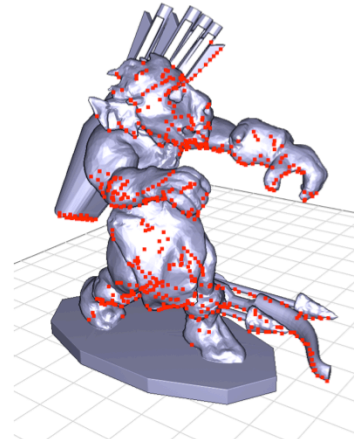
$\|\cdot\| = 121.7\text{mm}$

12

However, as the structures become taller, the tree grows a single pillar where the bridge grows four. The tree therefore becomes more competitive again. However, at such heights it also becomes much less stable.

Method Overview

1. Overhang detection
2. Bridge synthesis



[52mm scale Goblin ([TimePortalGames](#)) / [CC BY-NC-ND 3.0](#)]

Lefebvre & Claux, 2015

13

Here is the global overview of our method. We first detect the points that required support. Since we already talked about this earlier, we won't go into details here. We will focus on the second part, the bridge structure generation.

Attribution:

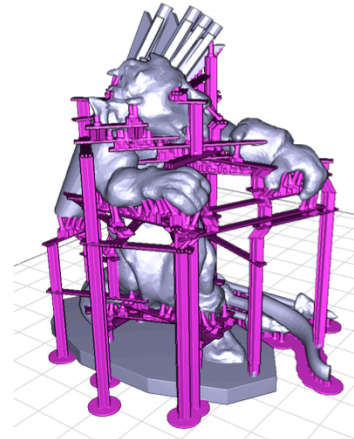
<http://www.thingiverse.com/thing:347046>

<http://www.thingiverse.com/TimePortalGames/about>

<http://creativecommons.org/licenses/by-nc-nd/3.0/>

Method Overview

1. Overhang detection
2. Bridge synthesis



[52mm scale Goblin ([TimePortalGames](http://www.thingiverse.com/thing:347046)) / [CC BY-NC-ND 3.0](http://creativecommons.org/licenses/by-nc-nd/3.0/)]

Lefebvre & Claux, 2015

14

We will focus on the second part, the bridge structure generation.

Attribution:

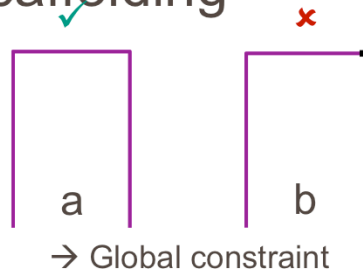
<http://www.thingiverse.com/thing:347046>

<http://www.thingiverse.com/TimePortalGames/about>

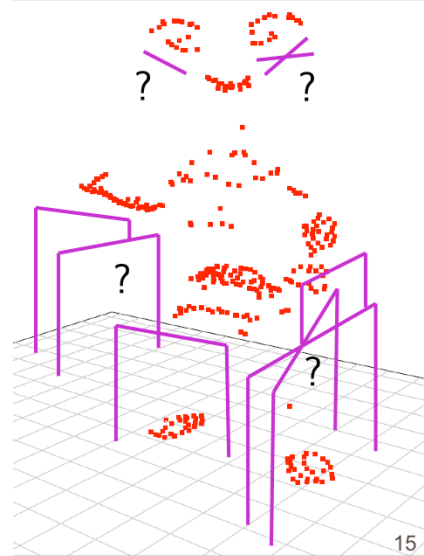
<http://creativecommons.org/licenses/by-nc-nd/3.0/>

Scaffolding: Problem Statement

- In - Required Points
- Out - Valid Scaffolding

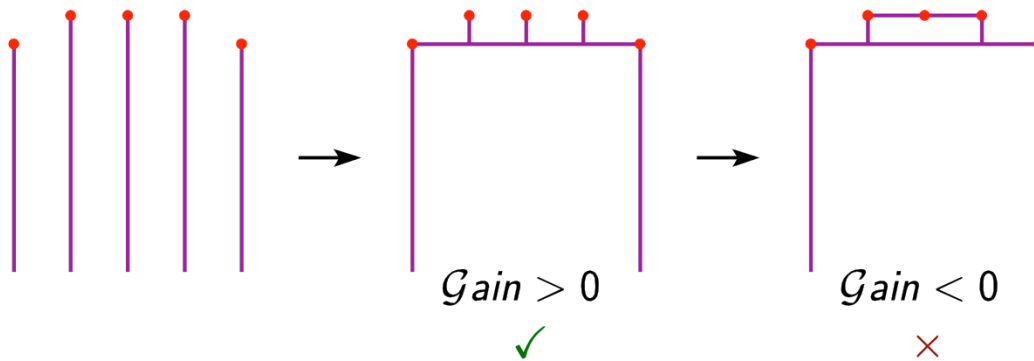


Lefebvre & Claux, 2015



As input we are given a number of points that require support. Our goal is to generate a structure which is valid – that is where every point is supported by a pillar or a bridge, and where every bridge extremity is supported by a pillar. The pillars are supported by other bridges, by the ground, or by the object itself.

Goal: *Minimal Length* Scaffolding



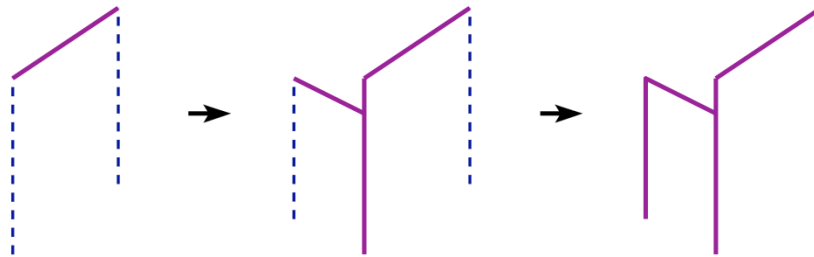
Lefebvre & Claux, 2015

16

We also seek to minimize the length of the structure. Intuitively we want to snap pillars onto bridges, to reduce the number of pillars touching the ground. However, this is not always beneficial as shown on this last case. The cost function takes into account the resulting length of the structure.

Our Approach

- Finding a **global optimum** is not easy
- Heuristic Greedy Algorithm



Lefebvre & Claux, 2015

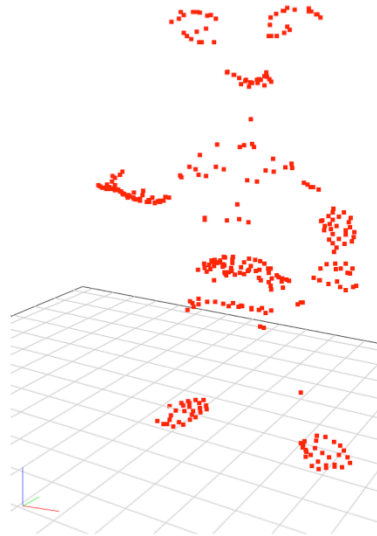
17

The problem resembles minimum rectilinear Steiner trees, but has more constraints. While the Steiner tree problem is known to be NP-hard, we have no proof yet that our problem is NP-hard as well. For now we conjecture that this is the case and rely on a heuristic algorithm.

If someone in the audience is interested by looking at a proof, we'd be happy to discuss this further!

Algorithm Overview

1. Required Points



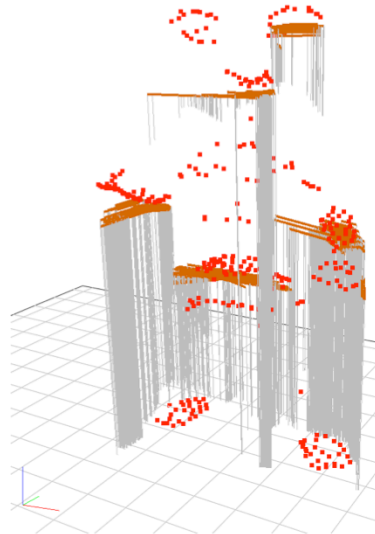
Lefebvre & Claux, 2015

18

Our algorithm iteratively improves the bridge structure. It starts from the set of points to be supported, shown here in red.

Algorithm Overview

2. Candidate Bridges



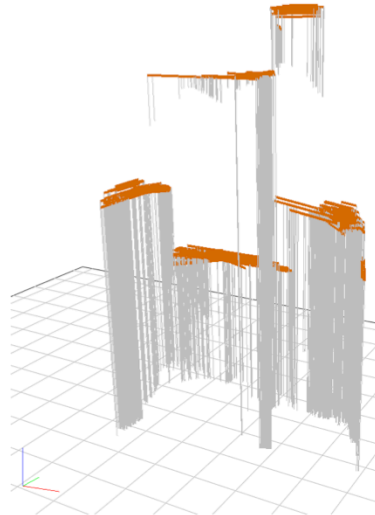
Lefebvre & Claux, 2015

19

We then efficiently produce a set of candidate bridges. I will explain how in a few slides.

Algorithm Overview

2. Candidate Bridges



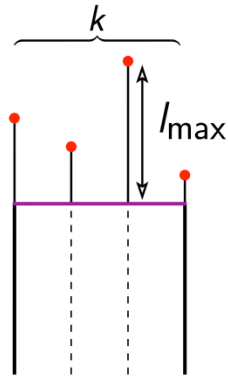
Lefebvre & Claux, 2015

20

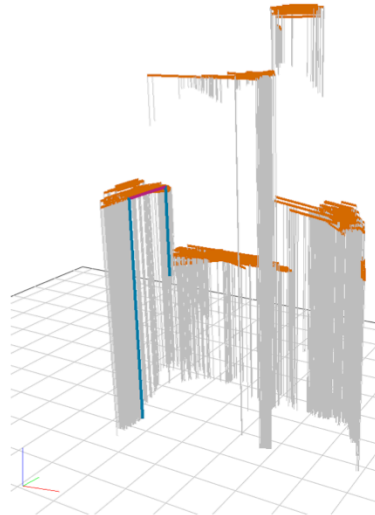
Within this set we identify the bridge giving the best improvement at this step.

Algorithm Overview

3. $\text{Score} = \text{Gain} - k \cdot l_{\max}$



Lefebvre & Claux, 2015



21

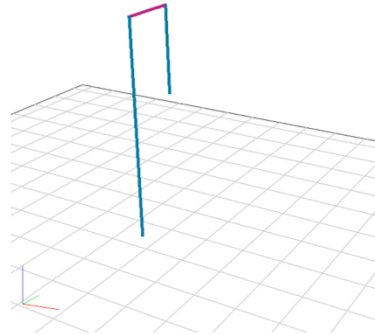
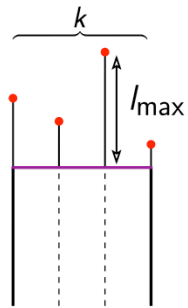
For this, we use a gain function defined as follows. We consider the number of supported structures above, and estimate the bridge pillar lengths by assuming they will go down all the way to the ground.

This is an approximation since they may later be snapped to other bridges, but this is a worst case assumption.

Algorithm Overview

$$\text{Score} = \text{Gain} - k \cdot l_{\max}$$

4. Greedy Selection



Lefebvre & Claux, 2015

22

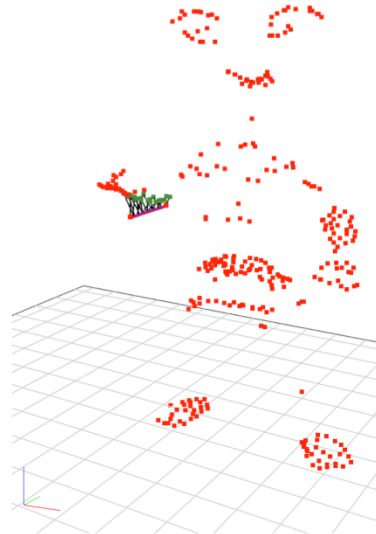
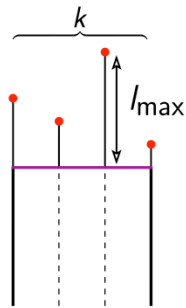
We select the bridge giving the best gain, that is the largest reduction in structure length.

If no such bridge exist, the structure is final.

Algorithm Overview

$$\text{Score} = \text{Gain} - k \cdot l_{\max}$$

5. Repeat

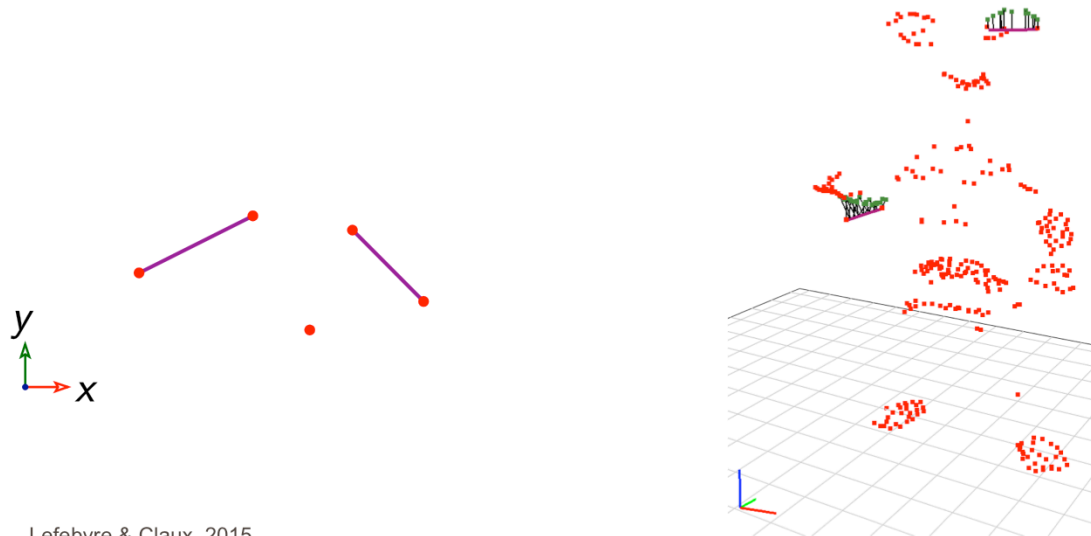


Lefebvre & Claux, 2015

23

Once the bridge is selected, we remove all supported points from the problem and insert the bridge extremities as points requiring support. This gives us back the same problem as before, but with less points to support. We iterate the algorithm until no improving bridge exist.

Candidate Bridges Generation

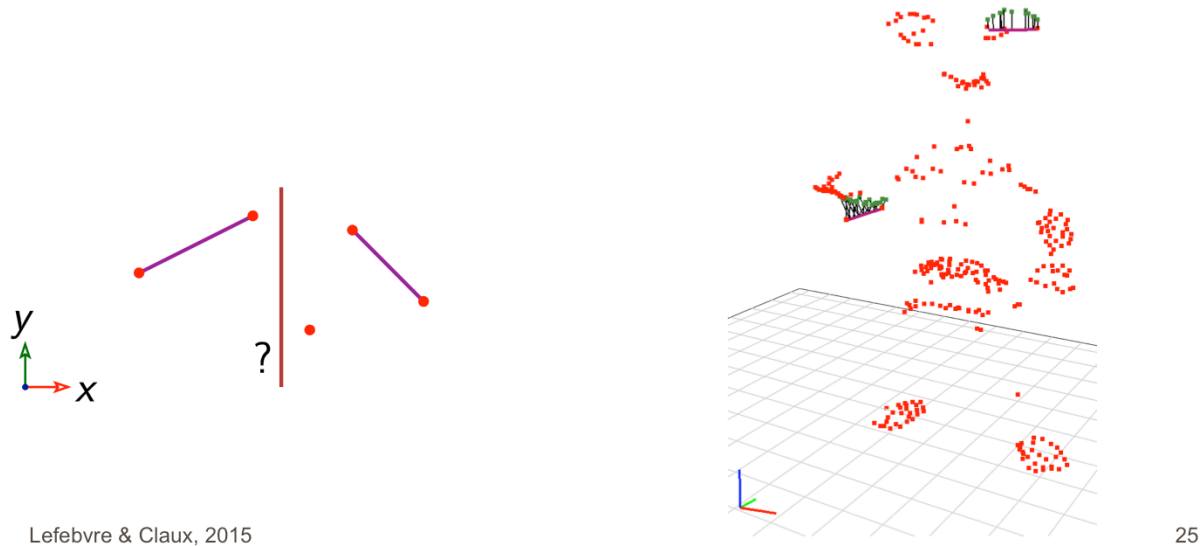


Lefebvre & Claux, 2015

24

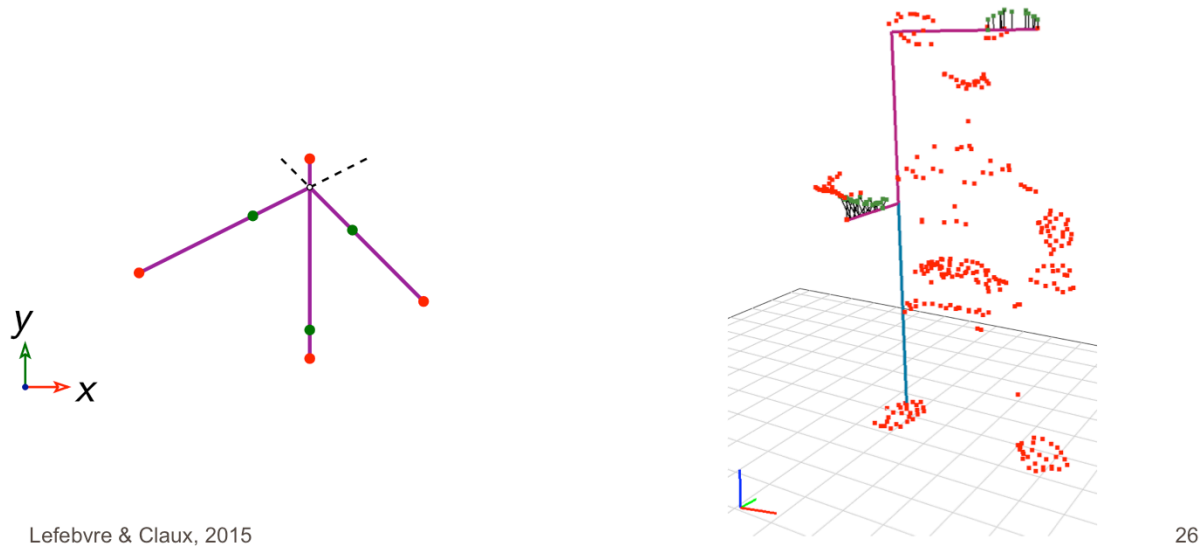
I will now describe how we produce the candidate bridges. Since this step is done often it has to be fast.

Candidate Bridges Generation



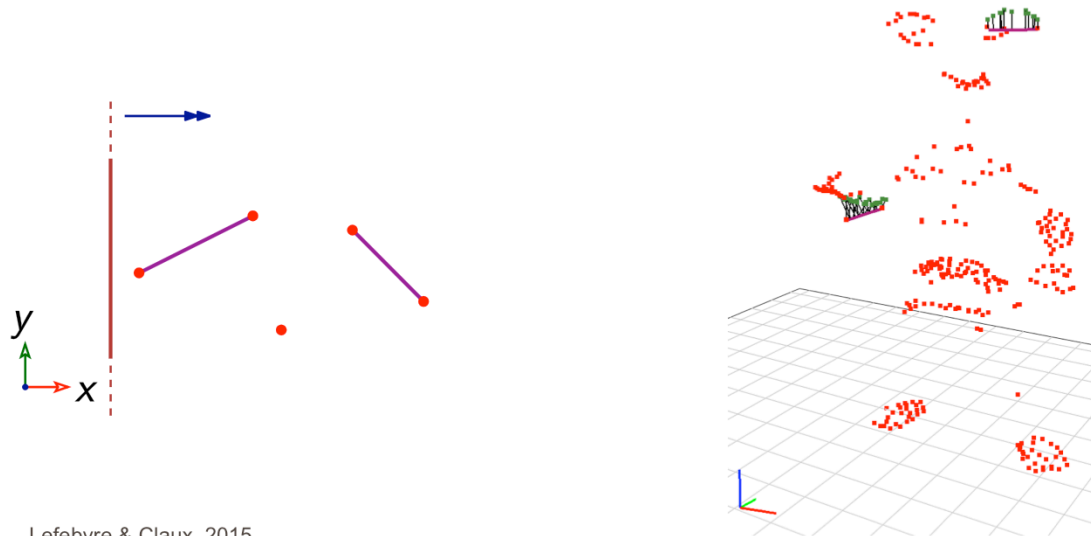
For now, let us consider a single orientation, here we search for vertical bridges. The left shows a 2D view as seen from above, while the right shows the 3D view. The two purple segments are already inserted bridges, the red dots are points to support.

Candidate Bridges Generation



In this case a good possible bridge is a bridge that would share a pillar with the two existing ones. This requires extending the existing bridges.

Candidate Bridges Generation

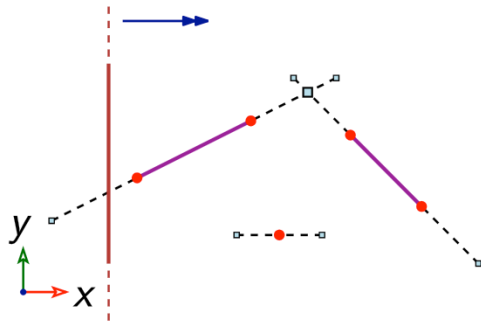


Lefebvre & Claux, 2015

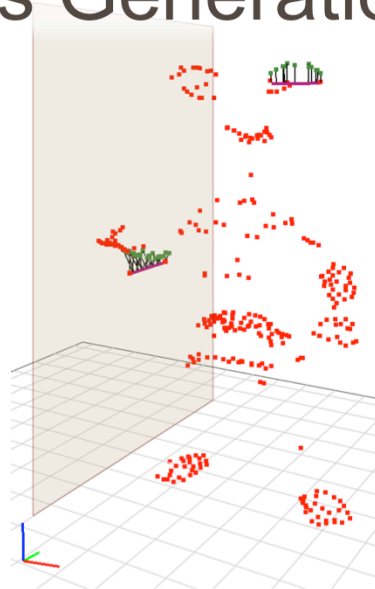
27

To detect such bridges, we sweep a plane from left to right, as illustrated by the vertical line on the left.

Candidate Bridges Generation



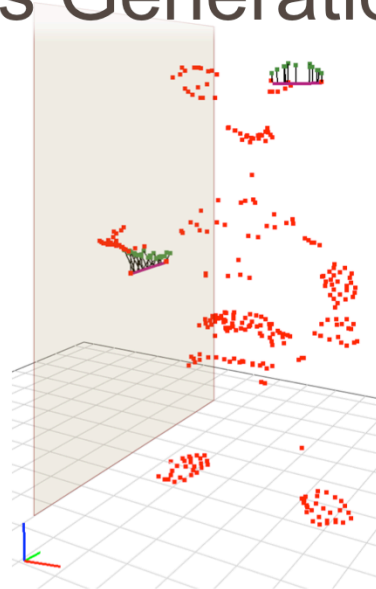
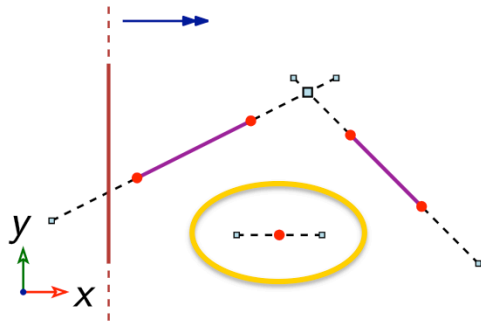
Lefebvre & Claux, 2015



28

We sweep the plane and stop in a number of geometric events. The events are all the points to be supported, the bridge extremities as well as the extremity of segments extending the bridges. These allow bridges to grow further. We consider all intersections between segments as events.

Candidate Bridges Generation

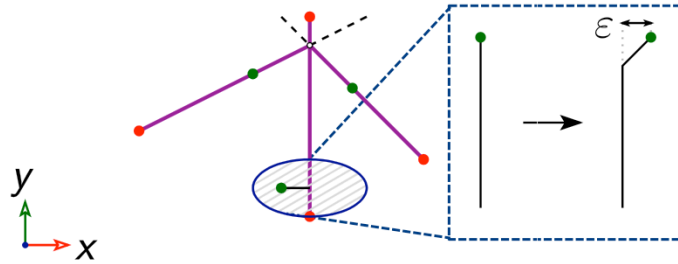


Lefebvre & Claux, 2015

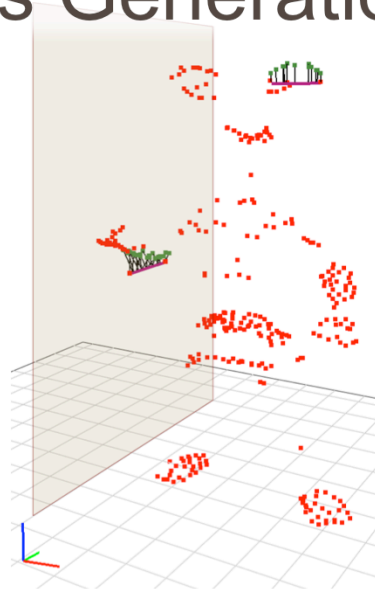
29

Segments are also added around points, so that they are allowed to be snapped to bridges by short angled pillars.

Candidate Bridges Generation



Lefebvre & Claux, 2015

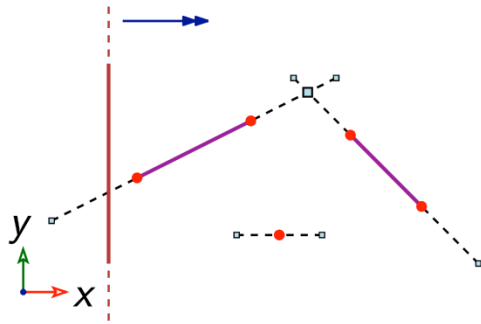


30

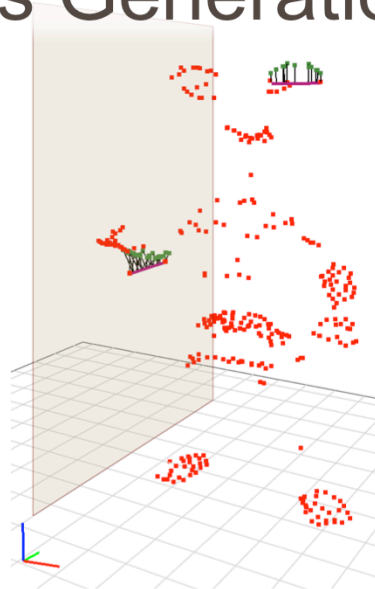
Here is an illustration of snapping a point to a nearby bridge by an angled pole. This is important, as there is otherwise very little chance that multiple points would align with bridges.

We limit the length of the angled pole to avoid instabilities.

Candidate Bridges Generation



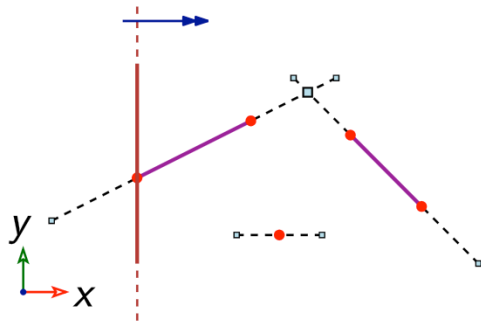
Lefebvre & Claux, 2015



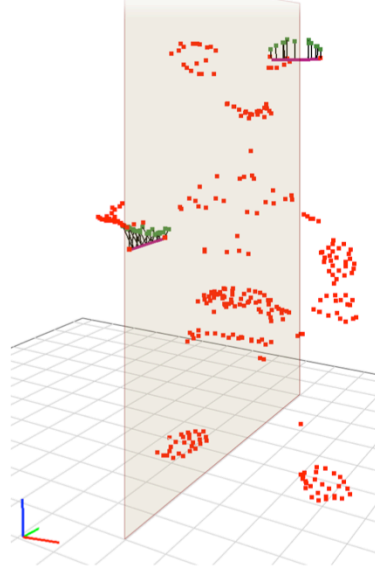
31

We sweep the plane through events.

Candidate Bridges Generation



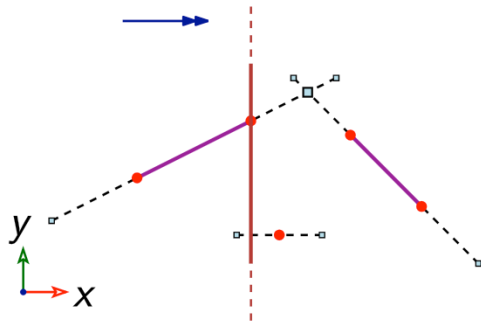
Lefebvre & Claux, 2015



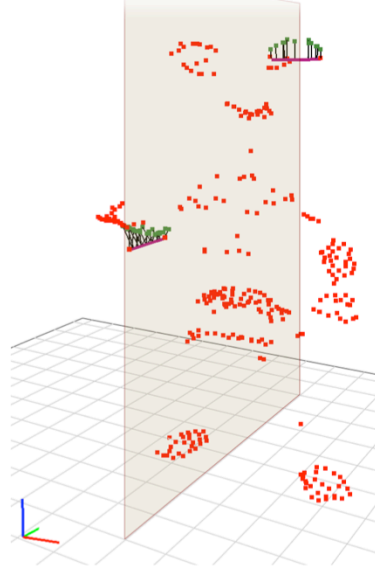
32

Here is the first considered event. No interesting bridge can be produced.

Candidate Bridges Generation



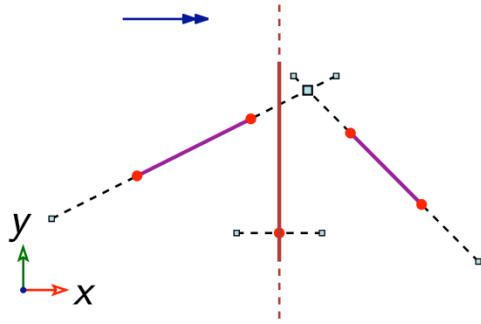
Lefebvre & Claux, 2015



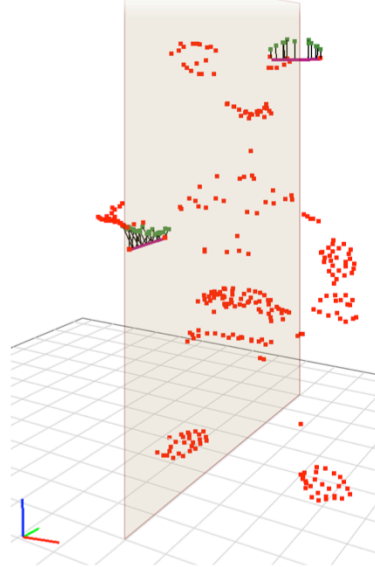
33

On this second event, a bridge candidate is added, connecting the bottom point to the top bridge extremity.

Candidate Bridges Generation



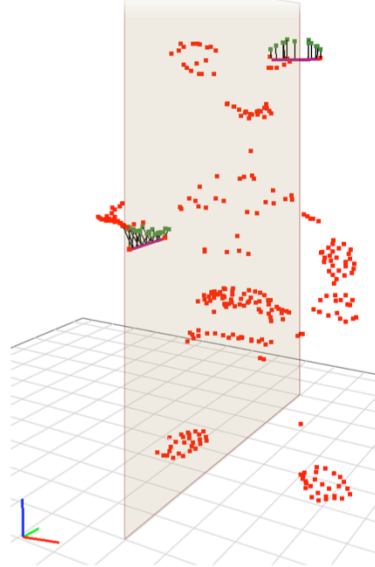
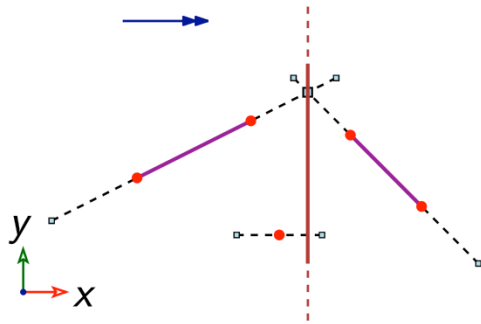
Lefebvre & Claux, 2015



34

This third event also produces a bridge candidate.

Candidate Bridges Generation

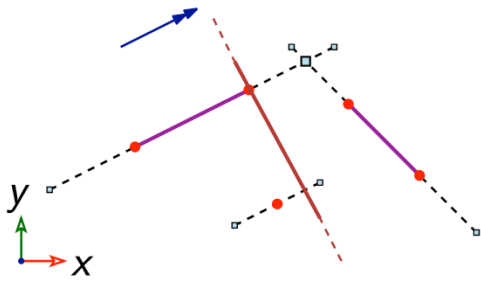


Lefebvre & Claux, 2015

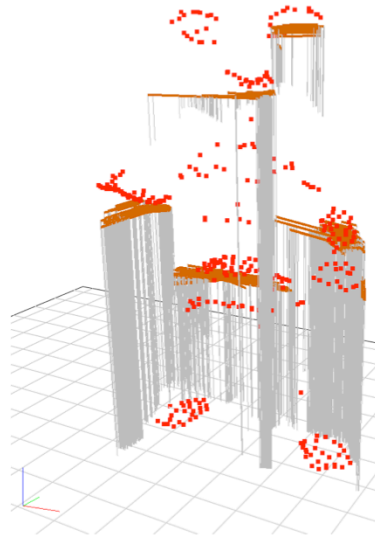
35

This fourth event is particularly interesting. It proposes a candidate bridge that snaps together the two existing bridges, as well as the bottom point. It is likely to be a very good candidate.

Candidate Bridges Generation



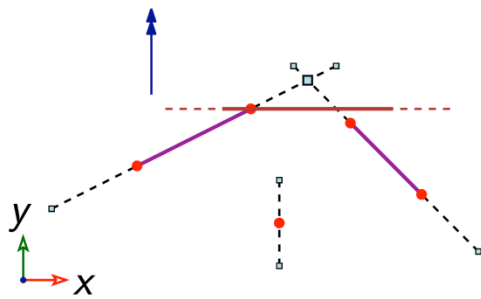
Lefebvre & Claux, 2015



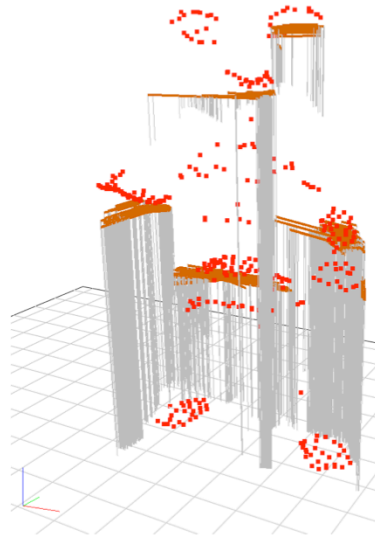
36

The sweep is performed in several directions.

Candidate Bridges Generation



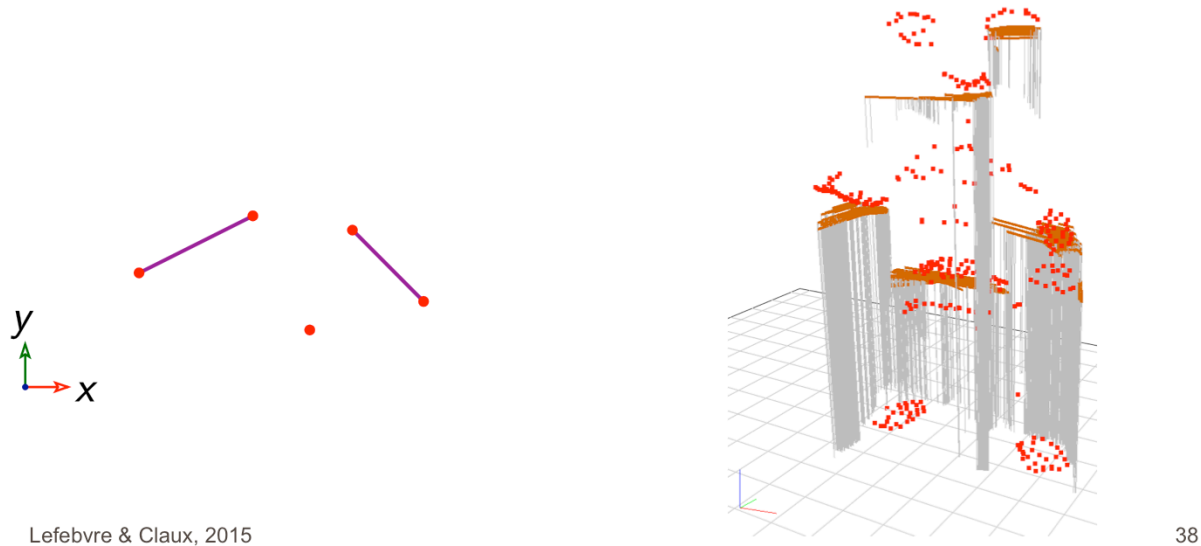
Lefebvre & Claux, 2015



37

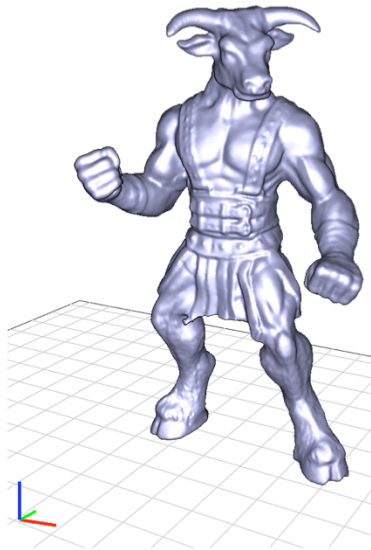
We use 8 directions in our implementation, that are explored in parallel.

Candidate Bridges Generation



This provides the complete set of candidates.

Algorithm Overview

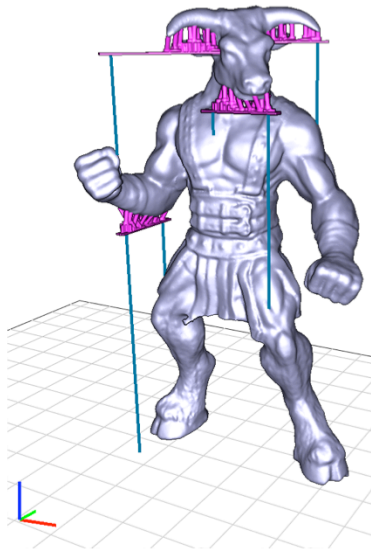


Lefebvre & Claux, 2015

39

Let's see how the algorithm iteratively builds the scaffolding for the minotaur model.

Algorithm Overview

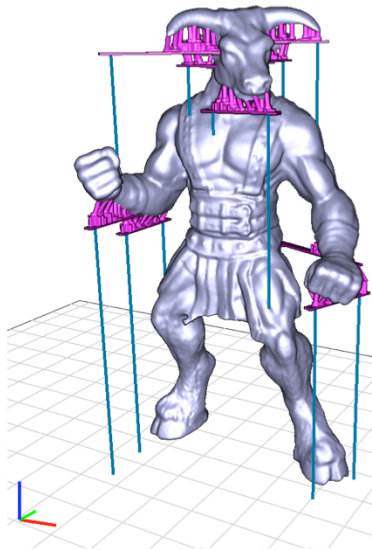


Lefebvre & Claux, 2015

40

The blue lines show the temporary pillars used for the gain computation.

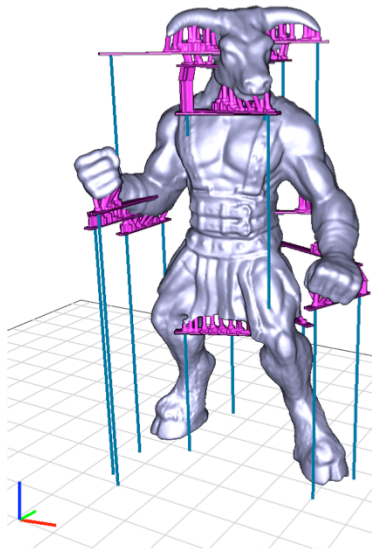
Algorithm Overview



Lefebvre & Claux, 2015

41

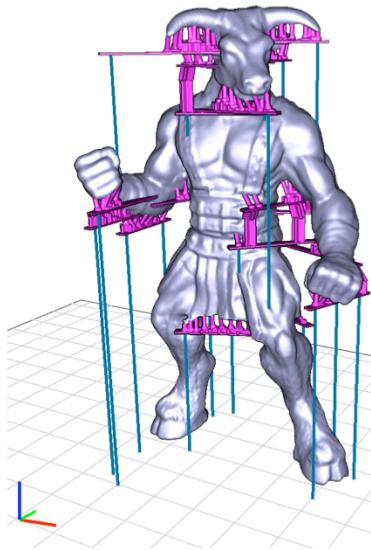
Algorithm Overview



Lefebvre & Claux, 2015

42

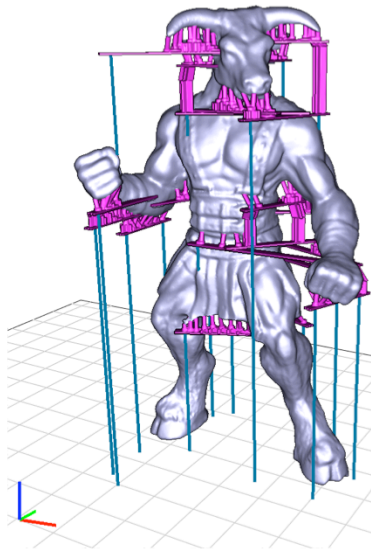
Algorithm Overview



Lefebvre & Claux, 2015

43

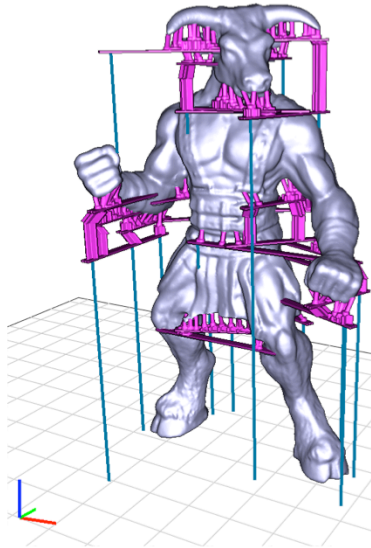
Algorithm Overview



Lefebvre & Claux, 2015

44

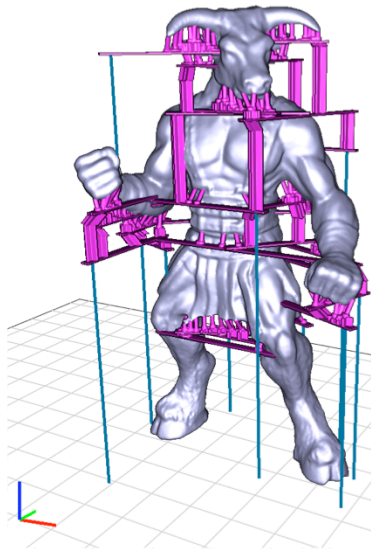
Algorithm Overview



Lefebvre & Claux, 2015

45

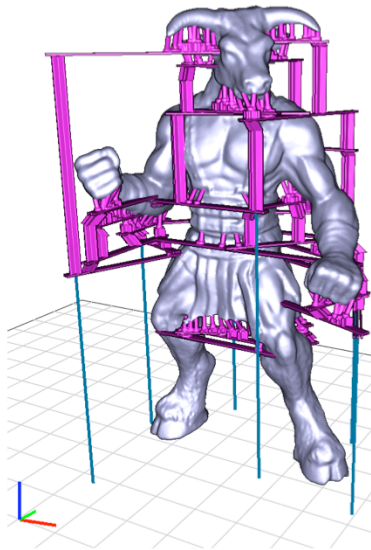
Algorithm Overview



Lefebvre & Claux, 2015

46

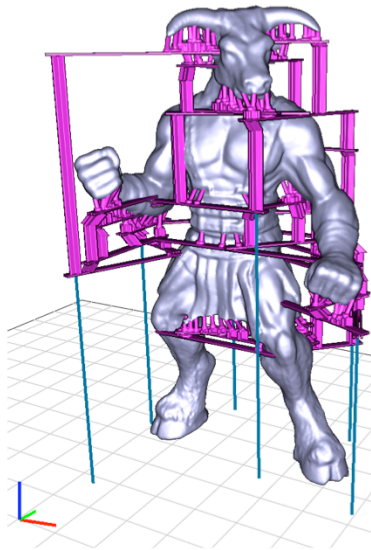
Algorithm Overview



Lefebvre & Claux, 2015

47

Algorithm Overview

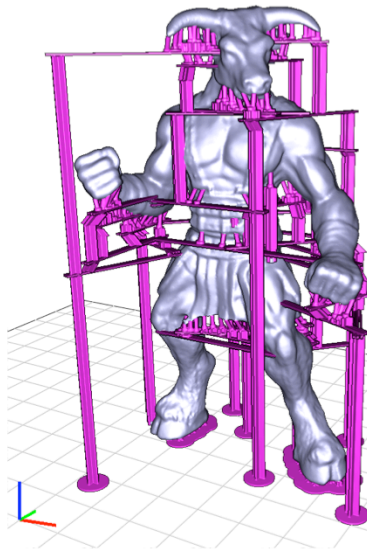


Lefebvre & Claux, 2015

48

This is the final step. No improving bridge exist, so the algorithm will convert all temporary pillars into final pillars.

Algorithm Overview



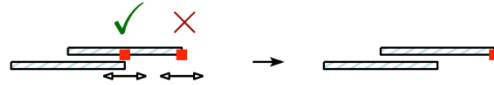
Lefebvre & Claux, 2015

49

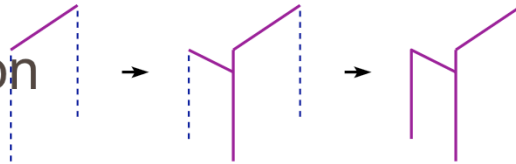
This is the end result.

The Road So Far

- Overhang Detection



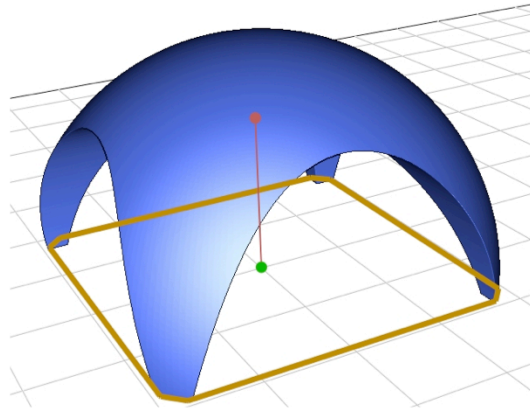
- Scaffolding Generation



- Stabilization

So far we have seen how to generate the scaffolding from the overhang points. Our bridge structures have an additional advantage: they can be used to stabilize a part during printing.

Stability During Printing

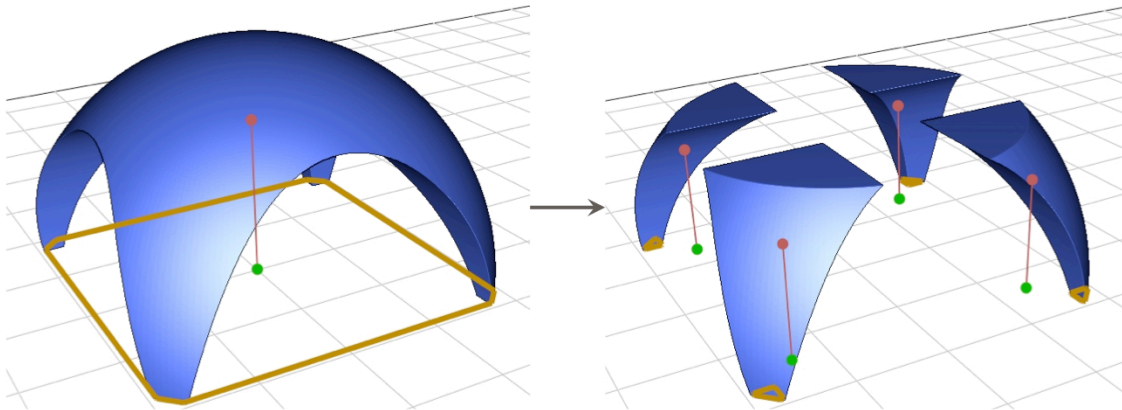


Lefebvre & Claux, 2015

51

Here is a toy example. After printing it is stable.

Stability During Printing

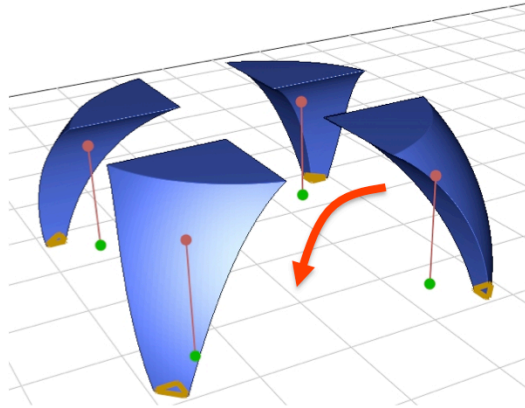


Lefebvre & Claux, 2015

52

During printing however, the four sides will be in precarious equilibrium. The orange polygon at the bottom is the base of support. On the right we can see that the center of masses project outside, which indicates that the four sub-parts are not in static equilibrium.

Stability During Printing



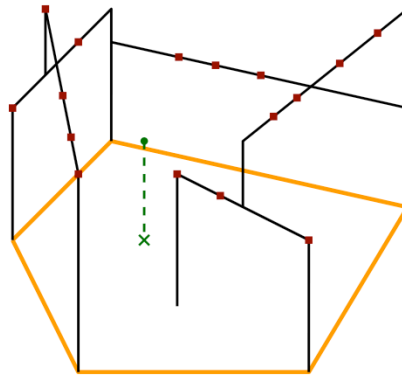
Lefebvre & Claux, 2015

53

These parts might topple under the forces exerted by the extruder.

Stability Property (Support)

CoM inside BoS \rightarrow Static equilibrium guaranteed

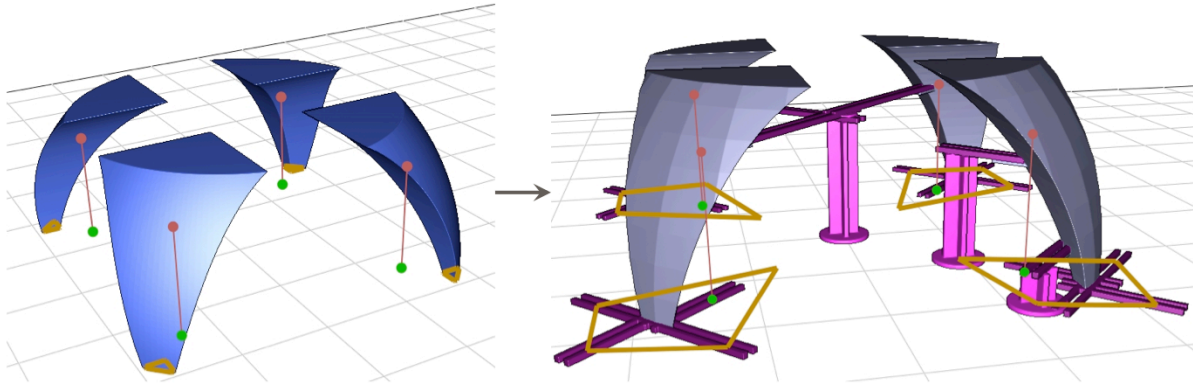


Lefebvre & Claux, 2015

54

Our bridges have an important property: They are always at equilibrium, and any point supported by them projects within the base of support of the bridges. (Only exception are the angled poles, but these remain small).

Stability Enhancement



Lefebvre & Claux, 2015

55

We therefore use our technique to add artificial support points that are then stabilized by our scaffolding. The base of support is artificially extended by introducing additional support points.

This is efficiently computed by sweeping a plane from top to bottom, tracking for each connected component its center of mass and base of support.

[we will detail this part further]

Results

Lefebvre & Claux, 2015

Results – Minotaur



Lefebvre & Claux, 2015



[Minotaur ([ajolivette](#)) / [CC BY-SA 3.0](#)]



10 cm
57

Here is the minotaur model, and its support structure. We also have actual printouts available for closer inspection.

Attribution:

<http://www.thingiverse.com/thing:46646>

<http://www.thingiverse.com/ajolivette/about>

<http://creativecommons.org/licenses/by-sa/3.0/>



This is a leg of a 3D printed robot. It is important to print it horizontally due to mechanical properties, and our bridge structure allows for that.

Results – Hilbert Cube



Lefebvre & Claux, 2015



[Hilbert Cube ([tbuser](#)) / [CC BY-SA 3.0](#)]

59

On this model we support the points inside without leaning too much on the object.

Attribution:

<http://www.thingiverse.com/thing:16343>

<http://www.thingiverse.com/tbuser/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

Limitations

- Sagging (see paper)



- Surface quality



[Star Trek 2009 Enterprise ([JackSpectre](#)) / [CC BY-SA 3.0](#)]

- Computation time

Lefebvre & Claux, 2015

60

There are some limitations. The bridges can sag during printing, but this never lead to print failure in all our tests.

The bottom surface quality obviously suffer, an issue shared amongst all techniques. Of course soluble plastic could be used for the structure, resulting in less damage.

The computation time is quite high, especially when the number of points to be supported largely increases. We currently investigate ways to group the points before generating the scaffoldings.

Attribution:

<http://www.thingiverse.com/thing:18346>

<http://www.thingiverse.com/JackSpectre/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

Print time

- In principle less plastic = faster
- Travel might increase, but travel is fast
- Main issue: Acceleration on small connectors



Lefebvre & Claux, 2015

61

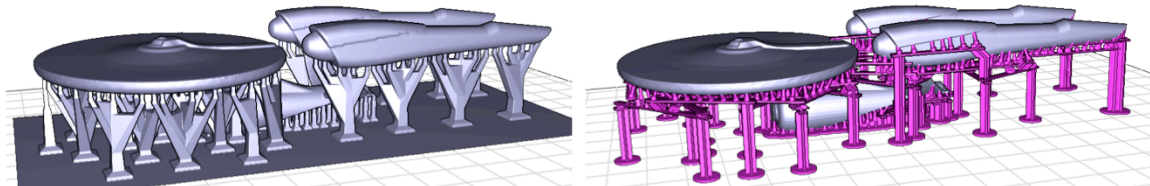
I would like to conclude with one issue that surprised us at first. Even though our structures are much smaller, the gain in print time is not always as significant. One might think that this is due to increased travel time, but a closer analysis reveals that travel time does not significantly grow.

The main problem stems from the slow acceleration of the print head. When printing small details, such as the connectors, the carriage never reaches the top speed.

We therefore investigate ways to print better connectors that print faster, without having to slow down the printer. This is an interesting aspect of print time optimization.

Open question

Hybrid approach (trees + bridges)



[Star Trek 2009 Enterprise ([JackSpectre](#)) / [CC BY-SA 3.0](#)]

Lefebvre & Claux, 2015

62

Another interesting direction of future work is to consider hybrid approaches, using trees and bridges. However, to produce reliable trees we believe a good mechanical model is crucial. Because these structures print at the minimal sizes and at high speeds, they are usually not very well printed, and cannot be considered as homogenous isotropic materials, which complicates simulation.

Attribution:

<http://www.thingiverse.com/thing:18346>

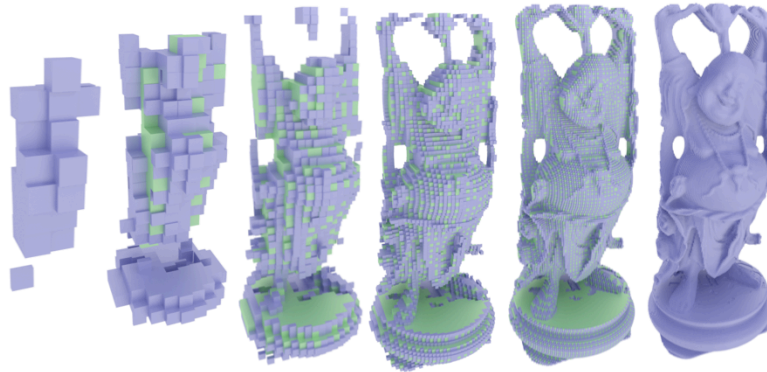
<http://www.thingiverse.com/JackSpectre/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

Acknowledgements

- Artists sharing their models
- ERC Shapeforge (StG-2012-307877)
- Website and binary release in preparation
<http://shapeforge.loria.fr/icesl/>

We'd like to thank artists and funding agencies.



INTEGRATING SOLID MODELING WITH SLICING

Modeling for Filament-Based 3D Printing

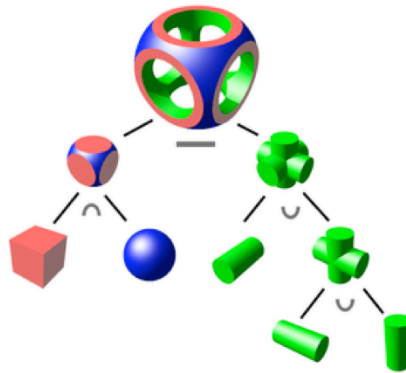
IceSL: GPU accelerated CSG modeler and slicer

Sylvain Lefebvre
INRIA Nancy - LORIA

We present now the modeling and slicing engine behind IceSL.

Constructive Solid Geometry


- Modeling operations between *solids*






http://en.wikipedia.org/wiki/Constructive_solid_geometry

Lefebvre & Claux, 2015

One popular way of modeling objects is to use Constructive Solid Geometry, or CSG. CSG assembles solids using union, intersection and difference operators into a tree. You can make complex solids with complex trees. It is relatively intuitive, even for beginners. For instance, the difference operator is handy to remove matter from an existing object, such as when holes need to be created off, say, a solid bar. Primitives can be of any kind, including complex meshes.



Doggie new joints
by sylefeb, published Oct 20, 2012

```

module half_dog()
{
  difference()
  {
    dog();
    scale(Mult)  translate([0,-40,-5])  cube([40,100,70]);
  }
}

module flegconn()
{
  difference()
  {
    dog();
    scale(scl)  translate([-40,-40,14])  cube([80,80,70]);
  }
}

module blegconn()
{
  difference()
  {
    dog();
    scale(scl)  translate([-40,-40,2])  cube([80,80,70]);
  }
}

module dog()
{
  scale(Mult)
  {
    body();
    translate([0,-15.0,48.4])  rotate([0,180,0])  head();
    translate([-4.6,3.2,28])  rotate([30,90,180])  fleg();
    translate([4.6,3.2,28])  mirror([1,0,0])  rotate([30,90,180])  fleg();
    translate([-4.5,3,4])  rotate([30,90,180])  bleg();
    translate([4.5,3,4])  mirror([1,0,0])  rotate([30,90,180])  bleg();
  }
}

```

75

133

0

4

0

ad This Thing!

Thing Info

Instructions

Thing Files

0 Comments

4 Made

133 Collections

0 Remixes

Here is the doggie model taken from thingiverse and part its source code.

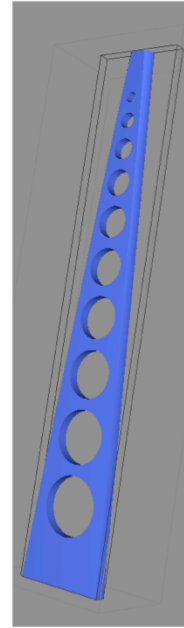
Made by a 13 years old

```
long = 100
r = scale(long,30,2) * translate(0,0,0.5) * box(1)

intersize = long/11
pos = -long/2+intersize
spacing = 2
for i=1,10 do
  c = translate(pos, 0, 0) * cylinder( (1+i)/2, 2 )
  r = difference( r, c )
  pos = pos + (i+1)/2 + spacing + (i+2)/2
end

cn = rotate(90,Y) * translate(0,0,-long/2) * cone(5/2,18/2,long)

emit( intersection(r,cn) )
```

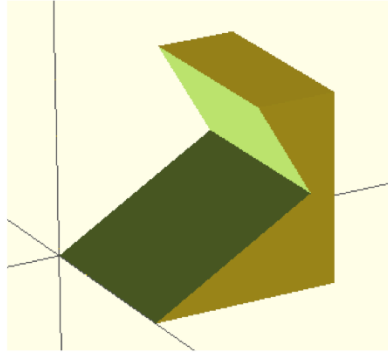


Lefebvre & Claux, 2015

Writing scripts might seem hard, but here is an example made by a 13 years old after some basic explanations.

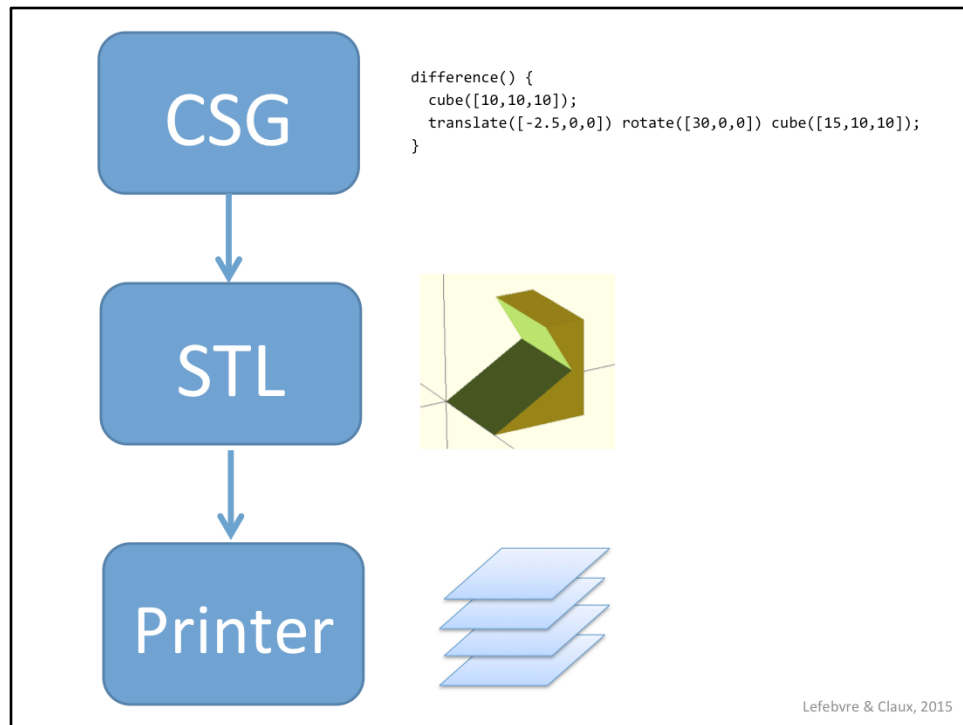
OpenSCAD

```
difference() {  
  cube([10,10,10]);  
  translate([-2.5,0,0]) rotate([30,0,0]) cube([15,10,10]);  
}
```

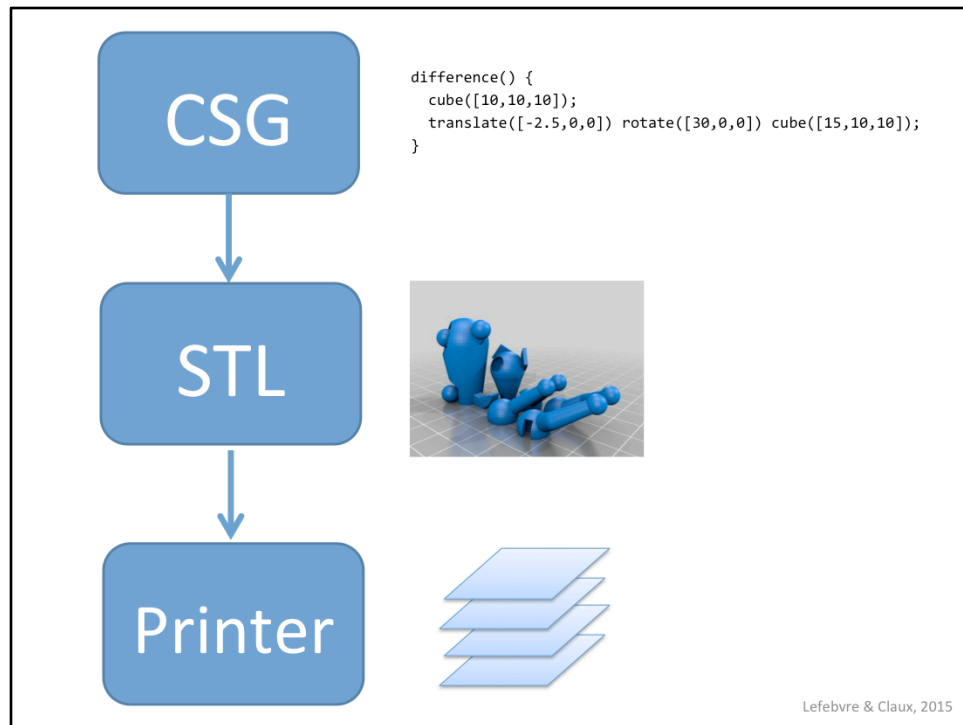


Lefebvre & Claux, 2015

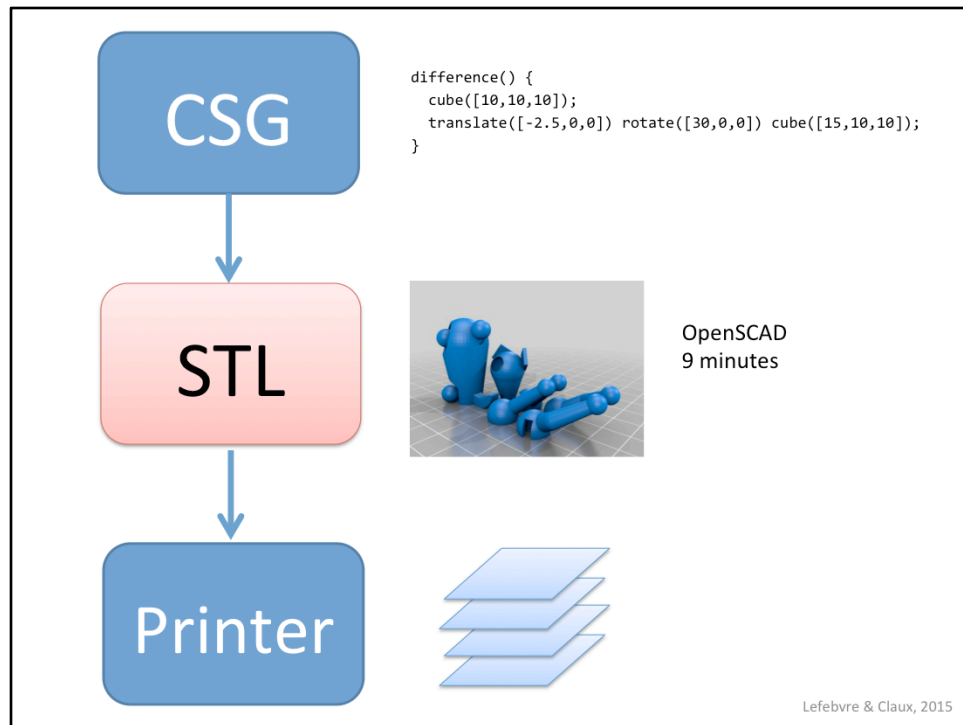
OpenSCAD for instance, uses CSG trees. This is an example model, with its example CSG tree source code.



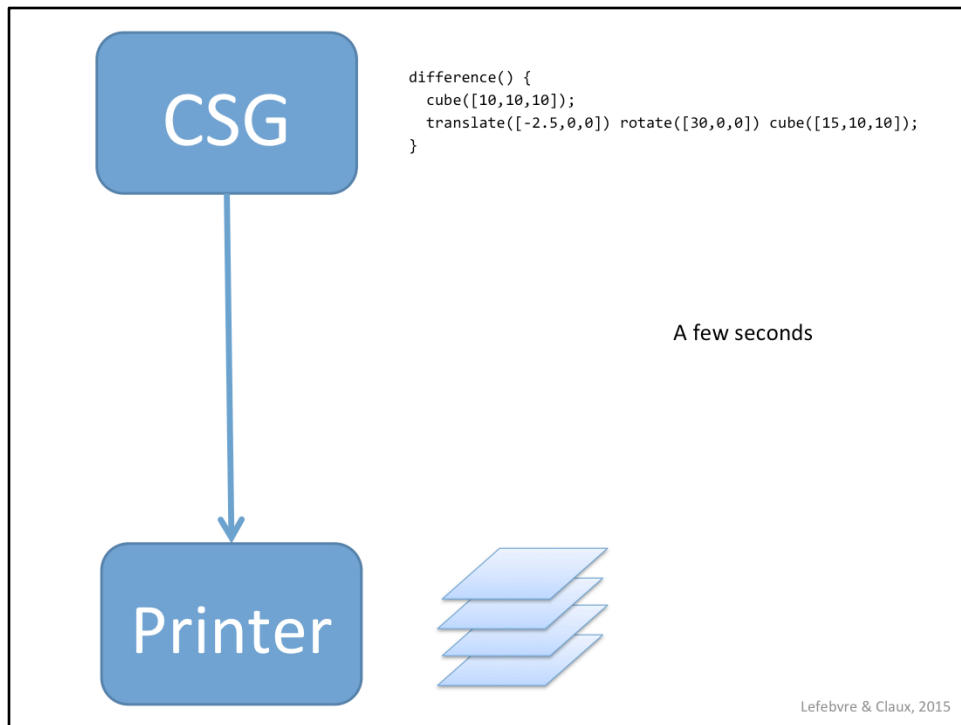
OpenSCAD, like other software applications, evaluates the CSG tree and turns it into a mesh, which it then gives to the slicer/printer.



To produce the STL mesh file, ready to be printed, OpenSCAD transforms each CSG tree primitive into a mesh and then applies the CSG operators with these primitives, from the bottom up and in a recursive fashion up the tree.



It takes 9 minutes to generate the final geometry.



It would be better to produce a printer-understandable Gcode file right from the input tree without having to generate a mesh first. This would considerably simplify the printing process, as generating the mesh is an obvious bottleneck.

Problems

- CSG with meshes is difficult (numerical issues)
- Slicers need well formed meshes
- OpenSCAD produces high quality meshes
 - Relies on CGAL
 - Very powerful, but computationally intensive
- Number of triangles quickly increases
 - Surface details, internal details, etc.

Lefebvre & Claux, 2015

CSG operations on meshes are very difficult. Many algorithms exist and a lot of them indeed have numerical issues.

OpenSCAD relies on CGAL to do the CSG operations. CGAL is reliable but is very calculation intensive.

The number of triangles can quickly grow, which tends to make calculations slower and slower as CSG operators are being evaluated throughout the tree.

IceSL

[Video]

Lefebvre & Claux, 2015

Let us show you a video of IceSL which demonstrates how the CSG can be evaluated in real-time, for rendering and interactive modeling purposes, using a different approach.

Comparison



- CSG + slicing
 - 0.25mm height, 25% infill, no shell

IceSL

CSG+slicer: 31 seconds

OpenSCAD + Kisslicer

CSG: 16 minutes
Slicer: 10 seconds

Lefebvre & Claux, 2015

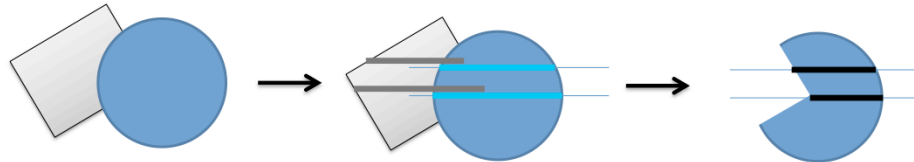
IceSL does not need to create a mesh for printing, and directly visualizes the CSG tree during interactive editing. As you see in the previous video, CSG can be performed in real-time for rendering. The current slide gives some figures about slicing performance.

Same result!

- Compute CSG *before* slicing



- Compute CSG *after* slicing



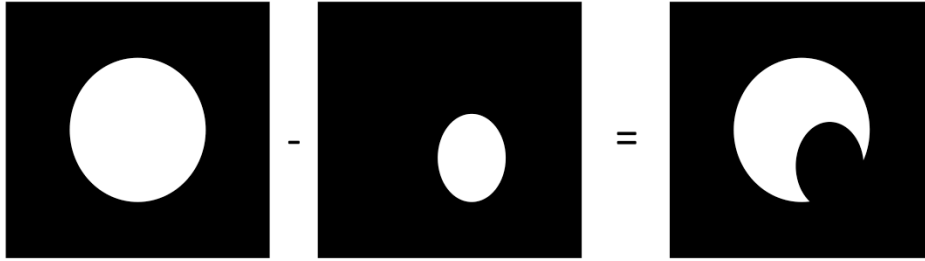
Lefebvre & Claux, 2015

Let's take a look at what makes this possible.

Slicing the result of the CSG model, or evaluating the CSG after generating object slices produces the same result.

Direct CSG slicing

- Perform CSG in the slices
 - Instead of entire model [\[RepRap Host\]](#)

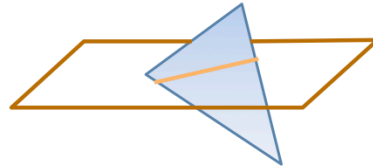


Lefebvre & Claux, 2015

Since slices are composed of 2D primitives, the CSG is considerably simpler to do within a slice with 2D primitives, than with full 3D meshes.

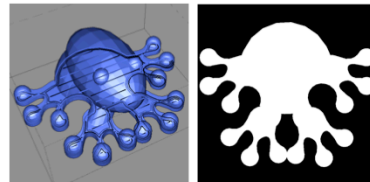
Main approaches for slicing

- Vector based:
 - Intersect slice plane – mesh: 2D polygon outline
 - Offset 2D polyline
 - Ex: *Makerbot Slicer*



- Raster based:
 - Render mesh onto slice plane as an image
 - Extract contours
 - Ex: *RepRap Host*
 - Direct slicing from raster [Zeng et al. 2011]

[Frog (owenscenic) / CC BY-NC-SA 3.0]



Lefebvre & Claux, 2015

Generating object slices can be done in two ways.

One way is to generate vector geometry from the intersection between planes and the object and then to create offsetted polylines for contours from this geometry

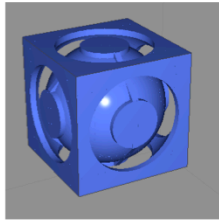
Another way is to somehow render objects « onto slice planes » and then extract the rasterized slices contours. We'll see later how we can do that.

Attribution:

<http://www.thingiverse.com/thing:3284>

<http://www.thingiverse.com/owenscenic/about>

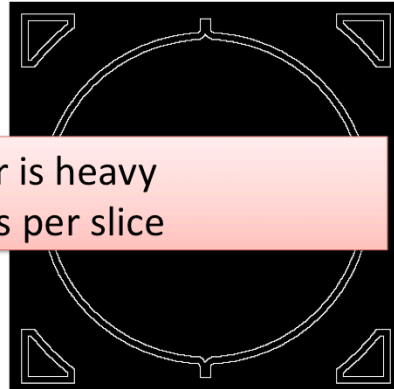
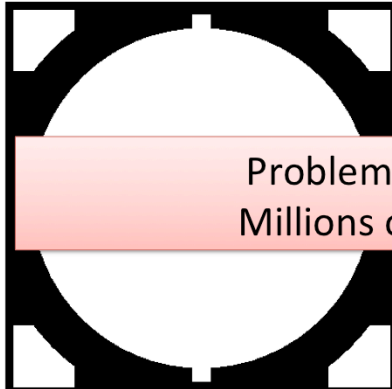
<http://creativecommons.org/licenses/by-nc-sa/3.0/>



Raster-based

Slice = image

White pixel = inside ; black = outside



Problem: Raster is heavy
Millions of pixels per slice

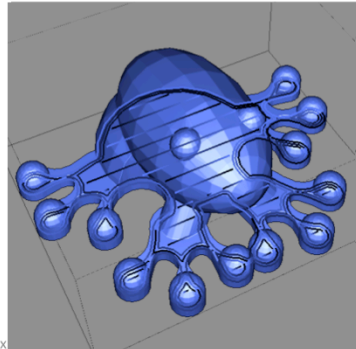
Slice #95

Lefebvre & Claux, 2015

One thing to keep in mind when we do this is that each slice requires a lot of memory, as the raster will typically be the size of the object, and have the resolution of the printer. That's a lot of pixels.

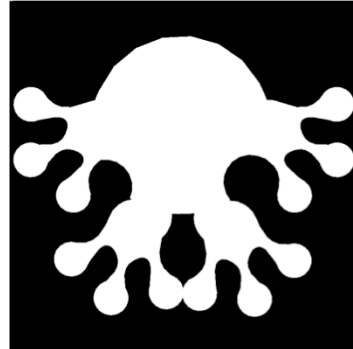
GPU

- Graphics processors are perfect for this
 - Designed for parallel processing of pixels



Lefebvre & Claus

White pixel = inside ; black = outside



GPUs are ideal for this purpose, because they are primarily meant to produce raster images from 3D objects.

IceSL

- Raster representation
 - *Only at slicing time*
 - *At printing resolution*
 - *Transient, disposable. Used as input for tool path generation.*
- Geometry always under its original form
 - No meshes !

Lefebvre & Claux, 2015

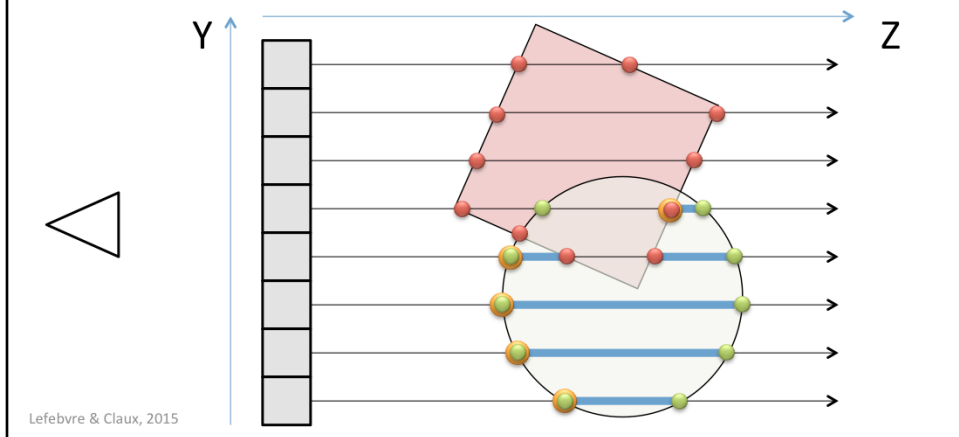
When you talk raster-bases slicing, a raster representation is only generated at slicing time, for each slice, at printing resolution. The generated rasters are temporary data generated using the original model, prior to the tool path generation phase, producing the final G-Code file.

Geometry never leaves its original form.

Let's look at how raster slices are generated with the GPU.

Raster representation

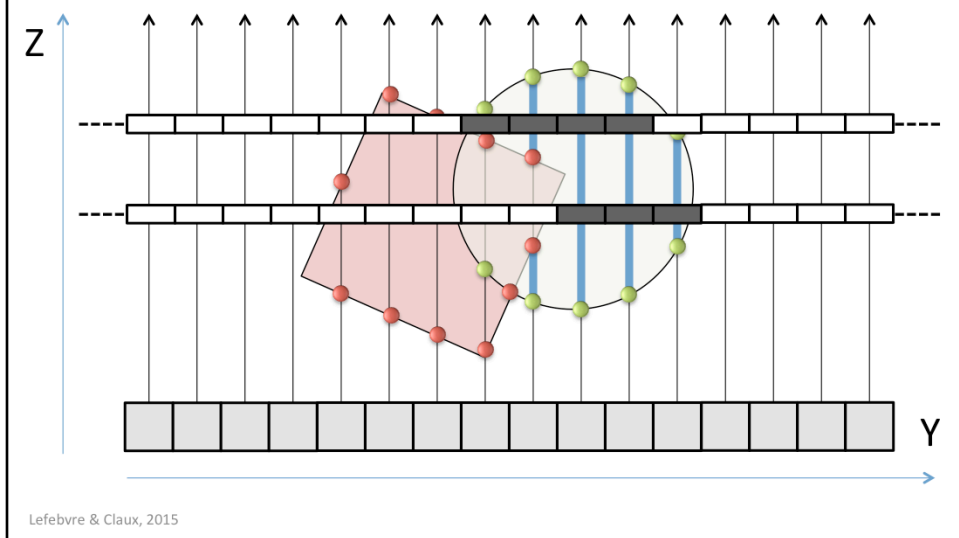
- A-buffer [Carpenter 1984]
 - Sorted list of in/out events per pixel



We first create an orthogonal transformation and set up rasterization to work in a bottom-up or top-down fashion (Z axis)

We render all the primitives into an A-Buffer, which records all rasterized fragments in depth order for each pixel

Slice extraction



To extract a slice at a given Z coordinate, we evaluate the CSG at a any given X,Y coordinate on this slice. We get the raster slice that way.

A-buffer

- Fast construction techniques
 - Per-pixel Linked Lists [Crassin 2010]
 - HA-buffer [Lefebvre et al. 2013]



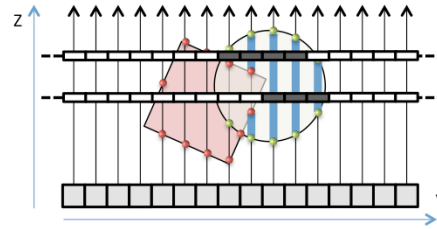
Single pass, integrated within OpenGL (simple)

Lefebvre & Claux, 2015

The construction of the A-Buffer can be fully integrated into the pipeline of the graphics API (eg. OpenGL or DirectX).
We have methods to do lock-free sorted insertions for the fragments.

Algorithm for slicing

- Setup view from bed
- Build A-buffer
- For each slice
 - Extract image
 - Build tool path (see paper)

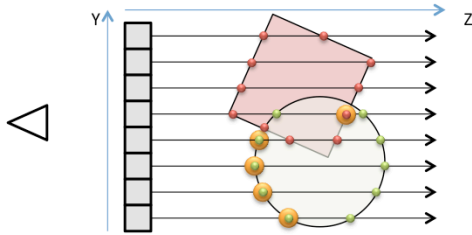


Lefebvre & Claux, 2015

Slicing is just a matter of building an A-Buffer once for all, and then to extract slices at various levels out of this A-Buffer. The actual tools paths can then be generated for each raster image.

Algorithm for rendering

- Setup view from eye
- Build A-buffer
- For each pixel
 - Find first intersection



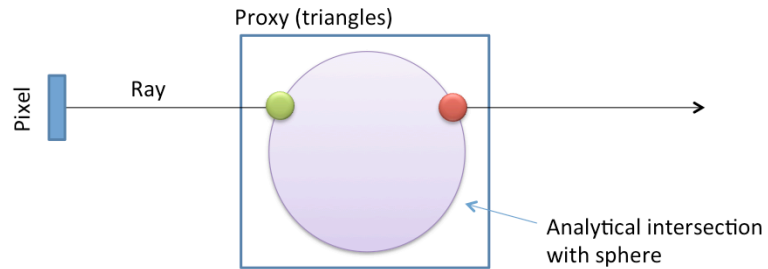
Same algorithm for slicing and rendering

Lefebvre & Claux, 2015

For interactive rendering, an A-Buffer is generated for each view. The CSG is evaluated in a front-to-back fashion and stops at the frontmost visible fragment at each pixel location, which is then shaded and rendered.

Other benefits

- Analytical primitives
 - If you can draw it, we can slice it

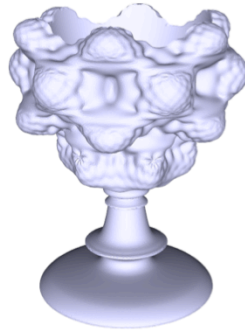


Lefebvre & Claux, 2015

Any kind of solid primitive becomes printable using this method, as long as we have rasterization code for it. Analytical primitives can be raytraced through some proxy geometry like a convex hull or just a quad primitive.

Other benefits

- Mixing primitive types is easy and reliable
- Mandelbulb-cup =
STL cup holder + (Mandelbulb implicit – sphere)

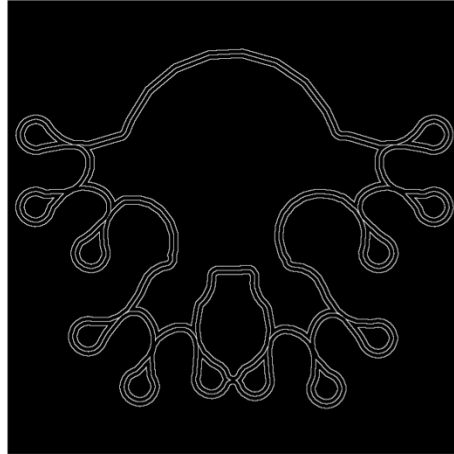


Lefebvre & Claux, 2015

Mixing primitive types is straightforward and CSG can work seamlessly with them. Here is an example of a complex cup generated with a mesh and an implicit primitive.

Other benefits

- Image-based path extraction
 - Morphological erosion



Lefebvre & Claux, 2015

There are other benefits in using raster images for slices.
Image-based morphological erosion algorithms can be used to calculate tool paths.

Modeling with the dexels

- Example: morphological operations
- Applications for 3D printing include:
 - Molds and hollowing
 - Removing small features

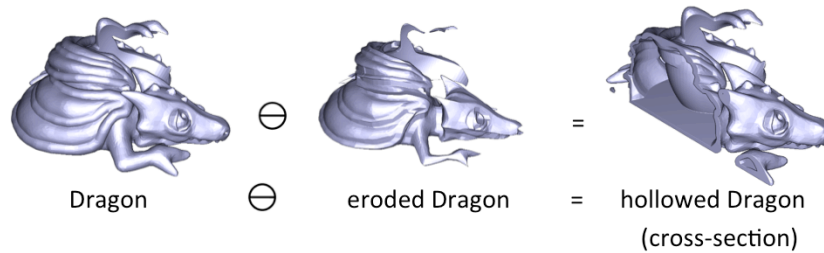
Lefebvre & Claux, 2015

It is possible to leverage the A-Buffer/doxel representation to perform interesting modeling operations, especially those of interest in the context of fabrication, for instance creating molds or hollows or removing small undesired features.

Example: hollowing

- Hollowing uses CSG with a self eroded model

[Pet monster Valentine ([andreas](#)) / [CC BY-NC-SA 3.0](#)]



- Hollowed model faster and cheaper to print
- Molds use a dilated model

Lefebvre & Claux, 2015

Let's take an example with a hollowed model as it's a good way to save on material expenditures and print time.

Hollows can be made by subtracting an eroded model from the original model.

Attribution:

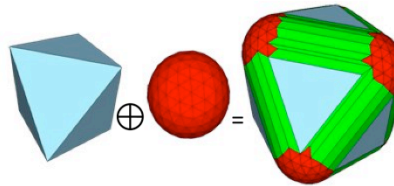
<http://www.thingiverse.com/thing:17204>

<http://www.thingiverse.com/andreas/about>

<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Morphological operations

- dilation \oplus = Minkowski sum of model with a ball



- Erosion \ominus = complement of Minkowski sum of complement model with a ball

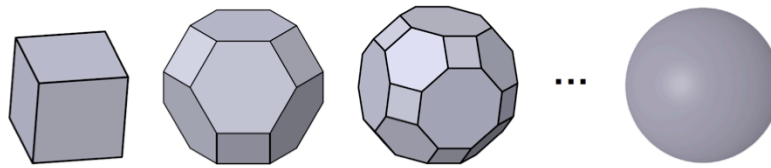
Lefebvre & Claux, 2015

A dilation between two solids is the Minkowski sum between these solids. Take all the balls at every possible surface position over the blue solid, and union them together into the blue solid. You get a dilation.

An erosion is just a matter of using complements for the input and output.

Morphological operations

- *Approximated* erosion/dilation = use *approximated* ball
- Approximation OK for hollowing or removing small features, as long as the offset is small
- Use zonotope as approximating solid
[\[Martinez et al. 2015\]](#)

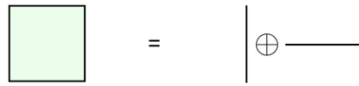


Lefebvre & Claux, 2015

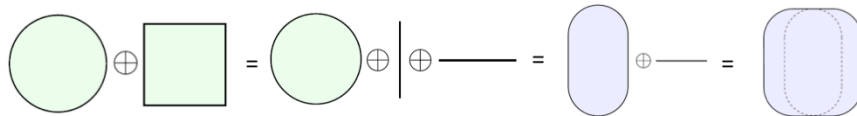
It is possible to use an approximation of a ball instead of a real ball to approximate the dilation or erosion. It's no big deal for hollowing.
A special type of solid, a zonotope, is ideal for this approximation.

Morphological operations in IceSL

- Zonotope = Minkowski sum of line segments



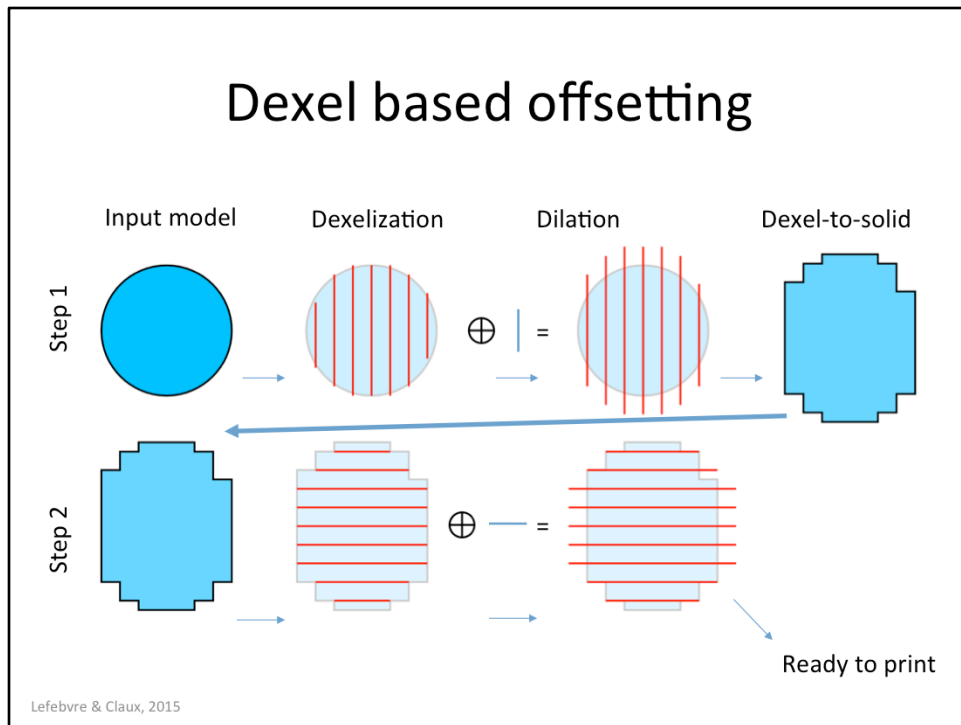
- Offsetting with zonotope = successive offsetting with these line segments



Lefebvre & Claux, 2015

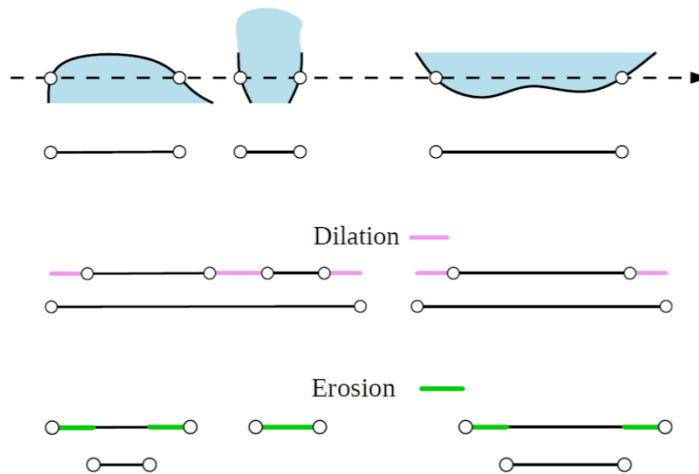
Zonotopes are themselves Minkowski sums of line segments.
 Successively offsetting a solid with a zonotope's line segments gives the Minkowski sum between this solid and the zonotope.

Dixel based offsetting



- Offsetting becomes just a matter of, for each zonotope line segment,
- 1) Generating a dixel structure using the direction of the line segment
 - 2) Dilating or eroding the dexels (see next slide)
 - 3) Transforming the resulting dexels back into a solid representation
 - 4) do this for every line segment

Dixel-based offsetting

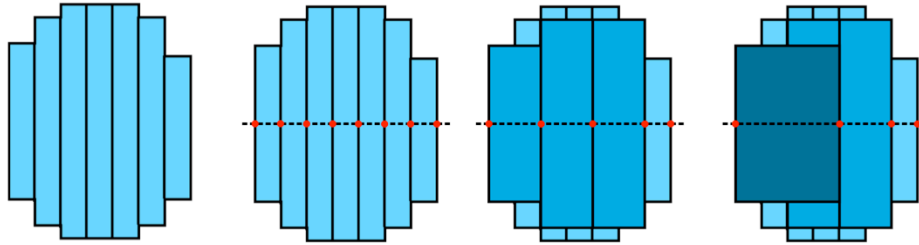


Lefebvre & Claux, 2015

Dilation or erosion is straightforward and efficiently done on GPUs between the dixels

Doxel-to-solid

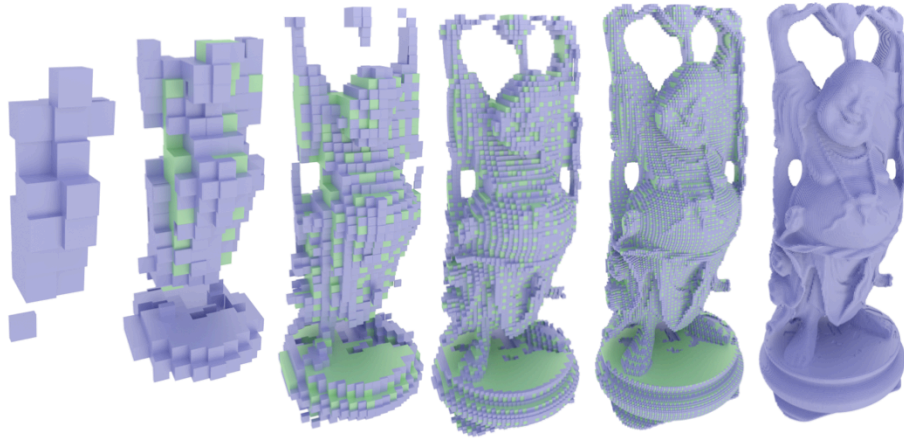
- Doxel-to-solid conversion uses hierarchy of parallelepipeds to reduce fragment count



Lefebvre & Claux, 2015

When transforming offsetted dexels back into a solid, inbetween segment offsetting steps, we avoid generating too many fragments by using a hierarchical extraction method. This alleviates GPU memory occupation and improves performance.

Doxel-to-solid



Lefebvre & Claux, 2015

Model: Stanford University Computer Graphics Laboratory

We can see here the various boxes extraction, at all resolutions.

More morpho modeling

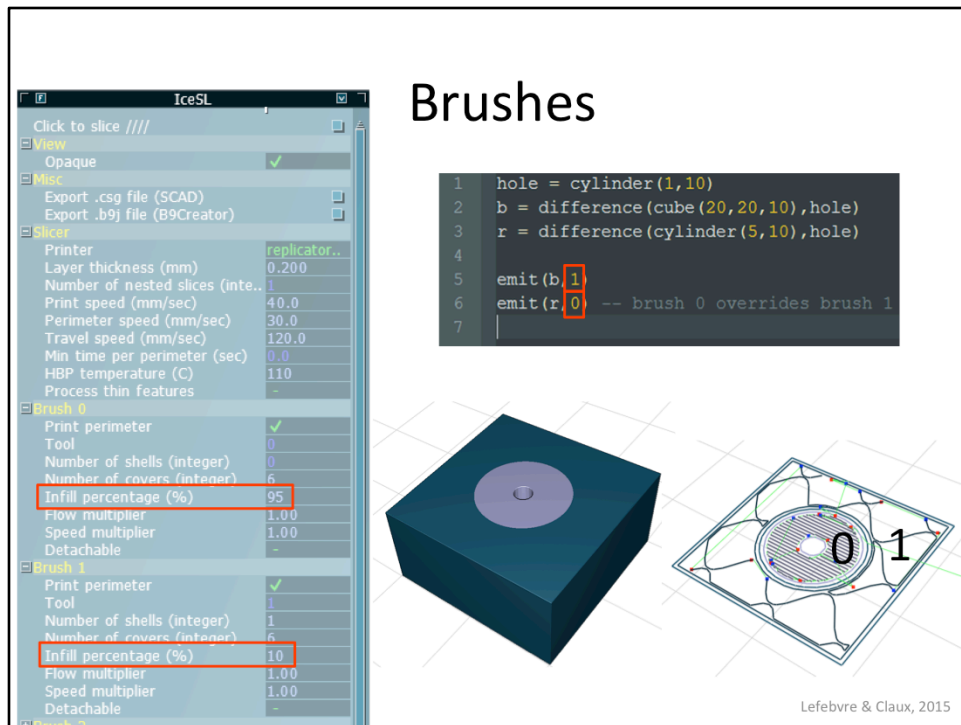


$$(\mathcal{M} \setminus ((\mathcal{M} \ominus \mathcal{Z}_1) \oplus \mathcal{Z}_1)) \oplus \mathcal{Z}_2$$

Lefebvre & Claux, 2015

Model: Stanford University Computer Graphics Laboratory

Morphological operations can also be used for other purposes. Here they're used to detect small features and highlight them.



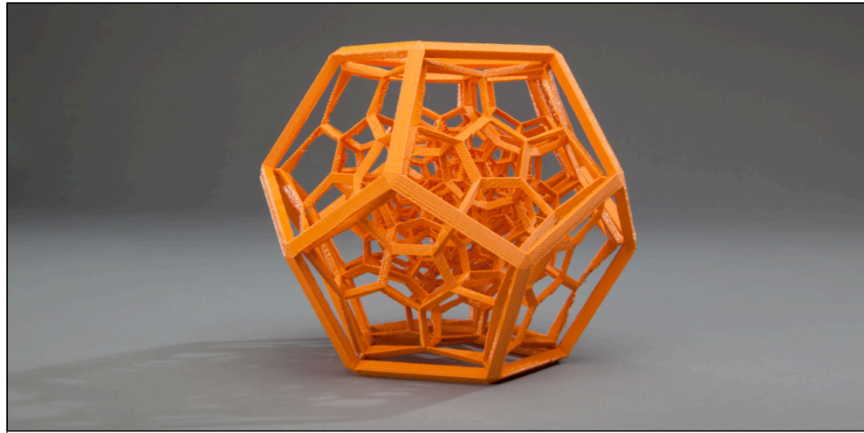
We've seen how perimeters, shells and infills work previously. These printing parameters are defined through objects called 'brushes' in IceSL. Brushes are useful when it comes to controlling material density. For example, one can define a brush with high infill percentage for object parts known to undergo high mechanical stress once printed and used. This is the case with the inner cylindrical part of this object, which uses brush #0. Remaining parts use brush #1, with lower infill percentage. The lowest brush number takes precedence when overlaps occur.

Test it!

- IceSL is available there:
 - <http://webloria.loria.fr/~slefebvr/icesl>
- Requirements:
 - OpenGL 4.2 support
- Comments and suggestions welcome
 - <https://groups.google.com/forum/#!forum/icesl>

Lefebvre & Claux, 2015

IceSL has many features, and is available for free! We provide a Windows version with an installer. Contact us for a Linux version. We are also open to extending the scripting language to help teams doing research on slicing.



SLICE AND MESH REPAIR

Modeling for Filament-Based 3D Printing



GEOMETRIC REQUIREMENTS

- Slice geometry to generate contours for toolpath generation
- Manifold and watertight geometry required to disambiguate inside/outside

Dinh & Gelman, 2015

Slice geometry to get outlines which determine our shells, infill, etc.

In order to disambiguate for a slice what's inside and outside, the 3D geometry has to be manifold and watertight. And of course, manifold means There are no dangling geometry or overlapping geometry that would make it unclear what is considered inside the object and what's outside.



TYPICAL MESH PROBLEMS

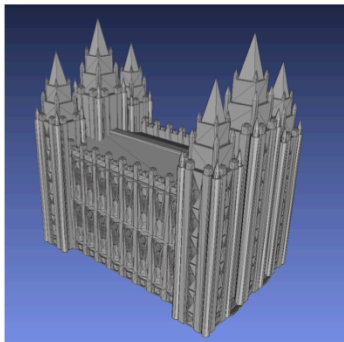
- Non-manifold
- Not watertight
- Inconsistent orientation
- Self-intersecting meshes

Dinh & Gelman, 2015

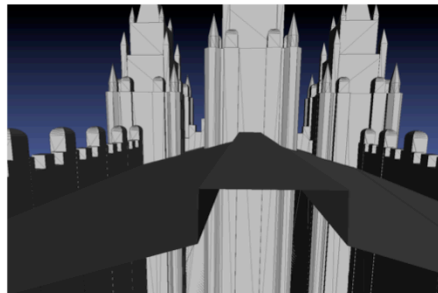


PROBLEM MESHES

Users often create complex geometry by overlaying meshes without properly merging them



[The Salt Lake City Temple (Mormon)
([kafarn](#)) / [CC BY-SA 3.0](#)]



Interior View of The SLC Temple

Dinh & Gelman, 2015

Here are just some examples of problematic meshes.

Many self-intersections – it is very typical of consumers to create geometry by overlaying primitive shapes. They only care about the outer surface.

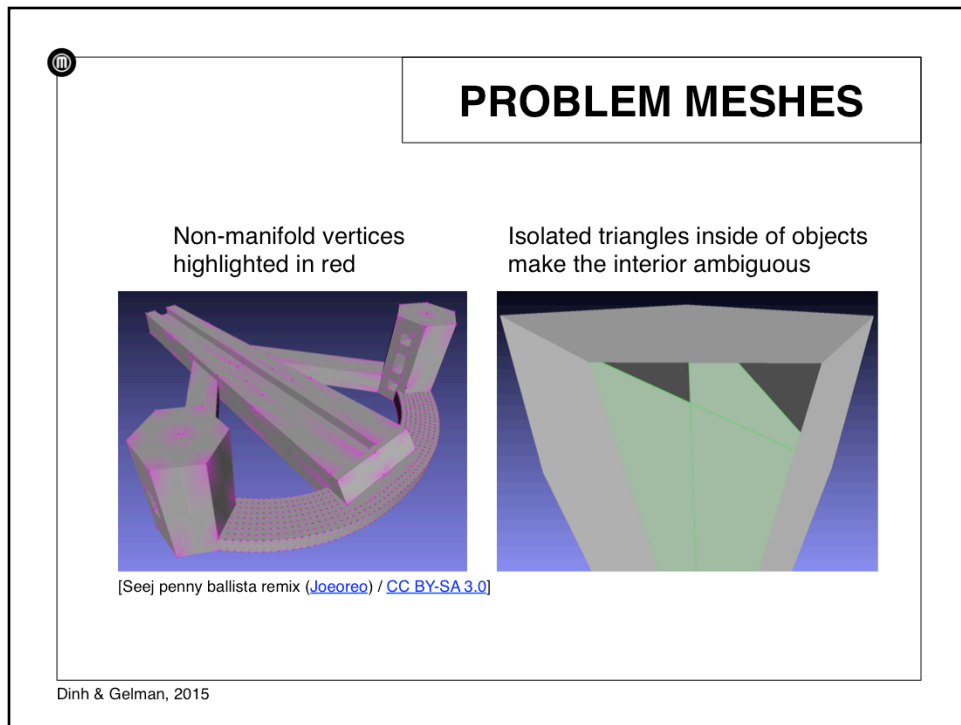
We are able to slice this type of geometry by union-ing in 2D all intersecting contours.

Attribution:

<http://www.thingiverse.com/thing:805594>

<http://www.thingiverse.com/kafarn>

<http://creativecommons.org/licenses/by-sa/3.0/>



Left: Vertex non-manifolds are highlighted, mostly from boundary and overlapping triangles.

Right: Cut-out of mesh with interior, isolated triangles

Attribution:

<http://www.thingiverse.com/thing:123425>

<http://www.thingiverse.com/Joeoreo/about>

<http://creativecommons.org/licenses/by/3.0/>



SLICE-BASED REPAIR

- Make surface orientation consistent for manifold meshes
- Union all intersecting parts
 - Self-intersecting
 - User-specified intersections via plating
- Apply printer constraints
 - Watertight with respect to build plate
 - Larger than print volume
 - Wall thickness given filament width

Dinh & Gelman, 2015

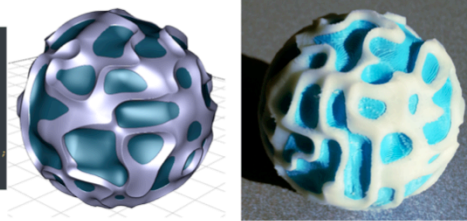
We do minimal mesh repair in that we simply make the surface orientation consistent. We leave the non-manifolds and keep track of them during slicing.

We union all intersecting parts using a particular fill rule.

References:

For a complete review of mesh problems and 3D repair strategies, see:
"A Practical Guide to Polygon Mesh Repairing" held at Eurographics 2012
http://www.meshrepair.org/eg2012_meshrepair_slides.pdf

```
sphA = implicit(v(-30,-30,-30), v(30,30,30), [{  
  float minDistanceSphereTracing=0.001;  
  float perturb(vec3 p)  
  {  
    return 5.0*abs(noise(p/7.0+2.0));  
  }  
  float distanceEstimator(vec3 p)  
  {  
    return 0.3*(max(sphere(p,26),-sphere(p,23)) + perturb(p));  
  }  
}])
```



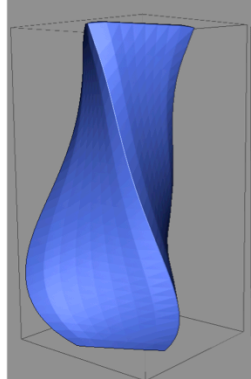
LARGE VOLUMES AND DETAILED OBJECTS

Modeling for Filament-Based 3D Printing

Slice shaders

```
int material(vec3 p,int groups)
{
  if (groups > 0) {
    int g = int(p.z*6.0+1.2*sin(p.y*5.0)+0.7*cos(p.x*3.0));
    if ( (g % 2) == 0 ) {
      return 0;
    } else {
      return 1;
    }
  }
  return -1;
}
```

[yet another vase (joo) / CC BY-SA 3.0]



Print and picture by S. Lefebvre



Procedural Texture [Perlin 1985]

Cost = code only
applied to slice (implicit)

We present now slice and warp shaders.

In computer graphics, it is sometimes desirable to define the exact appearance and shape of a surface using shaders instead of using explicit geometry. The same is true in the context of 3D printing where the concept of shaders can be applied solids.

Slice shaders can be used to procedurally select the printing material within a slice, in the context of multi-material printing. The print head position 'p' is here passed to the 'material' function which returns the material number to use at that specific position, or -1 if no material should be deposited at all.

Tool path generation will take the output of the slice shader for each slice.

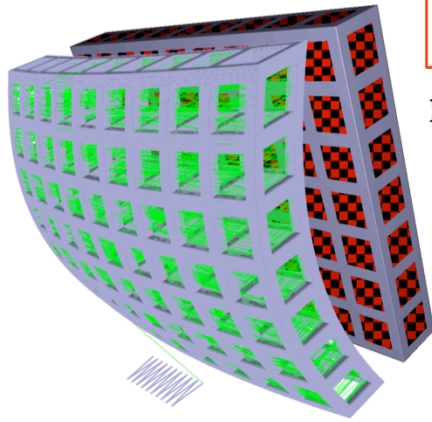
Attribution:

<http://www.thingiverse.com/thing:16378>

<http://www.thingiverse.com/joo/about>

<http://creativecommons.org/licenses/by-sa/3.0/>

Warp shaders

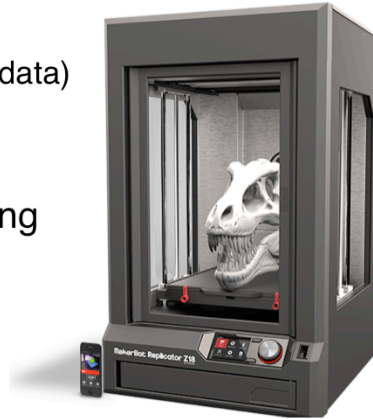


```
vec2 warp(vec3 p)
{
    vec2 d = 0.2 * vec2( 0.0, cos((p.z-0.5)*2.0)
        + 0.5 * cos((p.x-0.5)*2.0) );
    return p.xy + d;
}
```

Warp shaders are available to procedurally alter slice geometry. Slice and warp shaders can be combined as is the case on this slide. A warp shader is used to twist the solid, and a slice shader is used to hollow cubical parts out of it. As you can see we leverage GLSL all the way down, for both the rasterization and printing pipeline.

MESH > MAIN MEMORY

- High-resolution models (e.g., scan data)
- Out-of-core mesh processing & slicing
- Out-of-core file management



Dinh & Gelman, 2015

How do we handle large models that don't fit into main memory – e.g., 28 million polygons on Lucy model?

Use out-of-core slicing.



MESH > MAIN MEMORY

- Out-of-core processing required:
 - Model loading and slicing
 - Simplification (for display)
 - Analysis and repair
 - Union-ing contours
- Approaches
 - “Out-of-Core Compression for Gigantic Polygon Meshes”, Isenburg, M. and S. Gumhold, Proceedings of SIGGRAPH 2003
 - “Out-of-Core Construction and Visualization of Multiresolution Surfaces”, Lindstrom, P., Proceedings of SIGGRAPH 2003
 - “Manifold-guaranteed out-of-core simplification of large meshes with controlled topological type”, Liu et al., The Visual Computer, 2003

Dinh & Gelman, 2015

So we’ve solved the problem of having a dense toolpath that doesn’t fit into main memory.

What about large models that don’t fit into main memory – e.g., 28 million polygons on Lucy model?

Use out-of-core slicing.

References:

<http://www.cs.unc.edu/~isenburg/oocc/>

https://computing.llnl.gov/vis/images/pdf/I3D03_Lindstrom.pdf

<http://link.springer.com/article/10.1007%2Fs00371-003-0222-2>



Z18: TOOL PATH > MAIN MEMORY

- **Example:** slice a cube at full build volume, 100 microns, standard 10% infill
- **Solution:** stream data through stages
- Memory usage reduced from 6 GB to 300 MB
- Software architecture considerations:
 - Easy to create new stages and define dependencies
 - Most inter-stage dependencies known at compile time
 - Dynamically link stages based on print properties

Dinh & Gelman, 2015

So we have a solution for meshes that don't fit into main memory. What about the problem of having a dense toolpath that doesn't fit into main memory. This is the case we encountered when generating a toolpath for the full Z18 build volume.

Example: tool path we get is very dense.

Solution: Stream the data through stages. The amount of data in memory at any time is dynamically sized based on a window of data. We can restrict the size of the window so that it fits within main memory.

Most stage inter-dependencies are known at compile time, so that we get static connections, and the compiler can optimize the code.

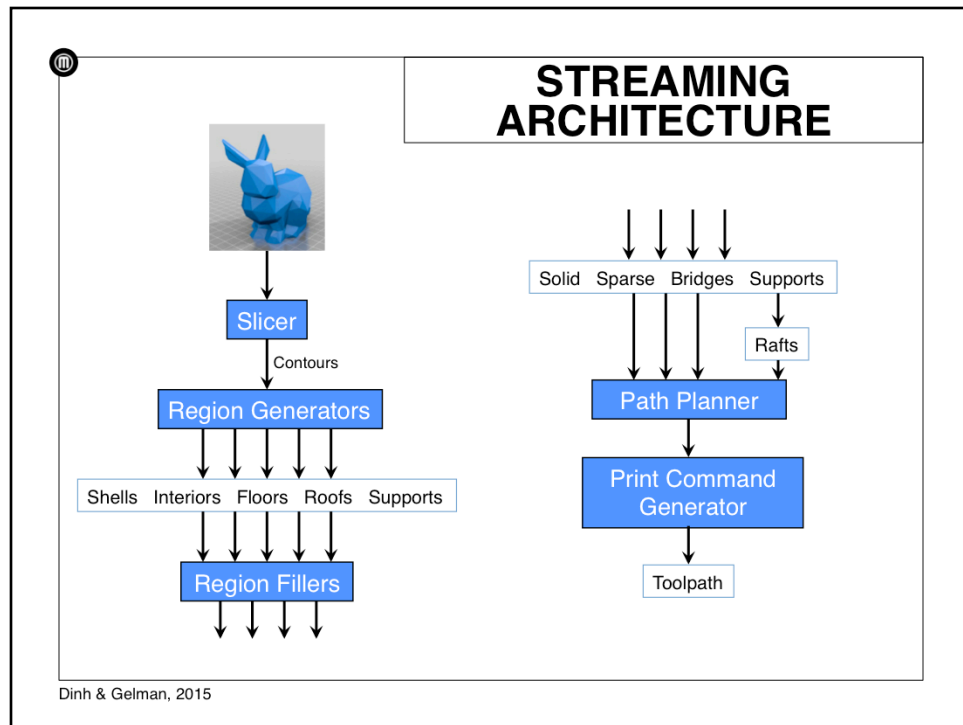
So what does this architecture look like?

Reference:

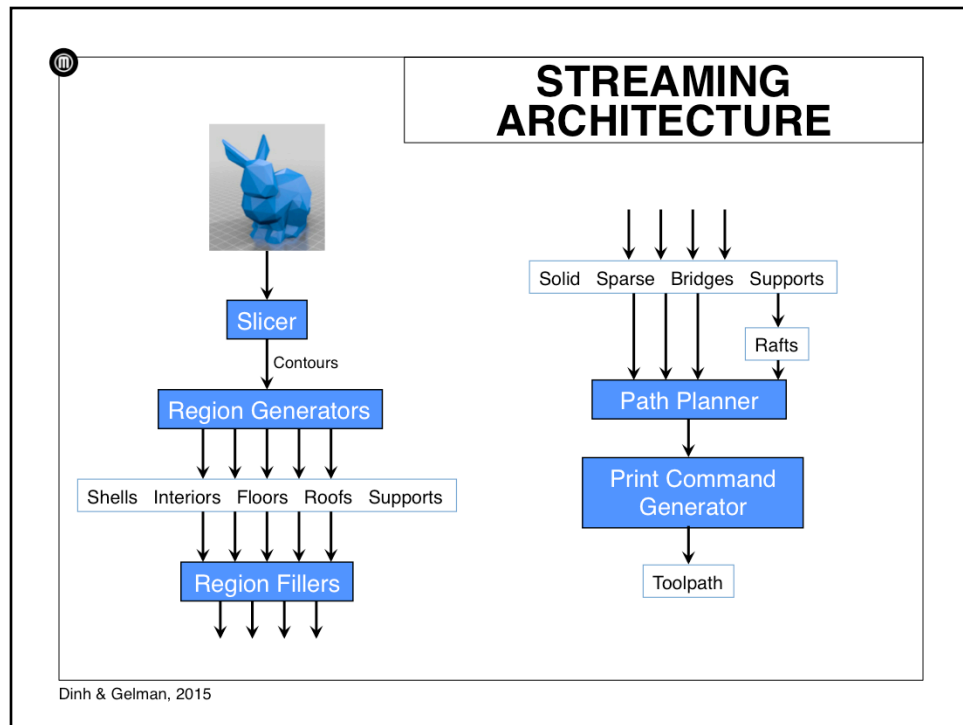
Similar streaming architecture for Polyjet-style printers (Objet Connex 500)

"OpenFab: A Programmable Pipeline for Multi-Material Fabrication", Vidimce et al., SIGGRAPH 2013

The stages and data stream in OpenFab are quite different, however.



- 1) Slicer - First stage is the actual slicing of the 3D geometry to obtain 2D outlines of the object being printed.
- 2) Region Generators – Next the outlines go through a series of region generators. These generators construct regions such as where the interiors are where we'll print infill, solid floors, solid roofs and where supports will need to be printed. In order to determine where supports are needed, the model is essentially sliced from top down. For example, outlines from above that are larger than outlines below means that supports will be needed in the top layer that is exposed.
- 3) Filler - Once the regions have been determined, the filler stage then fills out these regions with solid or sparse infill, with bridge fill, or with supports. At this time extrusion guards are also added, and rafts generated under the object and supported areas. At the end of this stage, we now have specified all tool paths to be printed, but they are not strictly ordered – the only real ordering we have right now is the bottom up, layer to layer, ordering. But within a layer, all the paths are unordered.



4) Path Planner – All these unordered paths are then processed by Path Planner which generates a toolpath according to a set of strict rules, plus some heuristics: Shells are printed before infill.

Adjacent shells are printed innermost to outermost

Start at point closest to last when going from layer to layer

Avoid crossing over exterior parts of the print (e.g., roofs and exposed areas)

5) Command Generator – coming out of Path Planner the toolpath is just a list of coordinates telling us where to extrude. The Command Generator then adds speed, temperature, and fan adjustments, and then outputs these commands into a toolpath file (like gcode, or in our case a json toolpath).

Developers:

Filipp Gelman
Andrey Patrov
Gregory Studer
Michael Zappitello

Print Quality Maven:

Rebecca Levitan

Former Member & Architect:

Joseph Sadusk

Lead:

H. Quynh Dinh

ACKNOWLEDGEMENTS

MakerBot Toolpath Team



OPEN CHALLENGES / Q&A

11:45am – noon