Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Drolc

Razrez 3D modela za potrebe 3D tiskanja

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Tehnologija 3D tiskanja omogoča hitro izdelavo poljubnih predmetov in je povzročila pravo revolucijo na številnih področjih. S prihodom cenovno dostopnejših 3D tiskalnikov je sedaj mogoče tiskanje izvajati že v domačem okolju, vendar le za dovolj majhne predmete. Če znamo večje predmete razrezati na več manjših delov, lahko te ločeno natisnemo ter kasneje zlepimo v enoten predmet. V diplomski nalogi predstavite svojo rešitev za razrez 3D modela. V ta namen najprej kratko predstavite značilnosti 3D tiskanja, opis 3D modelov ter sam problem razreza le-teh. Predstavite tudi postopek razreza z ustreznim algoritmom ter izdelajte učinkovito rešitev, ki iz opisa enega večjega 3D modela tvori več manjših 3D modelov določenih dimenzij. Uporabnost svoje rešitve preverite z meritvami za primerno zahteven predmet.

Za pomoč in usmerjanje pri izdelavi diplomske naloge se zahvaljujem mentorju viš. pred. dr. Igor Rožancu. Staršem pa se zahvaljujem za potrpljenje in podporo med študijem.

Kazalo

Slike

Tabele

Seznam uporabljenih kratic

kratica	angleško	slovensko
3D	three-dimensional	tridimenzionalno
STL	stereolithography - file format	stereolitografija - format datoteke
SLA	stereolithography - additive process	stereolitografija - aditivni postopek
FDM	Fused Deposition Modeling	ciljno nalaganje
SLS	Selective Laser Sintering	selektivno lasersko sintranje
LOM	Laminated Object Manufacturing	nalaganje krojenih plasti
ASCII	American Standard Code for Information Interchange	ameriški standardni nabor za kodiranje znakov
CAD	Computer-Aided Design	računalniško podprto načrtovanje

Povzetek

Naslov: Razrez 3D modela za potrebe 3D tiskanja

Avtor: Gašper Drolc

3D tiskalniki za domačo uporabo imajo omejeno delovno prostornino, zato je treba prevelike 3D modele razrezati na več delov, da jih lahko natisnemo ločeno. Cilj diplomske naloge je razviti programsko rešitev, ki omogoča razrez 3D modela na poljubno majhne dele. Rešitev naj bo preprosta za uporabo ter uporablja ustrezne algoritemske rešitve.

3D model je tipično definiran z množico trikotnikov, ki opisujejo njegovo površino. Razrez 3D modela poteka v več korakih. Najprej je treba razrezati obstoječe trikotnike ter iz njih sestaviti mnogokotnike na rezu. Za podporo pri triangulaciji površine mnogokotnika na robu smo implementirali algoritem za razvrščanje mnogokotnikov v hierarhično drevo ter algoritem za združevanje mnogokotnika z njegovimi luknjami. Samo triangulacijo mnogokotnikov smo implementirali z *Ear clipping* algoritmom. Programsko rešitev smo razvili v programskem jeziku Python z uporabniškim vmesnikom, ki deluje iz ukazne vrstice.

Po prikazu delovanja programske rešitve smo opravili tudi meritev in analizo časovne zahtevnosti posameznih korakov rezanja 3D modela. Ta je pokazala, da je triangulacija mnogokotnikov časovno najbolj zahteven korak v programu, saj čas izvajanja triangulacije z *Ear clipping* algoritmom narašča eksponentno z ločljivostjo 3D modela. Ne glede na to, se programska rešitev v praksi dobro obnese, če 3D model ni prezahteven.

Ključne besede: 3D tiskanje, 3D model, ear clipping, triangulacija, Python.

Abstract

Title: Decomposition of 3D model for 3D printing

Author: Gašper Drolc

Typical consumer level 3D printers have a limited build volume, so 3D models that are too big need to be divided into several smaller pieces that can be printed separately. The goal of this thesis is to develop a program for cutting 3D models into several smaller segments. The idea is to find a simple yet effective solution which is built with suitable algorithmic solutions.

A 3D model is typically defined with a set of triangles that describe its surface. Cutting a 3D model is performed in several different steps. We begin by developing a procedure for cutting the existing triangles, and use them to construct polygons by connecting the edges. To support triangulation of polygons with holes we implemented an algorithm for sorting polygons into a hierarchical tree and an algorithm for merging polygons with its holes. Next we implemented an *Ear clipping* algorithm for polygon triangulation. The program was developed in the Python programming language with a command line interface.

After presenting how our solution works we performed measurements and an analysis of time complexity for the separate steps. The anaylsis showed that the polygon triangulation is the most time consuming step in the program, and that the execution of triangulation using *Ear clipping* rises exponentialy wih the resolution of the 3D model. Nevertheless the solution performs well if the 3D model is of moderate complexity.

Keywords: 3D printing, 3D models, ear clipping, triangulation, Python.

Poglavje 1

Uvod

Cenovno dostopni 3D tiskalniki imajo večinoma majhno delovno prostornino, kar omejuje velikost tiskanih izdelkov. Rešitev te omejitve je razrez prevelikih 3D modelov na več manjših delov. Posamezni deli se nato natisnejo in zlepijo v končen izdelek.

Programi za razrez že obstajajo, vendar pa so pogosto plačljivi ali zapleteni za uporabo. To je tudi razlog, da smo tovrstno rešitev razvili sami.

Cilj diplomskega dela je poenostavljanje postopka za izdelavo poljubno velikih izdelkov ne glede na omejenost delovne prostornine cenovno dostopnih 3D tiskalnikov. Uporabnikom brez poglobljenega znanja 3D modeliranja želimo omogočiti izdelavo tudi zelo velikih izdelkov, ki so sestavljeni iz poljubnega števila posameznih kosov. Razvili bomo namenski program za razrez 3D modelov na dele, ki se lahko natisnejo v omejeni prostornini 3D tiskalnika. Program bo za delovanje od uporabnika zahteval samo datoteko 3D modela, želeno končno velikost izdelka in dimenzije tiskalne prostornine.

V drugem poglavju predstavimo tehnologijo 3D tiska, opišemo različne vrste 3D tiskalnikov in predstavimo postopek 3D tiskanja. V tretjem poglavju opišemo rešitev problema, predvsem vse posamezne korake razreza 3D modela ter uporabljene algoritme. V četrtem poglavju predstavimo razvit program ter opišemo opravljene meritve in analizo časovne zahtevnosti posameznih korakov razreza. Na koncu zaključimo nalogo z opisom doda-

tnega dela, ki bi ga bilo mogoče še opraviti.

Poglavje 2

Opis 3D tiska

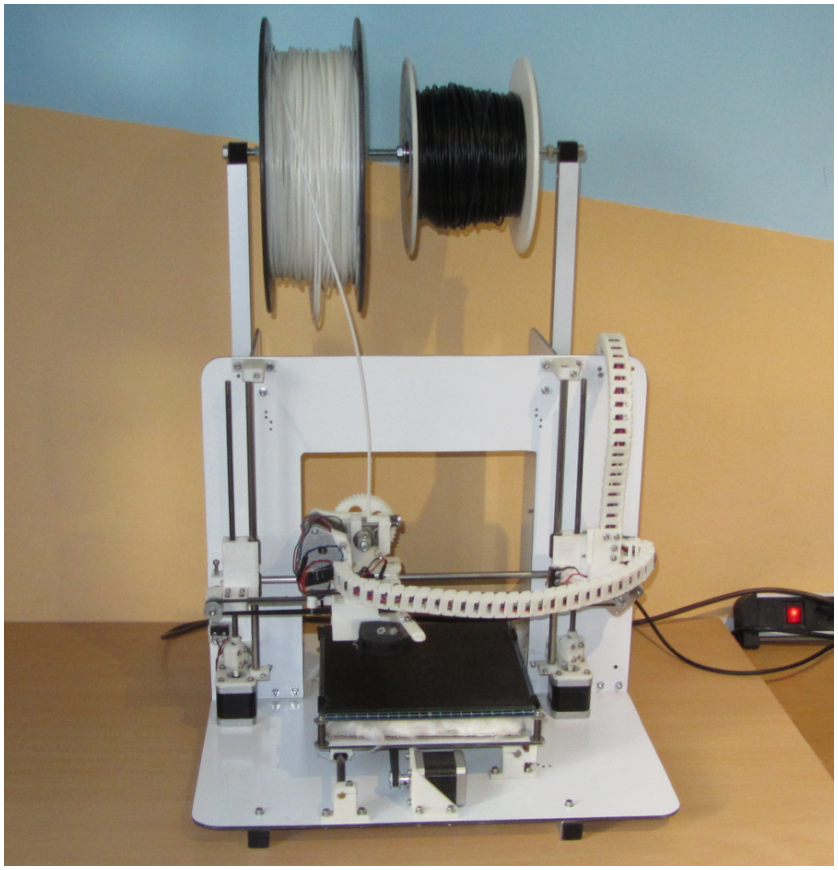
3D tisk omogoča izdelavo objektov skoraj katerekoli oblike. Tridimenzionalen objekt se izdela z uporabo aditivnega postopka, pri katerem se v zaporednih plasteh odlaga material v različnih dvodimenzionalnih oblikah [2].

Za izdelavo prototipov se v visoko tehnološki industriji ta postopek uporablja že od osemdesetih let prejšnjega stoletja. Tehnologija 3D tiska je po prihodu odprtokodnega projekta RepRap [23] leta 2006 postala dostopna tudi manjšim podjetjem in posameznikom. Na sliki 2.1 je primer 3D tiskalnika razvitega v projektu RepRap. Danes se 3D tiskanje uveljavlja na skoraj vseh visoko tehnoloških področjih. Napredni 3D tiskalniki kovine se uporabljajo v raketni [28], letalski [1] in avtomobilski [12] industriji. V medicini se raziskuje tiskanje nadomestnih organov [14], kosti [25], ter protez [11]. Cena cenovno dostopnejših 3D tiskalnikov se je spustila pod 200 evrov, kar je že primerljivo s cenami tiskalnikov na papir.

2.1 Vrste aditivnih postopkov

Razviti so številni aditivni postopki. Razlikujejo se po ceni, hitrosti, natančnosti, uporabljenih materialih in mehanskih lastnostih končnih izdelkov. Spodaj so našteti nekateri izmed postopkov.

- SLA - stereolitografija (angl. stereolithography)



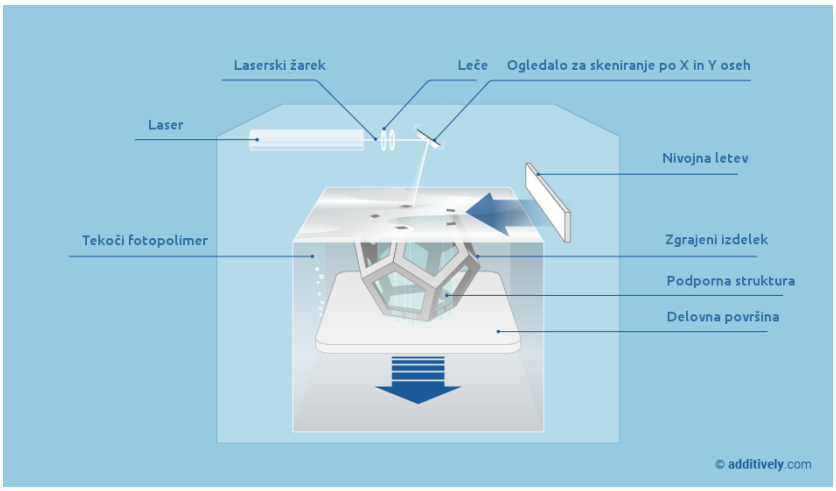
Slika 2.1: 3D tiskalnik Mendel90 iz projekta RepRap

- FDM - ciljno nalaganje (angl. fused deposition modeling)
- SLS - selektivno lasersko sintranje (angl. selective laser sintering)
- LOM - nalaganje krojenih plasti (angl. Laminated object manufacturing)

2.1.1 Stereolitografija

Stereolitografija (SLA) je eden izmed najstarejših komercialnih aditivnih postopkov. Končni izdelek se gradi v slojih s strjevanjem tekočega fotopolimera. Diagram postopka je prikazan na sliki 2.2. Fotopolimer je poseben material

ki se strdi ob stiku s svetlobo. Posamezni 2D sloji se gradijo z uporabo laserja, ki material strdi na primernih mestih [17].



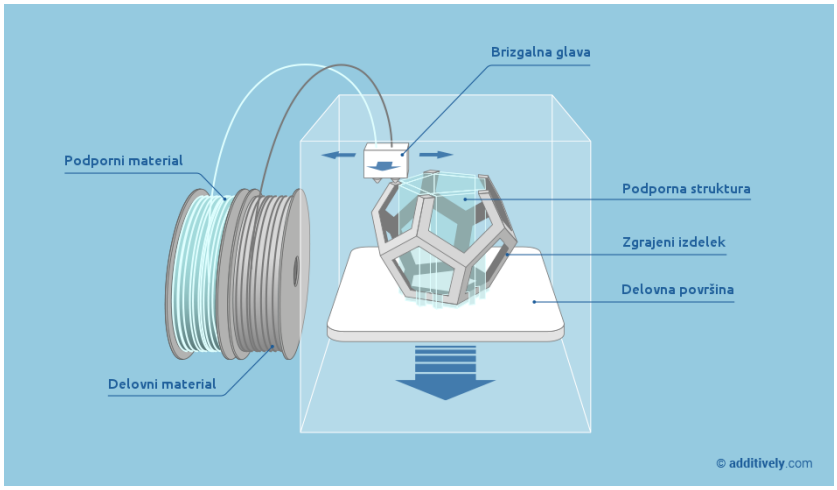
Slika 2.2: Postopek SLA [17]

2.1.2 Ciljno nalaganje

Pri cenovno dostopnejših tiskalnikih se uporablja aditivni postopek ciljno nalaganje (FDM). Ta postopek gradi 3D izdelek s ciljnim nalaganjem termoplastičnega materiala. Termoplastični materiali se pri višji temperaturi iz trdnega stanja stopijo v delno tekoče stanje. Delovni material se v brizgalni glavi segreje do tekočega stanja in odlaga na površino v dvodimenzionalnih oblikah. Z zaporednim nalaganjem 2D oblik se postopoma izdela končni tridimenzionalen objekt [2]. Diagram postopka FDM je prikazan na sliki 2.3.

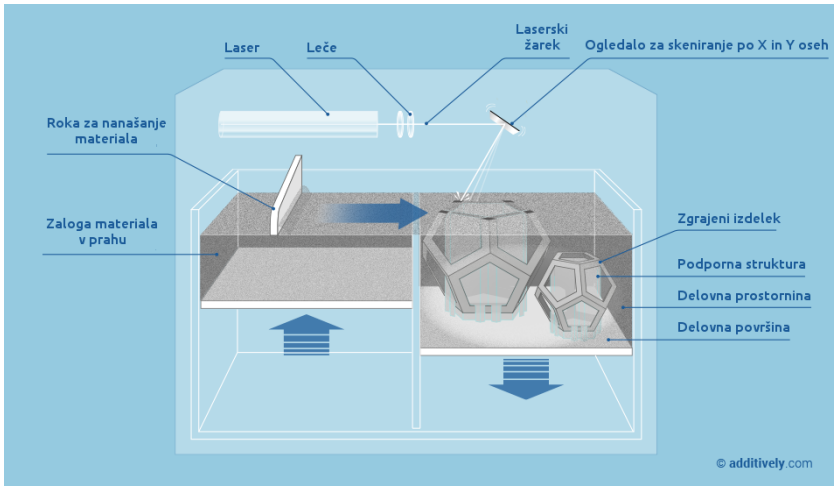
2.1.3 Selektivno lasersko sintranje

Selektivno lasersko sintranje (SLS) uporablja podoben postopek kot stereolitografija, vendar namesto tekočega fotopolimera uporablja material v prašku. Postopek je prikazan na sliki 2.4. Sloji se gradijo z uporabo močnega laserja, ki prašek topi na primernih mestih. Po končanem sloju se po delovni površini



Slika 2.3: Postopek FDM [15]

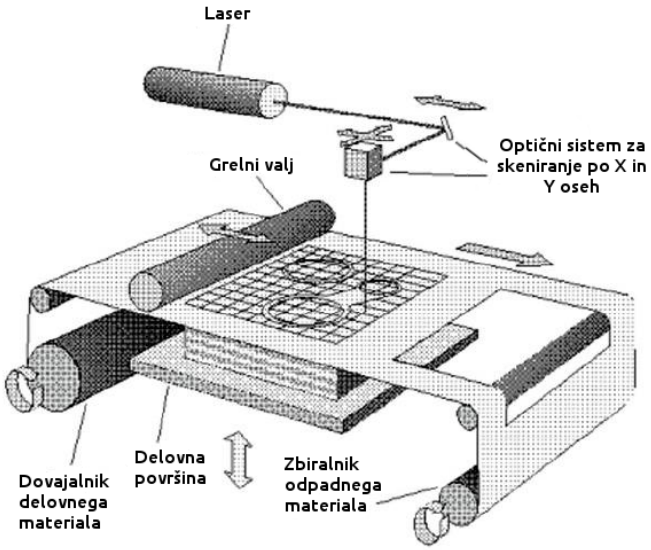
nanese tanka plast praška. Postopek lahko uporablja širok razpon materialov in poleg različnih plastik omogoča tudi tiskanje z jeklom, bronom ali titanom [2].



Slika 2.4: Postopek SLS [16]

2.1.4 Nalaganje krojenih plasti

Pri nalaganju krojenih plasti (LOM) se 3D izdelek gradi z lepljenjem posameznih plasti. Plasti se z nožem ali laserjem izrežejo v primerno obliko iz tanke folije ter nato zlepijo ena na drugo. Uporablja se lahko papir, plastika ali kovina [31]. Slika 2.5 prikazuje postopek nalaganja krojenih plasti.



Slika 2.5: Postopek LOM [31]

2.2 Postopek 3D tiska

Postopek 3D tiskanja se prične s 3D modelom. 3D model lahko oblikujemo v 3D modelirnem programu ter ga izvozimo v format STL. Primernih programov za oblikovanje 3D modelov je veliko, naštetih je samo nekaj najbolj priljubljenih.

- AutoCAD [3]
- OpenSCAD [13]

- FreeCAD [24]
- TinkerCAD [4]
- SketchUp [29]
- Blender [9]

Namesto oblikovanja novega lahko uporabimo že obstoječ 3D model, ki ga prenesemo iz spletne baze brezplačnih ali plačljivih 3D modelov. Nekaj nabolj priljubljenih baz je naštetih spodaj.

- Thingiverse [18]
- Cults [5]
- Pinshape [8]
- MyMiniFactory [20]

Naslednji korak je predelava 3D modela v strojno kodo 3D tiskalnika (angl. G-code) s programom za razslojevanje (angl. slicer). Program za razslojevanje izvorni 3D model razreže na zaporedno serijo slojev fiksne višine, ki skupaj aproksimirajo izvorno 3D obliko. Posamezni sloji so sestavljeni iz serije povezanih daljic neničelne širine, ki so v strojni kodi predstavljeni z ukazi za premikanje brizgalne glave ter hitrostjo odlaganja materiala. Bolj priljubljeni programi za razslojevanje so:

- Slic3r [22],
- Ultimaker Cura [30],
- Simplify3D [27].

3D tiskalnik nato bere in izvaja ukaze v strojni kodi. Brizgalna glava glede na ukaze v strojni kodi odlaga črte stopljenega materiala v slojih do končnega izdelka. Material se po odlaganju spoji z že nanešenimi deli izdelka. Čas tiskanja je odvisen od velikosti objekta, nastavljene gostote objekta in hitrosti tiskanja. Traja lahko od nekaj minut za manjše objekte in do več dni za večje objekte.

2.3 Postopki ki omogočajo tiskanje večjih izdelkov

Preveliki 3D modeli se lahko na več načinov razrežejo na manjše komponente. Lahko že ob oblikovanju 3D modela upoštevamo omejitve velikosti ter ga oblikujemo v več manjših delov, ki skupaj sestavljajo celoto. V tem primeru lahko posameznim komponentam na rezu dodamo tudi elemente za poravnavo pri sestavljanju končnega izdelka. Za uporabo te metode je potrebno napredno znanje oblikovanja 3D modelov.

Če uporabljamo že obstoječi 3D model, ga lahko uvozimo v enega izmed 3D modelirnih programov. Model nato ročno razrežemo na manjše kose z uporabo modelirnega programa. Obstoječi 3D modeli namenjeni za 3D tiskanje so večinoma shranjeni v STL formatu, ki ga podpirajo skoraj vsi 3D modelirni programi.

- Odprtokodni program Blender [9] že omogoča rezanje 3D modela z namenskim orodjem nož (angl. knife), vendar pa je težaven za uporabo in zahteva vsaj osnovno znanje uporabe 3D modelirnih programov. Pri rezanju na 10 ali več delov pa je lahko tudi precej zamudno.

- Komercialni 3D modelirni program AutoCAD [3] vsebuje orodje razrez po ravnini (angl. plane cut), s katerim določimo rezalno ravnino za razrez poljubnega 3D modela. Tudi AutoCAD je težaven za uporabo in zamuden pri razrezu na več delov. Glavna slabost pa je cena, mesečna naročnina stane več kot 200 evrov.

- Odprtokodni program OpenSCAD [13] vsebuje orodje prerez (angl. intersection), ki vrne prerez med 3D modeli. Posamezen del končnega izdelka lahko dobimo s prerezom med poljubnim 3D modelom ter kocko, manjšo od delovne prostornine 3D tiskalnika. Za preostale dele ponovimo postopek s translacijo kocke na primerne koordinate. Program je počasen pri 3D modelih višje ločljivosti ter zapleten za začetnike, saj za delovanje uporablja posebej razvit domensko specifičen jezik.

2.4 Format STL

Format STL je razvilo podjetje 3D Systems za shranjevanje površin 3D modelov [32] v programih za računalniško podprto načrtovanje (angl. CAD - Computer-Aided Design). Format STL je najpogostejši format 3D modela za potrebe 3D tiskanja. Geometrija 3D modela je v formatu definirana z množico trikotnikov. Trikotniki so predstavljeni z enotskim vektorjem smeri ter tremi točkami, ki definirajo pozicije robov trikotnika v tridimenzionalnem kartezičnem koordinatnem sistemu. Format podpira zapis podatkov v ASCII ali binarni obliki in je podprt v številnih programskih paketih. Vsebuje samo geometrijo modela brez barve, teksture in drugih atributov, ki se pogosto uporabljajo pri opisu 3D modelov. Spodaj je primer 3D modela tristrane piramide zapisane v ASCII obliki STL formata.

```
solid OpenSCAD_Model
  facet normal 0.480384 0.83205 0.27735
    outer loop
      vertex -0.288675 0.5 0
      vertex 0 0 1
      vertex 0.57735 0 0
    endloop
  endfacet
  facet normal -0.960769 3.73334e-16 0.27735
    outer loop
      vertex -0.288675 -0.5 0
      vertex -0 0 1
      vertex -0.288675 0.5 0
    endloop
  endfacet
  facet normal 0.480384 -0.83205 0.27735
    outer loop
      vertex 0.57735 0 0
```

```
      vertex -0 -0 1
      vertex -0.288675 -0.5 0
    endloop
  endfacet
  facet normal 0 0 -1
    outer loop
      vertex -0.288675 -0.5 0
      vertex -0.288675 0.5 0
      vertex 0.57735 0 0
    endloop
  endfacet
endsolid OpenSCAD_Model
```

Poglavje 3

Razrez 3D modela

V tem poglavju želimo opisati razviti postopek rezanja 3D modela, ki poteka v več korakih. Pri vsakem koraku želimo opisati problem in raložiti njegovo rešitev ter uporabljene algoritme.

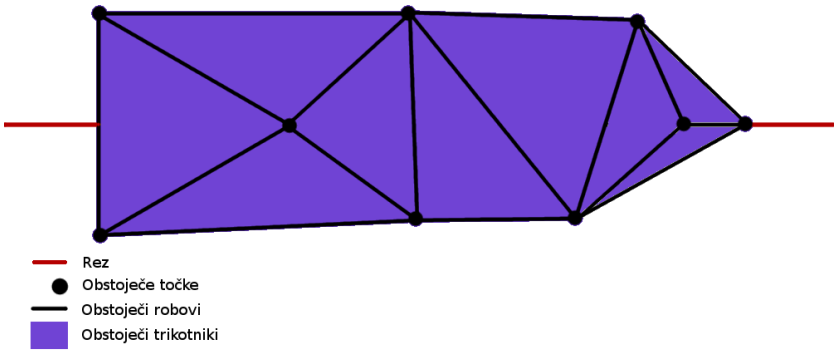
Program za razrez 3D modela smo razvili v programskem jeziku Python. Razrez se začne z nalaganjem datoteke modela v program. S knjižnico *numpy-stl* naložimo 3D model, ki je shranjen v formatu STL. V programu je površina 3D modela shranjena v objektu tipa `Model` s seznamom trikotnikov modela. Trikotniki so shranjeni v objektih tipa `Triangle`, ki hranijo koordinate točk in vektor normale posameznih trikotnikov.

Ravnine za razrez modela smo omejili na ravnine, ki ležijo pravokotno na z os. S to omejitvijo smo poenostavili implementacijo postopka rezanja 3D modela. Rez na ravnini poljubne orientacije lahko dosežemo s predhodno rotacijo celega 3D modela. Razrez je implementiran v funkciji `cut_at_z(z_height)`. Funkcija `cut_at_z(z_height)` prejme poljubno vrednost koordinate z, ki definira višino ravnine razreza.

3D model se razreže v dveh korakih. Prvi korak je **delitev in razrez obstoječih trikotnikov v seznama trikotnikov nad in pod ravnino razreza**. Po prvem koraku je izvorni model razrezan na dva modela, ki imata luknji v površini na prerezu reza. V drugem koraku **zapolnimo luknji s triangulacijo mnogokotnika na prerezu**.

3.1 Delitev in razrez trikotnikov

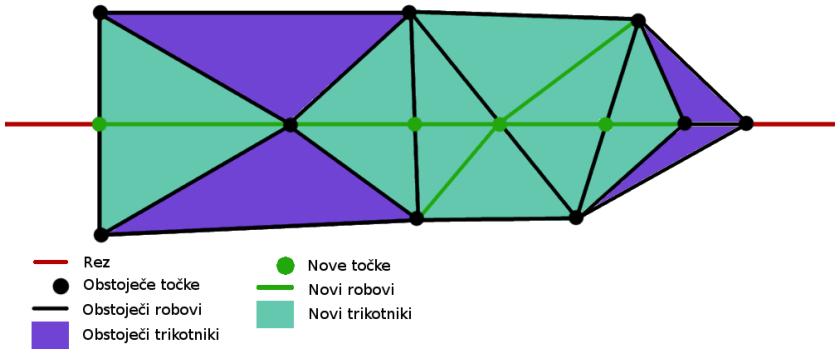
Razrez 3D modela pričnemo v funkciji `cut_at_z(z_height)` z delitvijo obstoječih trikotnikov na dva seznama `above_triangles` in `below_triangles`. V seznam `above_triangles` uvrstimo trikotnike, ki imajo `z` koordinato vseh treh točk večjo od `z_height`. V seznam `below_triangles` uvrstimo vse trikotnike, katerih `z` koordinata vseh treh točk je manjša od `z_height`. Preostale trikotnike pa razrežemo s funkcijo `cut_triangle_at_z(z_height)`.



Slika 3.1: Primer trikotnikov na rezalni ravnini

Funkcija `cut_triangle_at_z(z_height)` razreže obstoječi trikotnik na dva ali tri manjše trikotnike ter jih razvrsti v seznama `above_triangles` in `below_triangles`. Trikotnik se razreže tako, da se najprej poišče točki presečišča med robom trikotnika in ravnino prereza. Nato se iz obstoječih in novih točk sestavijo novi trikotniki. Slika 3.1 prikazuje 2D preslikavo začetnega stanja trikotnikov na rezalni raznini pred razrezom s funkcijo `cut_triangle_at_z(z_height)`.

Na sliki 3.2 je vidno končno stanje trikotnikov po razrezu. Trikotniki, ki s stranicami sekajo rezalno ravnino, se razrežejo na manjše trikotnike. Novi trikotniki so definirani iz točk starega trikotnika in presečišči stranic z rezalno površino.



Slika 3.2: Primer razreza trikotnikov na rezalni ravnini

3.2 Triangulacija prereza

Oba nova modela imata na prerezu luknji enake oblike, zato lahko triangulacijo izvedemo samo enkrat ter novo nastale trikotnike dodamo obema modeloma. Za triangulacijo smo implementirali algoritem *Ear Clipping* [7], ki je osnovan na izreku dveh ušes (angl. two ears theorem) [19]. Izrek pravi, da ima vsak preprost mnogokotnih s 4 ali več točkami vsaj dve "ušesi". "Uho"je točka mnogokotnika, ki s sosednjima točkama tvori trikotnik, ki leži znotraj mnogokotnika in ga ne prekriva noben drug del mnogokotnika. Tak trikotnik lahko odstranimo iz mnogokotnika z izbrisom "ušesa". Tako ostane mnogokotnik z manj točkami in brez površine odstranjenega trikotnika. S ponavljanjem odstranjevanja ušes lahko trianguliramo vsak preprost mnogokotnik.

Postopek začnemo z izgradnjo mnogokotnikov iz trikotnikov na prerezu. Posamezne točke mnogokotnika so shranjene v cikličnem seznamu in so urejene po vrstnem redu v mnogokotniku. Mnogokotnike nato razvrstimo v hierarhično drevo, iz katerega nato razberemo mnogokotnike novih površin s seznami njihovih pripadajočih lukenj. Mnogokotnike novih površin nato s triangulacijo pretvorimo v seznam trikotnikov teh površin ter jih združimo s pripadajočimi luknjami na prerezu.

3.2.1 Gradnja mnogokotnikov na prerezu

V funkciji `get_polygons_from_edge_triangles()` poteka gradnja mnogokotnikov iz vhodnega seznama trikotnikov na prerezu in iz višine reza. Funkcija vrne seznam vseh robnih mnogokotnikov na prerezu. Iz vhodnega seznama trikotnikov obdržimo samo robne trikotnike, ki s točno dvema točkama ležita na prerezu. Iz robnih trikotnikov se nato zgradijo mnogokotniki robov na prerezu. Točke mnogokotnika si morajo v seznamu slediti v pravilnem vrstnem redu. Zato se mnogokotnik gradi postopno iz trikotnikov, ki si sledijo eden za drugim na robu prereza. Trikotnika na robu sta si sosednja, če si na robu delita isto točko.

Gradnjo seznama začnemo s poljubnim trikotnikom. Točki začetnega trikotnika, ki ležita na prerezu, dodamo na začetek seznama novega mnogokotnika. Po obdelavi vsak trikotnik izbrišemo iz seznama robnih trikotnikov. Gradnjo mnogokotnika nato nadaljujemo z iteracijo po seznamu trikotnikov, kjer iščemo trikotnik, ki ima točko roba enako točki na zadnjem mestu v seznamu mnogokotnika. Tak trikotnik nato obdelamo tako, da drugo točko na prerezu, ki še ni v trenutnem mnogokotniku, vstavimo na konec seznama točk mnogokotnika. Iteracijo ponavljamo, dokler ne najdemo trikotnika z robovoma na prvem in zadnjem mestu v seznamu točk mnogokotnika. S tem smo smo našli vse točke mnogokotnika. Obdelani trikotnik nato izbrišemo in zaključimo gradnjo prvega mnogokotnika.

Celotni postopek ponavljamo, dokler ne zgradimo še preostalih mnogokotnikov na prerezu. Vsi mnogokotniki so zgrajeni, ko je seznam trikotnikov na prerezu prazen.

3.2.2 Obdelava mnogokotnikov z luknjami

Kompleksnejši 3D modeli imajo v površini prereza lahko tudi luknje in nove površine znotraj teh lukenj. Mnogokotniki zgrajeni v 3.2.1 predstavljajo robove teh površin. S funkcijo `assemble_polygon_sorted_tree()` ugotovimo, kateri mnogokotniki predstavljajo zunanje robove nove površine in kateri so

luknje v teh površinah. Funkcija prejme seznam mnogokotnikov in jih vrne v hierarhično urejenem drevesu.

Mnogokotniki so v drevo razporejeni tako, da vsak mnogokotnik pod sabo vsebuje mnogokotnike, ki so znotraj le tega mnogokotnika. Mnogokotnik je znotraj drugega mnogokotnika, ko so vse njegove točke znotraj tega mnogokotnika. Na prvem nivoju drevesa imamo tako mnogokotnike najbolj zunanjih robov novih površin. Njihovi otroci na drugem nivoju pa so luknje teh mnogokotnikov. Na tretjem nivoju so zunanji robovi novih površin znotraj lukenj na drugem nivoju, otroci mnogokotnikov na tretjem nivoju pa so luknje teh površin. Drevo se tako nadaljuje poljubno globoko.

Za razvrščanje mnogokotnikov v drevo smo razvili algoritem, ki razvrsti vsak posamezen mnogokotnik po naslednjih pravilih:

- Iskanje začnemo pri otrocih trenutnega vozlišča, če je novi mnogokotnik znotraj mnogokotnika v tem vozlišču.
- Novi mnogokotnik vrinemo na trenutno vozlišče, če je mnogokotnik trenutnega vozlišča znotraj novega mnogokotnika. Poddrevo starega vozlišča dodamo med otroke novega mnogokotnika.
- Novi mnogokotnik dodamo med mnogokotnike na trenutnem nivoju, če novi mnogokotnik ne vsebuje nobenega mnogokotnika na trenutnem nivoju in ni znotraj katerega izmed teh mnogokotnikov.

S temi pravili se vsi mnogokotniki razvrstijo na prava mesta v drevesu. V funkciji `is_inside_polygon(self, polygon)` se preveri, če mnogokotnik leži znotraj drugega mnogokotnika. Mnogokotnika se pretvorita v objekta tipa `Path` iz vgrajene knjižnice `matplotlib`. Vsebovanost enega mnogokotnika v drugem se nato preveri s funkcijo `contains_path(self, path)`, ki je del objekta `Path`.

Po koncu izvajanja funkcije `assemble_polygon_sorted_tree()` se drevo mnogokotnikov pretvori v seznam mnogokotnikov površin s pripadajočimi luknjami v funkciji `sorted_tree_to_sorted_polygon_list()`. Algoritem za obdelavo drevesa je opisan v dokumentaciji algoritma *Ear clipping* [7].

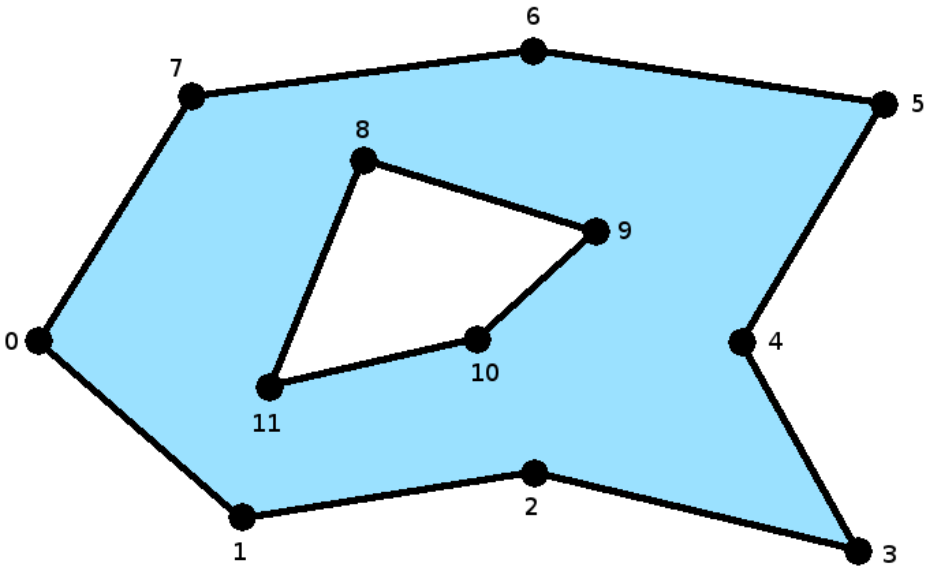
V funkciji `assemble_polygon_sorted_tree()` se v čakalni vrsti (angl. queue) hranijo vsa neobdelana vozlišča zunanjih mnogokotnikov. Na začetku se v čakalno vrsto dodajo vsa vozlišča iz prvega nivoja drevesa. Nato se mnogokotniki iz čakalne vrste postopoma obdelajo, da se jim kot luknje dodajo njihovi otroci, v čakalno vrsto pa dodamo otroke teh otrok. Obdelani mnogokotnik odstranimo iz čakalne vrste in nadaljujemo z obdelavo naslednjega.

3.2.3 Izrez lukenj iz mnogokotnikov

Mnogokotnik na prerezu lahko vsebuje tudi eno ali več lukenj. Taki mnogokotniki potrebujejo posebno obravnavo pred končno triangulacijo. Za triangulacijo takih mnogokotnikov smo implementirali postopek, ki je opisan v dokumentaciji *Ear clipping* algoritma [7]. Združevanje zunanjega mnogokotnika z njegovimi notranjimi mnogokotniki smo implementirali v funkciji `make_simple(self)`, ki pripada razredu `Polygon`.

Zunanji mnogokotnik in njegovi notranji mnogokotniki morajo za uspešno združitev imeti obraten vrstni red točk. V naši implementaciji ima zunanji mnogokotnik točke urejene v obratni smeri urinega kazalca, notranji mnogokotniki pa v smeri urinega kazalca. Smer točk mnogokotnika preverjamo s funkcijo `is_clockwise(points)`, ki se nahaja v datoteki `ear_clipping.py`. Vrstni red točk po potrebi obračamo v funkciji `reverse_direction(self)` razreda `Polygon`.

Slika 3.3 prikazuje primer mnogokotnika z eno luknjo. Seznam točk zunanjega mnogokotnika na sliki je `[0, 1, 2, 3, 4, 5, 6, 7]`, seznam točk `[8, 9, 10, 11]` pa predstavlja mnogokotnik luknje. Zunanji mnogokotnik združimo z notranjim, tako da dodamo dva nova sovpadajoča robova. Nova robova potekata med točkama zunanjega in notranjega mnogokotnika, med katerima ni nobenega drugega roba. Algoritem za iskanje primernih točk smo implementirali iz dokumentacije za *Ear clipping* algoritem [7]. Algoritem na notranjem mnogokotniku izbere točko z največjo koordinato `x`. Na zunanjem mnogokotniku pa točko, ki je najbližje tej točki, če med njima ni nobenega roba. Pri mnogokotniku s slike 3.3 bi algoritem vrnil točki `4` in `9`.

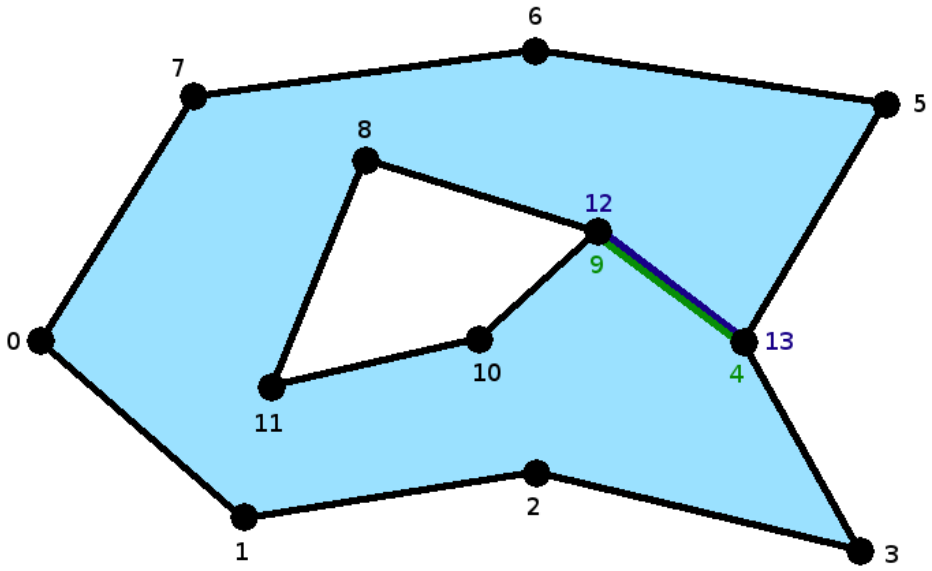


Slika 3.3: Primer mnogokotnika z luknjo

Zunanji mnogokotnik se nato združi z notranjim tako, da se med najdenima točkama doda nova sovpadajoča robova. Seznamu točk zunanjega mnogokotnika se za točko `4` vrinejo točke notranjega mnogokotnika in kopiji točk `4` in `9` Tako združen mnogokotnik je prikazan na sliki 3.4. Seznam točk združenega mnogokotnika je enak `[0, 1, 2, 3, 4, 9, 10, 11, 8, 12, 13, 5, 6, 7]`.

3.2.4 Triangulacija mnogokotnikov

Triangulacijo mnogokotnika prereza opravimo z *Ear clipping* algoritmom [7], ki smo ga implementirali v datoteki `ear_clipping.py` po opisu iz dokumentacije [7]. Slabost enostavne implementacije *Ear clipping* algoritma je razmeroma slaba časovna zahtevnost $O(n^3)$ [7]. Z implementacijo optimiziranega algoritma, ki je opisan v dokumentaciji *Ear clipping* algoritma [7], smo časovno zahtevnost zmanjšali na $O(n^2)$ [7]. Algoritem je implementiran



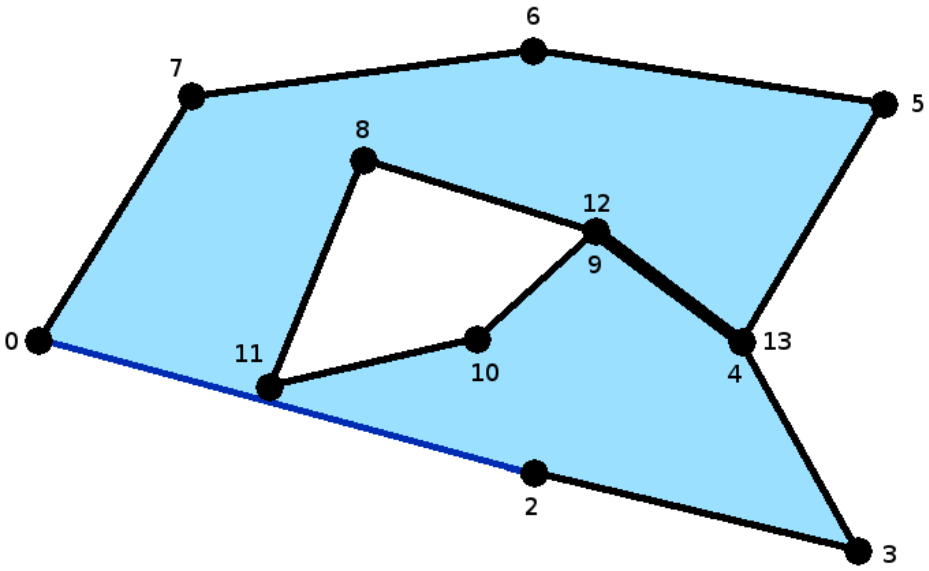
Slika 3.4: Mnogokotnik je združen z luknjo z dodatnima robovoma

v funkciji `triangulate_polygon()`.

Ear clipping algoritem triangulira mnogokotnik tako, da postopoma išče "ušesa"mnogokotnika. Uho mnogokotnika je trikotnik, ki je sestavljen iz treh zaporednih točk `t1, t2, t3`, za katere velja:

- kot v točki `t2` je manjši od 180°,
- rob med točkama `t1` in `t3` leži znotraj mnogokotnika,
- nobena druga točka mnogokotnika ne leži znotraj tega trikotnika.

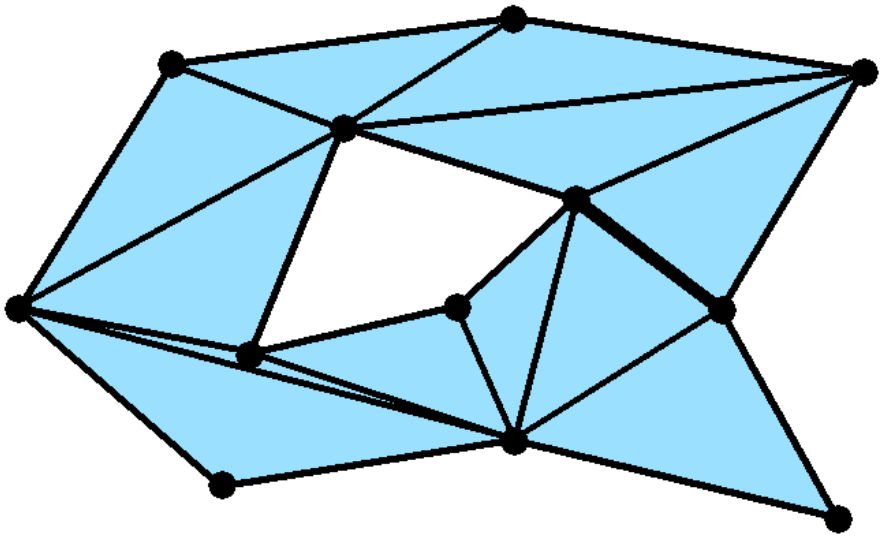
Tak trikotnik se odstrani iz mnogokotnika tako, da se točko `t2` odstrani iz seznama točk. Postopek se nato ponavlja, dokler mnogokotnik ne vsebuje manj kot 3 točke. Na sliki 3.5 je prikazan mnogokotnik iz slike 3.4 po prvem koraku algoritma. Slika 3.6 prikazuje končni rezultat triangulacije mnogokotnika s slike 3.6.



Slika 3.5: Mnogokotnik po prvem koraku *Ear clipping* algoritma

Testirali smo tudi *Seidelov* algoritem za triangulacijo mnogokotnikov, ki ima veliko boljšo časovno zahtevnost $O(n \log * n)$ [26]. Implementacija *Seidelovega* algoritma je težavna, zato smo uporabili implementacijo v datoteki `seidel.py` iz odprtokotne knjižnice poly2trap [6]. Uporabljena implementacija pa žal ne podpira triangulacije mnogokotnikov z luknjami. Med testiranjem smo odkrili tudi težave pri triangulaciji nekaterih robnih primerih. Zaradi teh težav se v končni rešitvi pri triangulaciji privzeto uporablja *Ear clipping* algoritem. Dodali pa smo tudi možnost izbire *Seidelovega* algoritma.

Funkcija `triangulate()` podpira uporabo obeh algoritmov. Funkcija po končani triangulaciji mnogokotnikov vrne seznam trikotnikov, ki sestavljajo površino prereza. Vrnjen seznam se nato doda seznamoma trikotnikov površine 3D modela nad in pod rezom. S tem je končan postopek rezanja 3D modela.



Slika 3.6: Mnogokotnik po koncu triangulacije

3.3 Razrez 3D modela na poljubno velike dele

V poglavjih 3.1 in 3.2 smo opisali postopek rezanja 3D modela na dva dela po z osi. Željeni končni rezultat naloge je razrez 3D modela na večje število primerno velikih delov, kar dosežemo s funkcijo `grid_cut_to_segments(max_x, max_y, max_z)` razreda `Model`.

Funkcija `grid_cut_to_segments(max_x, max_y, max_z)` razreže poljubno velik 3D model v primerno velike segmente 3D mreže. Velikost posameznega segmenta je omejena s parametri `max_x`, `max_y` in `max_z`. Postopek rezanja se začne z izračunom velikosti posameznega segmenta v vsaki izmed treh osi.

Rezanje 3D modela se izvede s funkcijo `cut_on_z_to_grid(segment_size, start_position)`, ki 3D model razreže po osi z na velikost določeno s parametrom `segment_size`.

Razrez vzdolž osi x dosežemo s predhodno rotacijo 3D modela okoli y osi za 90 stopinj. Po razrezu s funkcijo `cut_on_z_to_grid(segment_size,`

Slika 3.7: Primer razreza 3D modela reliefnega globusa zemlje na osem delov

`start_position`) posamezne segmente povrnemo v prvotno orientacijo z rotacijo okoli y osi za -90 stopinj. Razrez vzdolž y osi pa dosežemo s predhodno rotacijo okoli osi x za -90 stopinj. Po končanem razrezu posamezne segmente povrnemo v prvotno orientacijo z rotacijo okoli osi x za 90 stopinj.

Funkcija `grid_cut_to_segments(max_x, max_y, max_z)` po končanih rezih v x, y in z osi vrne tridimenzionalen seznam `xyz_segments[x][y][z]`, ki vsebuje posamezne segmente. Indeksi `x`, `y` in `z` predstavljajo pozicijo posameznega segmenta glede na prvoten 3D model. Na sliki 3.7 je prikazan rezultat razreza 3D modela s funkcijo `grid_cut_to_segments()`.

Poglavje 4

Predstavitev programa stl_split

V tem poglavju najprej opišemo uporabniški vmesnik in predstavili delovanje razvitega programa `stl_split`. V nadaljevanju opišemo postopek meritve hitrosti programa in predstavimo rezultate njene analize.

4.1 Uporabniški vmesnik

Z uporabo knjižnice `click` [21] smo razvili enostaven vmesnik, ki preko ukazne vrstice upravlja program `stl_split`. Za delovanje moramo programu podati pot do želene datoteke STL ter največjo velikost posameznega segmenta v oseh x, y in z. Program `stl_split` po končanem izvajanju shrani segmente prvotnega 3D modela v več datotek tipa STL.

Program `stl_split` nam s podajanjem opcijskega argumenta `--help` vrne navodila za uporabo v angleškem jeziku:

```
$ ./stl_split --help
Usage: stl_split [OPTIONS] FILE_PATH

  Cut the model in a STL file to segments of a specific size.

Options:
  --segment_x FLOAT      Output segment max size in the X axis.
```

```
--segment_y FLOAT       Output segment max size in the Y axis.
--segment_z FLOAT       Output segment max size in the Z axis.
--resize_to_x FLOAT     Resize the full model to the specified
                        size in the X axis.
--resize_to_y FLOAT     Resize the full model to the specified
                        size in the Y axis.
--resize_to_z FLOAT     Resize the full model to the specified
                        size in the Z axis.
--triangulation [ear_clipping|seidel]
                        (default: ear_clipping) Select which
                        algorithm to use for triangulation:
                        ear_clipping: slow, but works on all
                        types of models.
                        seidel: very fast algorithm, but buggy
                        and it fails to correctly triangulate
                        some models.
--save_path PATH        Save directory path. Segment files will
                        be saved in this directory. If not set,
                        files will be saved in same directory
                        as original model. Directory path will
                        be created if it does not exist yet.
--help                  Show this message and exit.
```

Obvezni argumenti programa so:

- `file_path` - pot do 3D modela, ki je shranjen v datoteki STL.
- `segment_x` - največja velikost končnega segmenta 3D modela v osi x.
- `segment_y` - največja velikost končnega segmenta 3D modela v osi y.
- `segment_z` - največja velikost končnega segmenta 3D modela v osi z.

V primeru, ko se treh velikosti segmenta ne poda ob zagonu programa, se le-te zahtevajo pred rezanjem 3D modela.

Z obveznimi argumenti program razreže izbrani 3D model na primerno število manjših segmentov ter jih shrani.

Z opcijskimi argumenti lahko prilagodimo izvajanje programa:

- `resize_to_x` - pred rezanjem se izbranemu 3D modelu spremeni velikost do izbrane velikosti v osi x.
- `resize_to_y` - pred rezanjem se izbrani 3D modelu spremeni velikost do izbrane velikosti v osi y.
- `resize_to_z` - pred rezanjem se izbrani 3D modelu spremeni velikost do izbrane velikosti v osi z.
- `triangulation` - izberemo algoritem triangulacije. Izbiramo lahko med `ear_clipping` ali `seidel`, privzet je prvi.
- `save_path` - pot, kamor se shranijo datoteke razrezanih segmentov. Privzeto se shranijo v isti imenik kot izvorna datoteka.

Spodaj je prikazan primer izvajanja programa:

```
$ ./stl_split --resize_to_z=50 Cylinder100.stl
Enter segment size on the X axis: 50
Enter segment size on the Y axis: 50
Enter segment size on the Z axis: 25
Loading file stl/Cylinder100.stl
Loaded model size X=40.0mm, Y=40.0mm, Z=40.0mm.
Resizing model to Z=50.0
Resized model size X=50.0mm, Y=50.0mm, Z=50.0mm.
Max segment size X: 50.0, Y: 50.0, Z: 25.0
Save path is empty. setting to original file directory
Save path: /home/gasper/PycharmProjects/stlModelSplitter/stl
Cutting model to segments:
Saving segment to /home/stlModelSplitter/Cylinder100_0_0_0.stl
Saving segment to /home/stlModelSplitter/Cylinder100_0_0_1.stl
```

V zgornjem primeru se 3D model, shranjen v datoteki `Cylinder100.stl` pred rezanjem, poveča do končne velikosti 50 mm v osi z. Nato se 3D model razreže na 2 segmenta dimenzije `x=50mm`, `y=50mm` in `z=25mm`. Datoteke shranjenih segmentov prevzamejo ime izvorne datoteke. Imenu izvorne datoteke se doda še pripona v obliki `_X_Y_Z`, ki označi položaj segmenta v koordinatnem sistemu izvornega 3D modela. V zgornjem primeru je v datoteki `Cylinder100_0_0_0.stl` shranjena spodnja polovica izvornega 3D modela, v datoteki `Cylinder100_0_0_1.stl` pa zgornja polovica izvornega 3D modela.

4.2 Meritev hitrosti programa

4.2.1 Opis postopka meritve

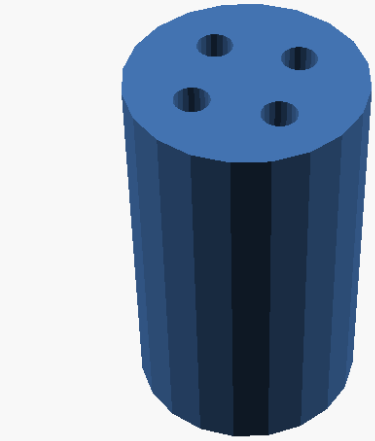
Za izdelani program smo želeli preveriti hitrost delovanja. V ta namen smo pripravili skripto `TestModels.scad` s parametričnim 3D modelirnim programom OpenSCAD [13]. Skripta izdela testni 3D model glede na podana parametra. Prvi parameter (`points`) določi, kakšno bo skupno število točk na rezu 3D modela. Drugi parameter (`holes`) pa določi število lukenj v površini prereza reza. S skripto smo nato izdelali večje število testnih 3D modelov različnih zahtevnosti in ločljivosti. Na slikah 4.1 in 4.2 sta primera izdelanih testnih 3D modelov.

Skripta `perf_measure.py` vsak posamezen testni 3D model testira tako, da ga razreže na pol ter med rezanjem shranjuje čase izvajanja posameznih operacij. Po končanem testiranju se rezultati shranijo v datoteko. Rezultate smo nato analizirali v programu Google Sheets [10].

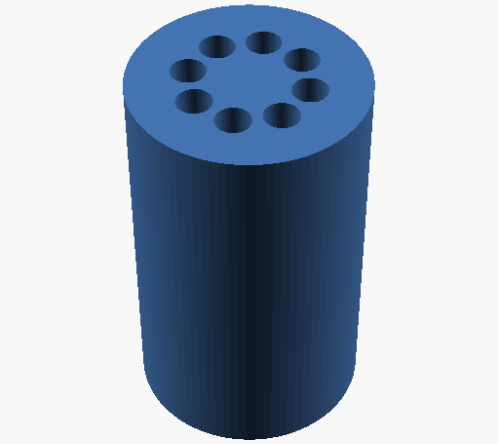
Program smo testirali na računalniku Dell Inspiron 15R 7520 z naslednjimi specifikacijami:

- Procesor: Intel Core i7-3612QM 2.1 GHz
- Pomnilnik: 8GB
- Trdi disk: SSD Samsung 840 Evo

- Operacijski sistem: Linux Mint 18 Cinnamon 64 bit
- Python različica: Python 2.7.12



Slika 4.1: Testni 3D model s štirimi luknjami in 200 točkami na prerezu



Slika 4.2: Testni 3D model z osmimi luknjami in 1600 točkami na prerezu

4.2.2 Analiza meritev

Pri analizi meritev nas zanima vpliv zahtevnosti 3D modela na čas izvajanja posameznih korakov rezanja 3D modela. Čas nalaganja datoteke smo izvzeli iz analize. Tabela 4.1 prikazuje čase izvajanja v posameznih korakih rezanja testnega 3D modela glede na število točk ter število lukenj v mnogokotniku prereza 3D modela. Na sliki 4.3 je graf primerjave porabljenega časa po posameznih korakih.

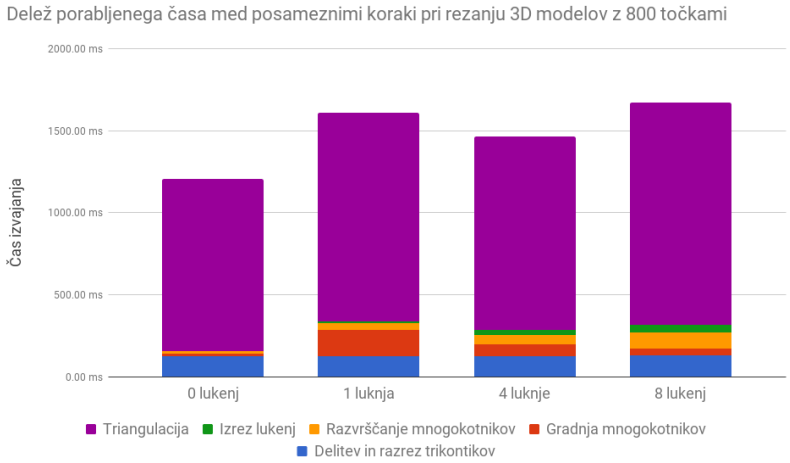
Iz table 4.1 in grafa na sliki 4.3 je razvidno, da program porabi največ časa pri triangulaciji z *Ear clipping* algoritmom. Čas triangulacije raste eksponentno s številom točk mnogokotnika. Mnogokotnikovo število lukenj pa

Št. lukenj	Št. točk	Delitev in razrez trikotnikov	Gradnja mnogokotnikov	Razvrščanje mnogokotnikov	Izrez lukenj	*Ear clipping* triangulacija
0	200	31.52 ms	3.48 ms	3.25 ms	0.11 ms	66.20 ms
	800	125.70 ms	16.04 ms	13.82 ms	0.48 ms	1049.23 0ms
	1600	249.87 ms	39.39 ms	25.60 ms	0.95 ms	4061.80 ms
	3200	500.55 ms	112.52 ms	50.90 ms	1.77 ms	16762.26 ms
1	200	31.48 ms	10.24 ms	7.39 ms	2.92 ms	81.55 ms
	800	125.03 ms	159.22 ms	39.79 ms	12.14 ms	1271.85 ms
	1600	268.17 ms	668.82 ms	108.39 ms	24.71 ms	5030.64 ms
	3200	499.44 ms	2022.68 ms	376.30 ms	44.46 ms	19611.37 ms
4	200	32.94 ms	6.24 ms	13.29 ms	7.43 ms	81.52 ms
	800	124.73 ms	70.59 ms	59.52 ms	28.61 ms	1182.12 ms
	1600	248.13 ms	268.95 ms	133.92 ms	57.50 ms	5194.22 ms
	3200	506.96 ms	1053.67 ms	356.51 ms	113.78 ms	19931.43 ms
8	200	31.51 ms	4.56 ms	24.81 ms	12.11 ms	93.12 ms
	800	128.39 ms	44.73 ms	94.76 ms	45.97 ms	1358.56 ms
	1600	263.85 ms	155.84 ms	201.24 ms	92.31 ms	5386.10 ms
	3200	512.99 ms	581.93 ms	458.44 ms	180.16 ms	21990.55 ms

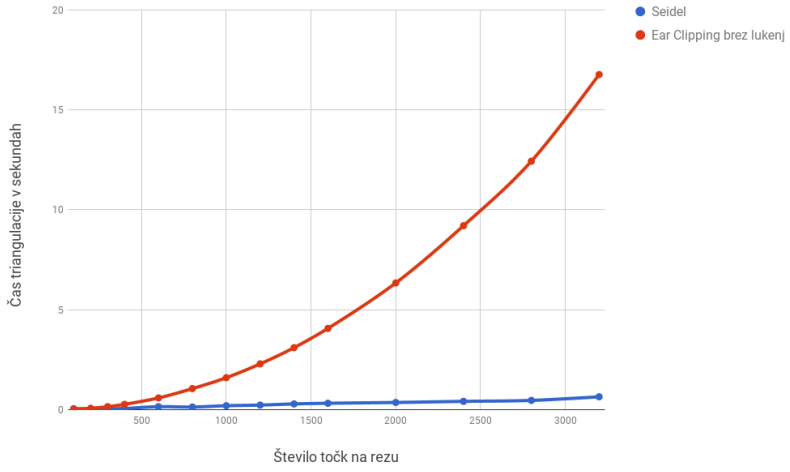
Tabela 4.1: Časi izvajanja posameznih korakov rezanja testnih 3D modelov

ne vpliva bistveno na čas triangulacije. S 3D modeli brez lukenj smo testirali tudi triangulacijo s *Seidel* algoritmom. Na sliki 4.4 je graf primerjave hitrosti izvajanja *Seidel* algoritma z *Ear clipping* algoritmom. *Seidel* triangulacijski algoritem je mnogo hitrejši pri 3D modelih z višjo ločljivostjo, žal pa ni uporaben pri 3D modelih z luknjami.

V poglavju 3.2.1 smo opisali gradnjo mnogokotnikov iz seznama trikotnikov na robu prereza. V tabeli 4.1 lahko razberemo, da čas gradnje mnogokotnikov raste eksponentno s številom točk na prerezu. Iz grafa na sliki 4.5 je razvidno da je čas gradnje mnogokotnikov močno odvisen tudi od števila lukenj na prerezu, saj število lukenj posredno določa število mnogokotnikov na prerezu. Večje število lukenj tako pomeni, da je vsak posamezni mnogoko-



Slika 4.3: Graf primerjave časa izvajanja posameznih korakov triangulacije

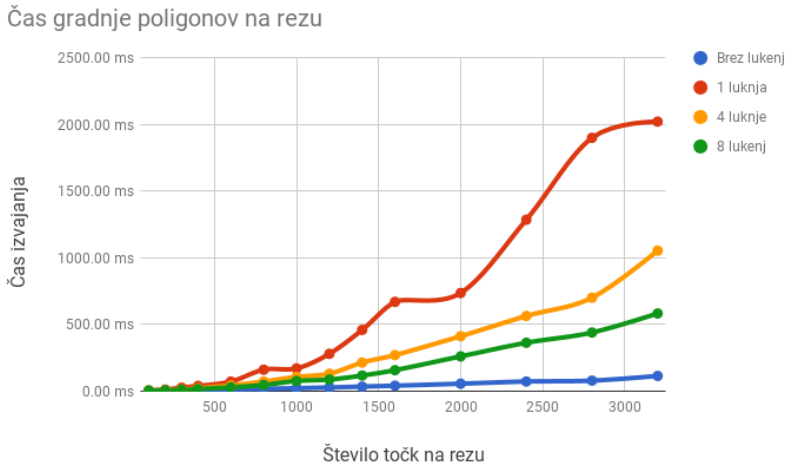


Slika 4.4: Graf primerjave časa izvajanja med *Seidel* in *Ear clipping* algoritmoma za triangulacijo

tnik sestavljen iz manjšega števila točk in je zato hitreje zgrajen. To je tudi razlog, da je gradnja mnogokotnikov pri testnih modelih z večjim številom lukenj hitrejša kot pri testnem modelu z eno samo luknjo. Testni model s 3200 točkami in 1 luknjo ima na prerezu 2 mnogokotnika, ki sta sestavljena

iz 1600 točk. Pri testnem modelu s 3200 točkami in 4 luknjami pa ima na prerezu 5 mnogokotnikov s po 640 točkami.

Izjema je testni model brez lukenj, ki bi praviloma moral imeti najdaljši čas izvajanja, a ima daleč najboljši čas gradnje mnogokotnikov. Razlog za to je (po vsej verjetnosti) začetni razpored robnih trikotnikov v seznamu. Če si sosednji trikotniki v seznamu sledijo eden za drugim, bo za gradnjo mnogokotnika potreben samo en prehod čez seznam.



Slika 4.5: Graf časa gradnje mnogokotnikov glede na število lukenj in točk na prerezu

Časi izvajanja ostalih korakov: delitev in razrez obstoječih trikotnikov, razvrščanje mnogokotnikov ter izrez lukenj naraščajo linearno s številom točk. Število lukenj na prerezu ne vpliva na čas delitve in razreza obstoječih trikotnikov. Število lukenj na prerezu vpliva linearno na čas razvrščanja mnogokotnikov in izrez lukenj.

Slaba časovna zahtevnost triangulacije mnogokotnikov se v praksi pri večini 3D modelov ne izkaže za preveč problematično. 3D model reliefnega globusa zemlje s slike 3.7, ki je shranjen v datoteki z velikostjo 5.1 megabajtov, se na osem delov razreže v slabih 17 sekundah.

Poglavje 5

Sklepne ugotovitve

Cilj diplomske naloge je bil poenostavitev razreza 3D modela na manjše dele, ki se lahko natisnejo v omejeni površini 3D tiskalnika. Naloga je bila izvedena z razvojem programa `stl_split`, ki omogoča enostaven razrez poljubnega 3D modela na segmente želene velikosti. Uporabniški vmesnik ukazne vrstice je preprost za uporabo ter za delovanje od uporabnika potrebuje le osnovne parametre.

Program z uporabo *Ear clipping* algoritma triangulacije pravilno reže tudi zahtevne 3D modele, ki lahko vsebujejo luknje v prerezu ter nove površine znotraj teh lukenj. Podpora za tako zahtevne 3D modele je bila težavna, vendar potrebna za robustno delovanje. Zaradi slabe časovne zahtevnosti algoritma *Ear clipping* je čas rezanja 3D modelov višje ločljivosti lahko zelo dolg, zato smo programu dodali tudi možnost uporabe odprtokodne implementacije *Seidelovega* algoritma triangulacije. Uporabljena implementacija *Seidelovega* algoritma deluje veliko hitreje, vendar bolj nezanesljivo in le za enostavne 3D modela. V prihodnje bomo iskali boljšo rešitev za triangulacijo mnogokotnikov.

V poglavju 4.2.2 smo z analizo meritev hitrosti programa ugotovili, da čas rezanja 3D modela hitro narašča z njegovo ločljivostjo. Glavni razlog za to je eksponentna rast časa triangulacije in gradnje mnogokotnikov z večanjem števila točk na prerezu. Velika izboljšava hitrosti bi bila implementacija

hitrejšega algoritma za triangulacijo, ki bi deloval tudi na mnogokotnikih z luknjami.

Prava smer za pospešitev triangulacije bi lahko bila nadgradnja odprtokotne implementacije *Seidel* algoritma iz knjižnice `poly2trap` [6]. Vendar ima trenutno uporabljena implementacija težave in ne deluje pri vseh mnogokotnikih, zato je možno, da bi lahko bilo odpravljanje težav zamudnejše od sveže implementacije *Seidelovega algoritma*.

Trenutni uporabniški vmesnik nam omogoča enostaven razrez večjega 3D modela na segmente, ki se lahko natisnejo v manjši delovni prostornini 3D tiskalnika. Od uporabnika program zahteva le osnovne parametre ter vhodno datoteko. Pomanjkljivost je pomanjkanje grafičnega predogleda končnega rezultata ter možnost ročne postavitve ravnine reza. Izdelava grafičnega vmesnika z možnostjo ročne postavitve rezov bi gotovo izboljšala uporabnost programa.

Program podpira delo le s 3D modeli shranjenimi v formatu STL. Program je namenjen za uporabo na področju 3D tiska, kjer je STL najpogosteje uporabljeni podatkovni format. Podpora za večje število formatov bi tako tudi bila možna izboljšava, vendar to ne bi toliko vplivalo na boljšo uporabnost kot grafični vmesnik ali menjava triangulacijskega algoritma.

Koristna nadgradnja bi bila tudi možnost dodajanja elementov za olajšanje poravnave posameznih kosov. Na prerezih 3D modela se lahko posameznim delom dodajo zatiči in utori, ki bi zagotovili pravilno poravnavo pri sestavljanju končnega izdelka.

Literatura

[1] Airbus. Pioneering bionic 3d printing. Dosegljivo: http://www.airbusgroup.com/int/en/story-overview/Pioneering-bionic-3D-printing.html. [Dostopano 25. 4. 2017].

[2] Amazing. Am basics. Dosegljivo: http://additivemanufacturing.com/basics/. [Dostopano 25. 4. 2017].

[3] Autodesk. AutoCAD naslovna stran. Dosegljivo: https://www.autodesk.eu/products/autocad/overview. [Dostopano 12. 1. 2018].

[4] Autodesk. Tinkercad naslovna stran. Dosegljivo: https://www.tinkercad.com. [Dostopano 12. 1. 2018].

[5] Cults3d. Cults naslovna stran. Dosegljivo: https://cults3d.com. [Dostopano 12. 1. 2018].

[6] deparkes. Trapezoidal decomposition of polygons in python. Dosegljivo: https://github.com/deparkes/poly2trap. [Dostopano 1. 11. 2017].

[7] David Eberly. Triangulation by ear clipping. Dosegljivo: https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf. [Dostopano 29. 5. 2017].

[8] Formlabs. Cults naslovna stran. Dosegljivo: https://pinshape.com. [Dostopano 12. 1. 2018].

[9] Blender Foundation. Blender naslovna stran. Dosegljivo: https://www.blender.org. [Dostopano 12. 1. 2018].

[10] Google. Google sheets naslovna stran. Dosegljivo: https://docs.google.com/spreadsheets. [Dostopano 15. 1. 2018].

[11] The Guardian. 3d-3d-printed prosthetic limbs: the next revolution in medicine. Dosegljivo: https://www.theguardian.com/technology/2017/feb/19/3d-printed-prosthetic-limbs-revolution-in-medicine. [Dostopano 25. 4. 2017].

[12] 3D Hubs. Automotive 3d printing applications. Dosegljivo: https://www.3dhubs.com/knowledge-base/automotive-3d-printing-applications. [Dostopano 25. 4. 2017].

[13] Marius Kintel. OpenSCAD naslovna stran. Dosegljivo: http://www.openscad.org. [Dostopano 12. 1. 2018].

[14] Heidi Ledford. Printed body parts come alive. *Nature*, 520:273, 2015. [Dostopano 30. 1. 2018]. URL: https://www.nature.com/polopoly_fs/1.17320!/menu/main/topColumns/topLeftColumn/pdf/520273a.pdf.

[15] Additively Ltd. Fused deposition modeling (fdm). Dosegljivo: https://www.additively.com/en/learn-about/fused-deposition-modeling. [Dostopano 29. 5. 2017].

[16] Additively Ltd. Laser melting (lm). Dosegljivo: https://www.additively.com/en/learn-about/laser-melting. [Dostopano 29. 5. 2017].

[17] Additively Ltd. Stereolithography (sl). Dosegljivo: https://www.additively.com/en/learn-about/stereolithography. [Dostopano 29. 5. 2017].

[18] MakerBot. Thingiverse naslovna stran. Dosegljivo: https://www.thingiverse.com. [Dostopano 12. 1. 2018].

[19] G. H. Meisters. Polygons have ears. *The American Mathematical Monthly*, 82(6):648–651, 1975. [Dostopano 30. 1. 2018]. URL: http://www.jstor.org/stable/2319703.

[20] MyMiniFactory. MyMiniFactory naslovna stran. Dosegljivo: https://www.myminifactory.com. [Dostopano 12. 1. 2018].

[21] The Pallets Projects. Click. Dosegljivo: http://click.pocoo.org/5/. [Dostopano 5. 11. 2017].

[22] Alessandro Ranellucci. Slic3r naslovna stran. Dosegljivo: http://slic3r.org. [Dostopano 12. 1. 2018].

[23] RepRap. Spletna stran reprap.org. Dosegljivo: http://reprap.org/wiki/About. [Dostopano 25. 4. 2017].

[24] Jürgen Riegel, Werner Mayer, and Yorik van Havre. FreeCAD naslovna stran. Dosegljivo: https://www.freecadweb.org. [Dostopano 12. 1. 2018].

[25] Science. 3d-printed 'hyperelastic bone' could be the future of reconstructive surgery. Dosegljivo: http://www.sciencemag.org/news/2016/09/print-demand-bone-could-quickly-mend-major-injuries. [Dostopano 25. 4. 2017].

[26] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry*, 1(1):51–64, 1991. [Dostopano 30. 1. 2018]. URL: https://pdfs.semanticscholar.org/3ccb/93e1b9803839efafffa51a85bffceb715010.pdf.

[27] Simplify3D. Simplify3D naslovna stran. Dosegljivo: https://www.simplify3d.com. [Dostopano 12. 1. 2018].

[28] SpaceX. Spacex launches 3d printed part to space, creates printed engine chamber. Dosegljivo: http://www.spacex.com/news/2014/07/31/spacex-launches-3d-printed-part-space-creates-printed-engine-chamber-crewed. [Dostopano 25. 4. 2017].

[29] Trimble. SketchUp naslovna stran. Dosegljivo: https://www.sketchup.com. [Dostopano 12. 1. 2018].

[30] Ultimaker. Ultimaker cura naslovna stran. Dosegljivo: https://ultimaker.com/en/products/ultimaker-cura-software. [Dostopano 12. 1. 2018].

[31] WhiteClouds. Laminated object manufacturing (lom). Dosegljivo: https://www.whiteclouds.com/3dpedia-index/laminated-object-manufacturing-lom. [Dostopano 18. 5. 2017].

[32] Wikipedia. Wikipedia stran za STL datotečni format. Dosegljivo: https://en.wikipedia.org/wiki/STL_(file_format). [Dostopano 12. 1. 2018].