



Computational Fabrication

William Thenaers

Lieven Libberecht

Groep 3
Academiejaar 2019-2020
2020-01-06



Aanpak

Om onze slicer genaamd "pancakes" te maken, gebruikten we C# WPF forms. Een 3D model inladen en weergeven gebeurt met behulp van de Helix toolkit. Voor de preview van de slices gebruiken we een pan- en zoomable canvas. Zoals aangegeven in de opdracht werd de Clipper library gebruikt voor polygon operaties.

We hebben de richtlijnen van de opdracht ongeveer chronologisch gevolgd:

- UI en algoritme om modellen te slicen en instellingen aan te passen;
- 3D object omzetten in Polygon3D objecten;
- Snijpunten zoeken op de hoogte van de snijlaag;
- Gevonden lijntjes verbinden om Polygon2Ds te vormen;
- Het volledige model eroderen met de helft van de nozzle thickness (offset);
- Een G-code generator schrijven om polygons om te zetten naar G-code instructies;
- Holes detecteren en ondersteunen;
- Extra shells genereren rond holes en binnen contours;
- Basis sparse infill in de vorm van rechthoeken en andere vormen;
- Floors en roofs detecteren, invullen en propageren, floors naar boven en roofs naar onder. Propageren met de hoeveelheid shells;
- Plaatsen waar support nodig is detecteren en aanduiden met polygons;
- Support polygons invullen met rectangle structuur en andere.

Alle code staat op een Github repository, zodat we samen op verschillende branches konden werken.

Taakverdeling

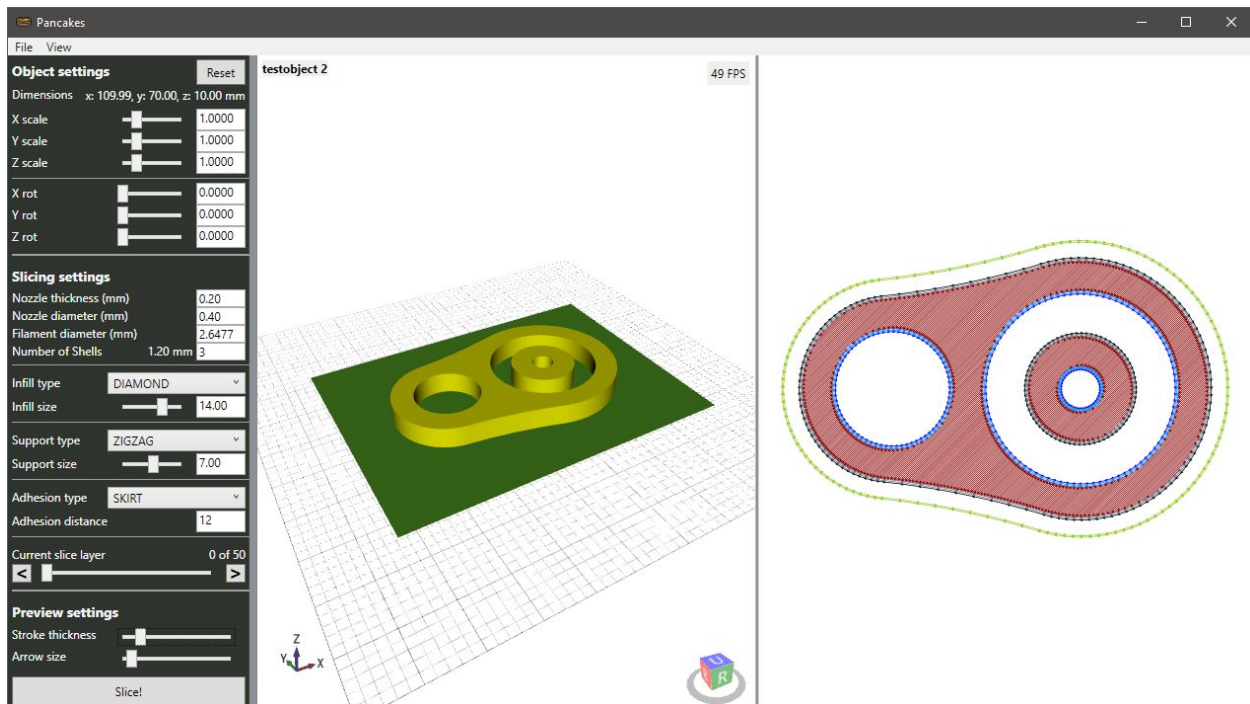
Om taken te verdelen en het project te bespreken, spraken we elke maandag op school af voor en na de les. In het algemeen implementeerde Lieven meestal de basis van de core features en breidde William die dan verder uit en voegde er soms wat extra features aan toe.



Technische implementatie

GUI

- De GUI is gemaakt in WPF, het importeren van het object gebeurt via Helix en de preview slice wordt getekend op een pan-and-zoom canvas ([Wpf.Controls.PanAndZoom](#)).
- Alle relevante parameters voor het slicen zijn aanpasbaar in de interface.
- Ook kunnen we het ingeladen model transformeren door driehoeken uit de ingeladen STL te transformeren met een transformatiematrix. Deze wordt opgesteld met de scale en rotation parameters.
- Achterliggend wordt het object ook automatisch gecentreerd op de printplaat van de printer.
- Aan de rechterkant wordt een preview getekend van de huidige slice met kleuren (zie extra's)



Figuur 1: De user interface met testobject 2 geopend, Links alle mogelijke instellingen, midden de Helix 2D viewport en rechts het preview canvas.



Slicing

- Uit de STL worden Polygon3D objecten gemaakt, bestaande uit Vertices. (Elk object heeft er drie, want een STL bevat enkel driehoeken)
- Deze objecten worden vervolgens "gesneden" op een bepaalde z-hoogte, en van de snijpunten worden Line objecten gemaakt of een Polygon2D indien alle punten van de Polygon3D in het vlak vielen.
- Vervolgens trachten we deze Lines aan elkaar te verbinden om Polygon2D objecten te verkrijgen. Bij elke nieuwe gevonden lijn, gaan we de bestaande Polygon2Ds af en kijken we of de lijn aan een begin of eindpunt toegevoegd kan worden.
- Dit proces herhaalt tot alle driehoeken uit de STL gesneden werden.
- Voor Polygon2Ds die nog open zijn, trachten we met elkaar te verbinden indien mogelijk, indien dit niet gaat, proberen we ze alsnog te verbinden door begin en eindpunten te vergelijken met een hogere equality epsilon waarde. Als ze in dit laatste geval terecht komen, worden ze in het roze getekend in de preview.
- We schreven dit algoritme al vóór we de slides erover kregen die over dit stuk ging, en hebben het daarna ook niet meer aangepast, omdat het goed leek te werken. Later ondervonden we wel enkele problemen waar sommige punten leken weg te vallen en daarmee ook volledige polygons.
- Na het vinden van alle Polygon2Ds, gaan we na wat de onderlinge hiërarchie is door te testen welke polygons welke andere omvatten, en sorteren we ze van buiten naar binnen. Zo markeren we de juiste polygons als contours of als holes.

Processing

- Na het eroderen van deze contours en holes, gaan we door de lijst van Slices van bovenaf en kijken we waar supports nodig zijn. Deze support polygons worden vervolgens naar de lagen eronder gepropageerd om de supports te kunnen laten starten van het grondvlak.
- De volgende stap is elke Slice te vergelijken met die eronder en erboven om oppervlakken te detecteren (floors en roofs). Deze detectie gebeurt zoals beschreven in de slides.
- Startende vanaf de onderste Slice, worden dan floors naar boven gepropageerd, en dit het aantal keer dat er Shells nodig zijn (volgens GUI parameter). Hetzelfde wordt gedaan voor roofs, startende vanaf de bovenste Slice.
- Shells worden gemaakt door de polygon te offsetten (naar binnen of buiten afhankelijk van of de polygon een contour of een hole is).
- Hierna worden de surface polygons van elke Slice ingevuld met dense infill, de support polygons met support infill en de binnenkant tussen contours en holes met gewone sparse infill. Na het invullen van een surface of support polygon, wordt die polygon zelf verwijderd uit de Slice, wel gebruiken we ze eerst om ze van de sparse infill af te trekken, zodat infills niet overlappen.



- Als de adhesion optie in de GUI geselecteerd is, krijgt de onderste Slice ook nog een brim of skirt.
- Na al deze stappen te doorlopen, worden de polygons uit elke slice ook gecached om ze te tekenen in het preview canvas.
- Waar mogelijk, wordt elke stap uitgevoerd op het maximale aantal threads dat beschikbaar is. Dit zorgt voor een sterke versnelling van het hele proces.

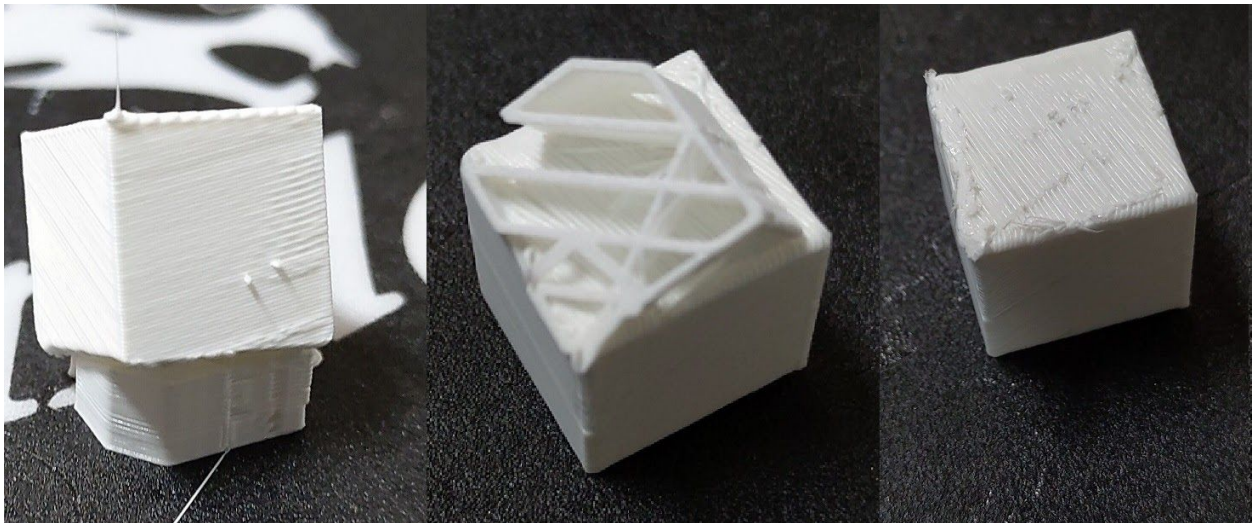
Gcode

- Een GCodeWriter klasse werd geschreven die een lijst van Slices omzet naar GCodeBase objecten.
- Elk GCode commando bevat een attribute om het type en nummer aan te geven (vb G en 1), en bevat parameters die ook elk een attribute bevatten om de Gcode parameter type aan te geven (vb X, Y, Z, E, F...).
- Op deze manier kunnen we makkelijk een GCode command toevoegen met meer intuïtieve benamingen voor parameters.
- Door op elk GCodeBase object simpelweg ToString() aan te roepen, krijgen we de string representatie van dat commando.
- Deze strings worden vervolgens naar een bestand weggeschreven.



Geïmplementeerde extra's









- Extra types infill, deze worden ook gedeeld met supports:
 - Geen
 - Single lines (aparte lijntjes, zoals dense infill, maar dan met meer ruimte onderling)
 - Single lines (90° gedraaid)
 - Rectangle (aparte rechthoeken)
 - Square (overlappende rectangles)
 - Diamond, zoals Single lines, maar in ruit patroon
 - Triangles, zoals Single lines, maar in driehoek patroon
 - Tri-hexagons, zoals Single lines, maar in hex-driehoek patroon
 - En voor supports, Zig-zag, zoals single lines, maar eind en beginpunten doorverbonden
- De afstand tussen de lijnen in het gekozen infill patroon kan gewijzigd worden;
- Support wordt gegenereerd voor alle delen van het object die meer dan 45° afwijken van de vorige laag, dus objecten zouden in elke oriëntatie geprint kunnen worden.

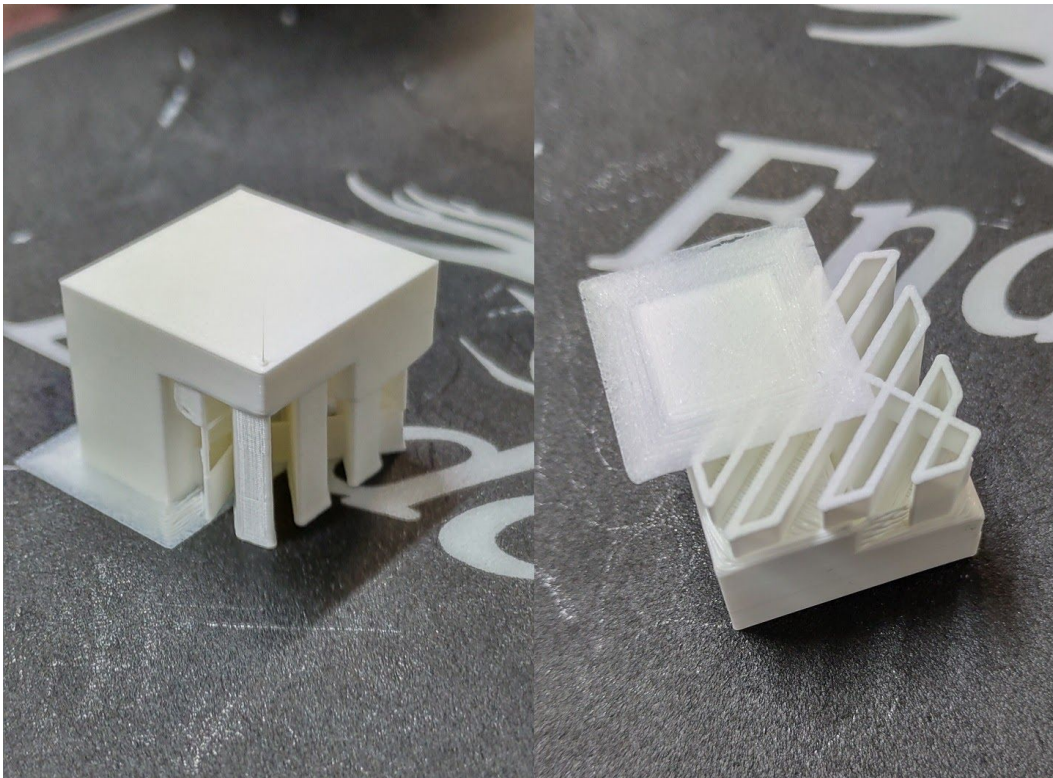


Figuur 2: Een kubus met links de print georiënteerd op één hoekpunt, midden de onderkant, waar de support goed te zien is en rechts waar de support werd weggenomen.

- Met de uitgebreide GUI kunnen verschillende instellingen gewijzigd worden. Men kan onder andere de schaal en oriëntatie van het model aanpassen om de print te optimaliseren (bijvoorbeeld om supports weg te werken). Dit gebeurt echter enkel manueel.
- Wel wordt de print automatisch gecentreerd op de printplaat van de printer.
- Indien door het object te roteren het in het 3D voorbeeld niet meer de grond raakt, worden intern de onderste lege lagen genegeerd bij het genereren van gcode.
- Er is ook een indicatie van de bounding box van het object, deze afmetingen zullen rood worden indien ze buiten de printplaat vallen.



- Toevoeging van brim en skirt.
- De nozzle en filament parameters kunnen ook aangepast worden, alsook het aantal shells.
- Het preview canvas is pan- en zoomable en de verschillende lijnen werden ingekleurd met volgende legende:
 -  Contour polygons
 -  Contour shell polygons
 -  Hole polygons
 -  Hole shell polygons
 -  Surface fill lijntjes
 -  Infill lijntjes
 -  Adhesion en support lijntjes
 -  Polygons die "open" leken, maar die daarna "gecorrigeerd" werden en toch toegevoegd
- Het hele slice proces verloopt op meerdere threads waar mogelijk.
- We voorzagen ook een functie om te testen of een Polygon eigenlijk een cirkel voorstelde, zodat we in GCode ook het ControlledArcMoveClockwise commando konden gebruiken (G2), maar de Ender3 firmware lijkt dit niet enabled te hebben.

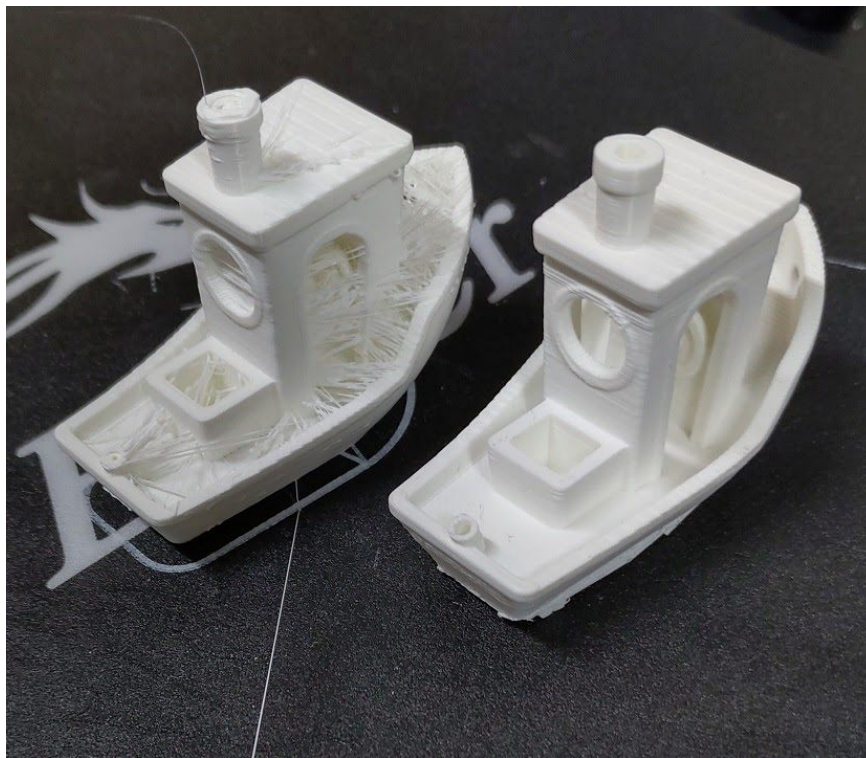


Figuur 3: Testobject 4 ondersteboven geprint om supports te forceren. Het zigzagpatroon is goed te zien, alsook de toevoeging van een brim.



Problemen

- Bij sommige modellen verdwijnen er op enkele lagen punten of driehoeken, met als resultaat dat die lagen enkele stukken missen of dat de polygons verkeerd worden verbonden. Dit zorgt dat sommige polygons dan verkeerd als holes of contour worden aangeduid.
- Zigzag support verdwijnt soms op een laag, hoewel de support polygon die ingevuld zou moeten worden wel doorgegeven wordt aan die laag en die daar onder. Er loopt dus iets mis bij het toevoegen van infill in die polygons.
- We hebben geprobeerd de G-code van Cura zo goed mogelijk te repliceren, maar we hebben nog steeds een probleem met (sterke) stringing.



Figuur 4: Hier een voorbeeld print van 3DBenchy, met links onze versie en rechts die van Cura. We zien dat bij ons er veel stringing optreedt bij het verplaatsen van de nozzle naar een andere polygon

- Overlappende shells zijn niet toegevoegd aan elkaar aangezien dit onverwachte fouten gaf, in de plaats daarvan worden ze verwijderd. Niet alle shells werden samengevoegd en bleven overlappen of werden samengevoegd in een onlogische volgorde (zelfsnijdende polygons).
- Soms verschijnen er "valse" oppervlakken die verkeerd gedetecteerd werden, maar toch worden ingevuld, dit probleem hangt hoogstwaarschijnlijk samen met het eerste probleem.